AN EVOLUTIONARY MACHINE LEARNING FRAMEWORK
FOR BIG DATA SEQUENCE MINING

by

Uday Krishna Kamath
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

___________________________  Dr. Kenneth De Jong, Dissertation Director

___________________________  Dr. Amarda Shehu, Committee Member

___________________________  Dr. Carlotta Domeniconi, Committee Member

___________________________  Dr. Tomasz Arciszewski, Committee Member

___________________________  Dr. Stephen Nash, Senior Associate Dean

___________________________  Dr. Kenneth S. Ball, Dean, Volgenau School
    of Engineering

Date: ________________________  Spring Semester 2014
    George Mason University
    Fairfax, VA
An Evolutionary Machine Learning Framework for Big Data Sequence Mining

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Uday Krishna Kamath
Master of Science
University of North Carolina at Charlotte, 1999
Bachelor of Engineering
University of Bombay, 1996

Director: Dr. Kenneth De Jong, Professor
Department of Computer Science

Spring Semester 2014
George Mason University
Fairfax, VA
Dedication

I dedicate this dissertation to my son Aaroh Kamath and to all other awesome kids with Autism.
Acknowledgments

First, I would like to thank my advisor Dr. Kenneth De Jong for his guidance and support during my doctoral study. Having a renowned expert and authority in evolutionary computation as my advisor made the pursuit of research really fruitful and memorable. I would like to thank Dr. Amarda Shehu for her continuous encouragement, guidance and for granting me the benefit of her expertise in computational biology without which much of my early research would not have gotten published. I want to thank Dr. Carlotta Domeniconi, for being one of the most approachable professors in the department and for many long discussions on theoretical and statistical techniques over several months, which formed a strong basis for my theoretical proofs. I also want to thank Dr. Tomasz Arciszewski for readily agreeing to be part of the committee and for showing me the connections between past and current art and this work. I also want to thank all faculty and staff of the Computer Science department, especially Dr. Sean Luke, for his ECJ software and the help he provided towards implementation of the software for this work.

On the personal front, I first want to thank my father Late Kokkarne Krishna Kamath and my mother Bharathi Kamath, for instilling the love of learning and for being the biggest encouragement in my life. I want to thank my parents-in-law M.L.N Shenoy and Kalavathy Shenoy for the support they gave me, especially during my qualifiers. I want to thank my brother Vivek Kamath and his family for their support. I want to thank Dr. Reżarta Islamaj Doğan, Jack Compton, Johan Kaers, Krishna Choppella, Steve Peter, and Dr. Sarang Kayande for helping me in some of the publications and for the literature reviews. I want to thank my employer Detica, for giving me flexibility in work hours and travel during my entire doctoral study. I want to thank all my relatives, friends, teachers and professors who in every way have contributed to my pursuit.

Most importantly, I want to thank my wife Pratibha, who was responsible for coaxing me to pursue my love for learning by making me register in the doctoral program. Without her taking care of most of the daily chores and helping me concentrate on my research, especially during some tough times, this dissertation would never have seen the light of day. I would also like to thank our dog Brandy, who was literally beside me during the long hours as a great comfort.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td></td>
<td>viii</td>
</tr>
<tr>
<td>List of Figures</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Abstract</td>
<td></td>
<td>xiii</td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Background and Related Work</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>Sequence Classification</td>
<td>8</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Feature-Based Classification</td>
<td>8</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Similarity-Based Classification</td>
<td>9</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Statistical Model-Based Classification</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Big Data Machine Learning</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Boosting and Large Margin Classifiers</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Evolutionary Computation</td>
<td>12</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Genetic Programming</td>
<td>13</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Spatially-structured evolutionary algorithms (SSEAs)</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Evolutionary Feature Generator (EFG)</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Input-Feature Space Transformation</td>
<td>15</td>
</tr>
<tr>
<td>3.2</td>
<td>EFG as an EA</td>
<td>17</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Feature Representation</td>
<td>17</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Genetic Operators</td>
<td>23</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Bloat Control</td>
<td>27</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Population and Generation Mechanism</td>
<td>27</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Fitness Function</td>
<td>28</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Hall of Fame</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Discussion</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>EFG for Hypersensitive site and Alu sequence classification</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Background and Introduction</td>
<td>31</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Hypersensitive Sites Recognition</td>
<td>31</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Alu Elements Predictions</td>
<td>33</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>4.2</td>
<td>EFG</td>
<td>34</td>
</tr>
<tr>
<td>4.2.1</td>
<td>EFG algorithm</td>
<td>35</td>
</tr>
<tr>
<td>4.2.2</td>
<td>EFS Algorithm</td>
<td>35</td>
</tr>
<tr>
<td>4.3</td>
<td>Datasets</td>
<td>41</td>
</tr>
<tr>
<td>4.3.1</td>
<td>HSS Dataset</td>
<td>41</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Alu Dataset</td>
<td>41</td>
</tr>
<tr>
<td>4.4</td>
<td>Experiments</td>
<td>41</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Cross-validation on Training Data Sets for HSS</td>
<td>42</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Cross-validation on Training Data Sets for Alu Sequences</td>
<td>42</td>
</tr>
<tr>
<td>4.5</td>
<td>Feature Analysis</td>
<td>43</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Statistical Analysis of Features</td>
<td>43</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Biological Analysis of Features</td>
<td>46</td>
</tr>
<tr>
<td>4.6</td>
<td>Conclusion</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>EFG for Promoter Region Identification</td>
<td>49</td>
</tr>
<tr>
<td>5.1</td>
<td>Promoter Region Identification</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>EFG Algorithm</td>
<td>51</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Feature Representation</td>
<td>51</td>
</tr>
<tr>
<td>5.2.2</td>
<td>EFG Algorithm</td>
<td>51</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Post EFG Feature Selection</td>
<td>51</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Support Vector Machines as Classifier</td>
<td>52</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Implementation Details</td>
<td>52</td>
</tr>
<tr>
<td>5.3</td>
<td>Datasets</td>
<td>52</td>
</tr>
<tr>
<td>5.4</td>
<td>Experiments</td>
<td>53</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Cross-validation on Training Datasets</td>
<td>54</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Evaluating Performance on Testing Datasets</td>
<td>55</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Comparison with State-of-the-Art Methods</td>
<td>57</td>
</tr>
<tr>
<td>5.5</td>
<td>Feature Analysis</td>
<td>58</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Statistical Analysis of Features</td>
<td>58</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Biological Analysis of features</td>
<td>59</td>
</tr>
<tr>
<td>5.6</td>
<td>Conclusion</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>EFG for Splice Site Prediction</td>
<td>61</td>
</tr>
<tr>
<td>6.1</td>
<td>The DNA Splice Site Prediction Problem</td>
<td>61</td>
</tr>
<tr>
<td>6.2</td>
<td>The EFG Algorithm</td>
<td>63</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Post EFG Feature Selection</td>
<td>63</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Support Vector Machines as Classifier</td>
<td>63</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>9.1.2</td>
<td>Parameter and Sensitivity Analysis</td>
<td>108</td>
</tr>
<tr>
<td>9.1.3</td>
<td>Meta-learning Experiments</td>
<td>111</td>
</tr>
<tr>
<td>9.1.4</td>
<td>Scalability Experiments</td>
<td>112</td>
</tr>
<tr>
<td>9.1.5</td>
<td>Comparison Against AdaBoost and Impact of Noise</td>
<td>117</td>
</tr>
<tr>
<td>9.2</td>
<td>Conclusion</td>
<td>120</td>
</tr>
<tr>
<td>10</td>
<td>EML Framework and Validation</td>
<td>122</td>
</tr>
<tr>
<td>10.1</td>
<td>Introduction</td>
<td>123</td>
</tr>
<tr>
<td>10.2</td>
<td>EML Framework</td>
<td>123</td>
</tr>
<tr>
<td>10.2.1</td>
<td>Data Parallelization</td>
<td>123</td>
</tr>
<tr>
<td>10.2.2</td>
<td>Parallel EFG</td>
<td>124</td>
</tr>
<tr>
<td>10.2.3</td>
<td>Feature Combination and Selection</td>
<td>125</td>
</tr>
<tr>
<td>10.2.4</td>
<td>Parallel Feature Interpretation</td>
<td>126</td>
</tr>
<tr>
<td>10.2.5</td>
<td>PSBML</td>
<td>126</td>
</tr>
<tr>
<td>10.3</td>
<td>Experimental Validation</td>
<td>126</td>
</tr>
<tr>
<td>10.3.1</td>
<td>Validation of EML: Genome-wide Sequence Classification</td>
<td>127</td>
</tr>
<tr>
<td>10.3.2</td>
<td>Validation of EML: Brain Computer Interaction as Time Series</td>
<td>131</td>
</tr>
<tr>
<td>10.4</td>
<td>Conclusion</td>
<td>134</td>
</tr>
<tr>
<td>11</td>
<td>Conclusion and Future Work</td>
<td>135</td>
</tr>
<tr>
<td>11.1</td>
<td>Conclusion</td>
<td>135</td>
</tr>
<tr>
<td>11.2</td>
<td>Future Work</td>
<td>136</td>
</tr>
<tr>
<td>11.2.1</td>
<td>Algorithmic Enhancements</td>
<td>136</td>
</tr>
<tr>
<td>11.2.2</td>
<td>Future Applications</td>
<td>137</td>
</tr>
<tr>
<td>A</td>
<td>Appendix</td>
<td>139</td>
</tr>
<tr>
<td>A.1</td>
<td>Annotation of Splice Sites</td>
<td>139</td>
</tr>
<tr>
<td>A.2</td>
<td>PSBML Experiments</td>
<td>142</td>
</tr>
<tr>
<td>A.2.1</td>
<td>Scalability Comparisons</td>
<td>142</td>
</tr>
<tr>
<td>A.2.2</td>
<td>Accuracy Comparison on a Large Real-world Dataset</td>
<td>142</td>
</tr>
<tr>
<td>Bibliography</td>
<td>145</td>
<td></td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 A table of the non-terminals and terminal nodes employed by EFG.</td>
<td>18</td>
</tr>
<tr>
<td>3.2 Boolean simplification rules for EFG post-processing and improving the readability of the features.</td>
<td>30</td>
</tr>
<tr>
<td>4.1 IUPAC code, adapted from [129].</td>
<td>36</td>
</tr>
<tr>
<td>4.2 auROC (area under ROC curve) and auPRC (area under PRC curve) comparison with all methods for HSS</td>
<td>43</td>
</tr>
<tr>
<td>4.3 auROC (area under ROC curve) and auPRC (area under PRC curve) comparison with all methods for Alu Sequences</td>
<td>44</td>
</tr>
<tr>
<td>5.1 Cross-validation auROC comparison between K-MerSVM and EFG.</td>
<td>55</td>
</tr>
<tr>
<td>5.2 Comparison between K-Mer and EFG over testing data sets for Plant, Drosophila, and Homo Sapiens with suffixes P,D and H respectively.</td>
<td>56</td>
</tr>
<tr>
<td>5.3 Comparison between EFG and other methods over the plant testing dataset.</td>
<td>58</td>
</tr>
<tr>
<td>6.1 Comparison of different methods for NN269 Splicing Sequences</td>
<td>66</td>
</tr>
<tr>
<td>6.2 Comparison of 11ptAVG data. The average and standard deviation in column 4 are obtained over 30 EFG runs.</td>
<td>69</td>
</tr>
<tr>
<td>6.3 Comparison for Worm Splicing Sequences</td>
<td>74</td>
</tr>
<tr>
<td>6.4 IG sums of subsets of features evaluated over acceptor (top) and donor data (bottom).</td>
<td>75</td>
</tr>
<tr>
<td>7.1 Error Rate comparing EFG-SAX with the state-of-the-art algorithms</td>
<td>85</td>
</tr>
<tr>
<td>7.2 Error Rate comparing EFG-SAX with feature, statistical, and kernel methods</td>
<td>87</td>
</tr>
<tr>
<td>7.3 Comparison of EFG-SAX with different algorithms</td>
<td>90</td>
</tr>
<tr>
<td>8.1 Overlap percentage between support vectors and PSBML hard instances.</td>
<td>106</td>
</tr>
<tr>
<td>9.1 Performance measured as auROC with PSBML on different grid sizes given in the respective columns.</td>
<td>111</td>
</tr>
<tr>
<td>9.2 UCI datasets training size, testing size and dimensionality used in the experiments.</td>
<td>112</td>
</tr>
</tbody>
</table>
9.3 Meta-learning results (AUC) comparing the base classifiers and PSBML combined with the same. ........................................... 113
9.4 Training speed (in seconds) and accuracy for the Checkerboard and the Sine Wave datasets. ....................................................... 115
9.5 Memory with various data sizes, peak training memory, and standard deviation for the KDD Cup 1999 dataset. ................................. 117
9.6 Training speed in seconds, mis-classification, area under ROC, and PRC for the KDD Cup 1999 dataset. ....................................................... 117
9.7 Performance of AdaBoostM1 (DS: Decision Stump), AdaBoostM1 (NB: Naive Bayes) and PSBML (NB: Naive Bayes) with no, 10%, and 20% noise. ........ 121
10.1 Whole organism Genome training datasets size and true positives fractions in the total set. ................................................................. 130
10.2 Area Under ROC (A) and Area Under PRC (P) comparisons of various state of the art algorithms with EML framework on the Worm dataset for both acceptor and donor splice sites. ........................................... 130
10.3 Area Under ROC (ROC) and Area Under PRC (PRC) comparisons of various state of the art algorithms with EML framework on the Fly dataset for both acceptor and donor splice sites. ........................................... 130
10.4 Comparison of EML using SAX preprocessing with the other methodologies. 133
A.1 Training speed (in seconds) and accuracy for the Rastrigin and the Rosenbrock datasets. ................................................................. 143
A.2 Performance on NSL-KDD. ................................................................. 144
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>EML Framework with two sub-components EFG and PSBML</td>
<td>5</td>
</tr>
<tr>
<td>3.1</td>
<td>Input feature-space transformation that needs only linear separator in its high dimension.</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Composition (left) and sequences with those motifs (right) graphically represented.</td>
<td>19</td>
</tr>
<tr>
<td>3.3</td>
<td>Region-specific domain information to capture Regional Features.</td>
<td>20</td>
</tr>
<tr>
<td>3.4</td>
<td>Position (left) and sequences with those motifs (right) features graphically represented.</td>
<td>21</td>
</tr>
<tr>
<td>3.5</td>
<td>Positional with shift (left) and sequences with those motifs (right) features graphically represented.</td>
<td>22</td>
</tr>
<tr>
<td>3.6</td>
<td>Correlational (left) and sequences with those motifs (right) features graphically represented.</td>
<td>23</td>
</tr>
<tr>
<td>3.7</td>
<td>Conjunction features combining two positional and one compositional feature.</td>
<td>24</td>
</tr>
<tr>
<td>3.8</td>
<td>Disjunction features combining two positional and one negated positional feature.</td>
<td>25</td>
</tr>
<tr>
<td>3.9</td>
<td>Crossover Illustration.</td>
<td>26</td>
</tr>
<tr>
<td>3.10</td>
<td>Mutation Illustration.</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>DNase Hypersensitive sites in a genome [112]. Reproduced by the permission of copyright owner.</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Alu-subfamily Sx, showing younger sequences having more mutations progressively <a href="top">119</a>. The expansion of Alu subfamilies (Yc1, Ya5a2, Yb9, Yb8, Y, Sg1, Sx and J) is superimposed on a tree of primate evolution <a href="bottom">119</a>. Reproduced by the permission of copyright owner.</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>EFG algorithm with EFS algorithm</td>
<td>38</td>
</tr>
<tr>
<td>4.4</td>
<td>Information Gain Distribution for EFG-EFS features in comparison with other feature related methods like K-mers and Gibbs Sampling.</td>
<td>45</td>
</tr>
</tbody>
</table>
4.5 Information Gain Distribution for EFG-EFS features in comparison with other feature related methods like K-mers and Gibbs Sampling.

4.6 Sample showing a complex feature from the EFG algorithm for Alu sequence (top) and the clustal alignment showing the same overrepresented signals (bottom).

5.1 Promoter Region in DNA [137]. Reproduced with permission by copyright owner.

5.2 Precision versus recall for K-merSVM and EFG.

5.3 FPR versus recall (bottom) curves for K-merSVM (red) and EFG (blue).

5.4 Left: F-scores of top 95 features obtained by the EFG. Right: F-scores of 128 4-mer features employed in [30].

6.1 Showing Splicing from DNA to Protein [137]. Reproduced with permission of copyright owner.

6.2 Precision values, plotted over recall points (left: acceptor, right: donor). Values are averages over 30 EFG runs. Error bars are 95% confidence intervals of standard deviations.

6.3 Precision over recall (left: acceptor, right: donor), plotted for the B2hum testing data set.

6.4 FPR over recall (left: acceptor, right: donor), plotted for the B2hum testing data set.

6.5 Acceptor and donor prediction scores are shown on the left and right axis, respectively for the chromosome AB012229.

6.6 Acceptor and donor prediction scores are shown on the left and right axis, respectively for the chromosome AB020236.

6.7 Precision over recall (left: acceptor, right: donor), plotted for the 40K subset sampled from the worm training data set.

7.1 PAA transformation visualized as a linear combination of the box-basis function, reducing 128 long sequences to 8 dimensions [171].

7.2 PAA approximations with predetermined breakpoints mapped to symbols. In the example above, with $n = 128$, $w = 8$ and $a = 3$, the time series is mapped to the word baabccbc [171].

7.3 Timeseries of length 128, showing overlapping signals starting at different positions from p 1 to 8 [171].

7.4 The EFG algorithm with the SAX algorithm.
7.5 Showing Cylinder, Bell and Funnel as three classes from top to bottom as time series. ................................................................. 89
7.6 Showing complex patterns in a time series, that when translated as symbols can help the EFG capture positional, correlational, and compositional elements as motifs. .................................................. 91
8.1 Two dimensional grid with various neighborhood structures. ............ 93
8.2 Density estimation of one dimensional data based on the distance from the boundary. .......................................................... 97
8.3 Circle dataset: Data distribution at epochs 25 (Left) and 50 (Right) using PSBML and GMMs. ....................................................... 103
8.4 Changes in weight distribution as function of time: (Left) exponential decay; (Right) logistic increase. ........................................... 104
8.5 Linearly separable Gaussian dataset: Data distribution at epochs 25 (Left) and 50 (Right) using PSBML and GMMs. ..................... 105
8.6 (Left) Circle dataset; (Right) Bivariate Gaussian dataset. ............... 105
9.1 Results are shown on the UCI chess dataset. The error rate is shown as a function of the neighborhood structure in (a) and probability $p_r$ in (b). The number of distinct training instances is shown as a function of $p_r$ in (c). . . 110
9.2 Synthetic datasets: (Left) Sine wave; (Right) Checkerboard. .......... 114
9.3 Mean training times per epoch with varying dataset sizes. .............. 118
9.4 Mean training times per epoch with varying threads. .................... 119
9.5 Average Peak Memory with varying training data size .................. 120
10.1 The EML Framework with different components and interaction for classification. ................................................................. 124
A.1 Genome Splice Predictions for 45868_HSCDIR2 showing predictions and true sites ................................................................. 140
A.2 Genome Splice Predictions for 45378_AB005548 showing predictions and true sites ................................................................. 141
A.3 Genome Splice Predictions for 4546_AB016492 showing predictions and true sites ................................................................. 141
Abstract

AN EVOLUTIONARY MACHINE LEARNING FRAMEWORK FOR BIG DATA SEQUENCE MINING

Uday Krishna Kamath, PhD
George Mason University, 2014
Dissertation Director: Dr. Kenneth De Jong

Sequence classification is an important problem in many real-world applications. Unlike other machine learning data, there are no “explicit” features or signals in sequence data that can help traditional machine learning algorithms learn and predict from the data. Sequence data exhibits inter-relationships in the elements that are important in understanding and predicting future sequences. However, finding these relationships is proven to be an NP-hard problem. When we use naive enumerations of combinations of elements or “brute force” iterative approaches for defining these features they often result in poor predictions. Some algorithms which perform well in prediction lack transparency, i.e., the discriminating features generated by these methods are not easily identifiable.

In addition, the size of the sequence-based datasets presents practical challenges to most learning algorithms. Most sequence-based datasets contain millions or even billions of instances, for example, the genome-wide sequences of organisms in bioinformatics. At these sizes, classic learning algorithms often become prohibitively expensive, making scalability an important issue. Therefore, there is a need for an approach that can help find features/signals in complex sequences, offer meaningful discriminators, produce good predictions, and can scale well in time and space.
This dissertation addresses the above issues by designing a comprehensive approach in the form of the Evolutionary Machine Learner (EML) framework. This framework can be employed on sequence-based datasets to generate explicit, human-recognizable features while solving the scalability issue. EML framework consists of a novel EA-based feature generation (EFG) algorithm for automatic feature construction. By modeling four complex sequencing problems in bioinformatics and generating meaningful, human-understandable features with comparable or better accuracy than the state of the art algorithms, the power and usefulness of the EFG algorithm is demonstrated. The EFG algorithm is also validated by applying it to time series classification problems showing the generic nature of the algorithm in finding the important discriminating patterns that assist in modeling sequence-based data.

EML framework addresses the scalability issue by means of a novel, parallel scalable machine learning algorithm (PSBML) based on spatially structured evolutionary algorithms. PSBML is validated on real-world “big data” classification problems for various properties of meta-learning, scalability and noise resilience using well known benchmark datasets. The PSBML algorithm is also proven theoretically to be a large margin classifier with linear scalability in training time and space, giving it a unique distinction among the existing large scale learning algorithms. Finally, the EML framework is validated on a large genome-wide bioinformatics classification problem and a large time series problem, showing that the combined algorithms achieve higher predictive performance, training time speed up, and the ability to produce human-understandable discriminating signals as features.
Chapter 1: Introduction

Many real-world machine learning problems involve extracting knowledge from training data that are represented as a sequence. Some of the applications and sequence data forms are in 1) communication, such as speech, handwriting, music, language, text; 2) time series, such as stock prices, weather readings, and web-events; and 3) biological sequences, such as DNA, mRNA and proteins. In this work, “sequence” means an ordered combination of symbols drawn from a finite alphabet set. Examples of alphabet sets are: the vocabulary of English words in the case of text classification, the four-letter alphabet of nucleotides in the case of DNA-sequence classification, or music notes in the music learning task. Given a sequence training data, the learning task is to assign a class label for unseen sequences which are similar to the trained sequence data. Inferring that a sequence exhibits a certain property is difficult when no a priori information is available on what gives rise to the sought property. In sequence-based classification, the immediate goal becomes the discovery of signals or features in the sequence data that correlate with the desired property and discrimination between sequences that contain the property and those that do not.

Sequence-derived features can be global or local with respect to the data. For instance, native sub-cellular localization of a protein molecule correlates strongly with a global feature: the amino-acid composition in the protein sequence [1, 2]. The discovery of amino-acid composition as a meaningful feature for detecting localization is a direct result of biological insight that proteins need to have certain biophysical properties in order to operate in certain cellular environments. Domain-specific insight can also be employed to obtain local features often referred to as sequence motifs. For instance, biological experiments have elucidated motifs that correlate well with protein domains, families, folds, and functional sites. Motifs such as those documented in the PROSITE database [3] are often employed to characterize new sequences [4].
Insight from domain experts in a particular problem area is difficult to translate into meaningful features when a combination of local and global features are needed. Many problems involving prediction in sequence data, e.g., of protein enzymatic activity, DNA hypersensitive sites, or RNA/DNA splice sites, call for complex features [5–12]. For instance, work in [7] shows that different types of features are needed to obtain high accuracy and precision in DNA splice site prediction.

In the absence of domain-specific insight to guide feature generation, exhaustive enumeration techniques are often employed. The enumeration technique known as “spectrum” features, which are finite length short sub-sequences from the alphabet combinations, is the most well known technique used in many bioinformatics problem [13]. A $k$-spectrum is the set of $d = |\Sigma|^k$ features that correspond to all strings of length $k$ ($k$-mers) generated from an alphabet $\Sigma$. A $d$-dimensional feature vector is then associated with an input sequence by recording the frequency of occurrence of each of the $d$ $k$-mers in the sequence. Some more complex iterative methods which use these short sub-sequences but look for complex positional or correlational interactions have also been used in some sequence classification applications [8]. Because the number of features can easily become intractable for enumeration, the number of feature types considered, as well as the complexity of designed features are limited [6]. Some algorithms which perform well in certain sequence prediction are the kernel methods or Markov models, which effectively are “black-box” algorithms, i.e., the discriminating features generated by these methods are not easily identifiable [14]. Many complex bioinformatics problems such as predicting regulatory regions in DNA, identifying promoter regions in transcription areas and annotating splicing regions in genomes can be mapped to sequence classification problems. The challenge is that domain experts like biologists would want to have the prediction performance of these non-transparent algorithms while retaining the advantages of explicit feature based methods. Biologists would like to have transparent features elucidating important characteristics such as branching site signals in splicing, silencer/enhancers in regulatory regions and promoter regions signals. More importantly, it will save the biologists from the huge wet lab costs, if these features
can be generated in an automated manner, without domain experts and can capture the complex inter-relationships between sequence elements from the dataset.

Another area of sequence classification is in the time series classification domain. Many real world applications such as motion detection in robotics, climate change detection based on anthropogenic measurements, financial market predictions based on stock price variations, etc., fit into the broad definition of time series classification. In time series classification, individual instances are sequences of numeric values associated with labels. Using traditional machine learning approaches such as treating the time series sequences as high dimensional vectors and classic classification algorithms have faced with the well known “curse of dimensionality” problem [15]. Recently, the field of time series classification has seen success by using preprocessing steps that discretize the time series and find the over-represented patterns as features [15]. The discretization of numeric high dimensional data to low level words based on fixed alphabets and the use of spectrum features has shown results comparable or even better than some complex state-of-the-art algorithms. Are there more complex global and local patterns in the discretized time series classification similar to bioinformatics applications mentioned above?

There is a need of a generic feature generation algorithm for sequence-based datasets in the classification domain in machine learning that is automated, transparent, and assists the underlying classifier to have good prediction power. Also, it is important to have an algorithm that is not limited by domain expert insight, types of features considered, or the ability to enumerate features. The dilemma, of course, is that by expanding the scope and complexity of the feature generation process, one is invariably confronted with an NP-hard problem [16]. A variety of general purpose search techniques have been shown to be effective for NP-hard problems.

Evolutionary Algorithms (EA) have been a popular search methodology, especially when the search space is large and finding interesting elements on the complex landscapes is challenging [17]. The ability of EA to explore exponentially large feature spaces makes them
appealing methods for feature generation in addition to the classic enumeration and branch-and-bound techniques. Intuitively, a question arises, “Can EA be used to search a larger and more complex feature space and generate transparent features from the training sequence data that can significantly improve the classification accuracy of a learning algorithm?” The success of an EA in the search space depends on having well defined “building blocks” as an effective representation, breeding operators that balance exploration and exploitation of these representations, and most importantly the fitness mapping of the representation. Can a generic EA-based automated feature generation technique be designed considering these issues for sequence classification domain? Will the EA based feature generation technique be effective and generic to solve wide class of bioinformatics and time series problems? This thesis will try to address these issues by designing an evolutionary algorithm for feature generation and validating it on many sequence classification applications.

Another important issue is that the size of the learning data presents a practical challenge to most supervised learning algorithms. Most sequence-based datasets are in the millions or billions in size, like the genome-wide sequences of organisms in bioinformatics. Many current machine learning techniques have scalability issues, either because they require the entire training set to be in memory for learning the model, or because the learning time during model induction grows exponentially with training set size, or both [18]. Basic solutions like reducing the size of the training datasets via sampling can be used, but they can introduce sampling errors [19]. Another approach is to employ complex changes specific to the desired machine learning algorithm for running on parallel and distributed architectures [20], but the flexibility and range of generic algorithms is lost. Generic scalable algorithms are more preferred in these sequence classification problem frameworks because of the different biases and constraints posed by the different sequence-based datasets. Intuitively, this problem can be posed as a search problem, i.e., to search for few but significant instances from the large training dataset. Boosting techniques in machine learning are designed to deal with hard-to-classify examples, but do so by making multiple passes over the training data [21]. Can EA assist a pluggable machine learner to reduce the large training dataset to a significant few
hard-to-classify examples? Can the evolutionary algorithm be executed in embarrassingly parallel manner to give the required speed up for the training process? To address these issues in a reasonably general way, a parallel boosting algorithm has been developed in this work that combines concepts from spatially structured evolutionary algorithms (SSEAs) and boosting techniques.

In this work, an evolutionary machine learner (EML) framework using concepts of EA and machine learning to generate features in a discriminating way as well as to scale and perform well on large sequence-based datasets is proposed. The EML framework comprises two main algorithms, Evolutionary Feature Generator (EFG) and Parallel Spatial Based Machine Learner (PSBML) as shown in Figure 1.1. The EFG algorithm will be responsible for generating complex discriminating and understandable features that help in the learning process. The PSBML algorithm will be responsible for using these features and generically using pluggable classifiers to scale and learn on the large datasets.

![Figure 1.1: EML Framework with two sub-components EFG and PSBML](image)

The best validation of a generic framework comprising many new proposed algorithms is to first validate each algorithm independently of the other for its claims. Then it needs to be validated that when these algorithms are combined in the proposed framework, it
solves the overall general problem. The structure of the thesis is laid out exactly in that manner. I present the first algorithm, EFG for feature generation and validate it on four complex sequence problems from the bioinformatics domain. Next, the generic nature of EFG is also validated on complex time series classification applications. I then present the parallel learning algorithm PSBML and derive some important theoretical properties of this parallel algorithm. PSBML is next validated on large datasets for its meta-learning, scalability in time, scalability in space, and robustness to noise characteristics. Finally, the two algorithms are combined and validated on large sequence based datasets involving genome-wide sequences in organisms and time series applications.

The contribution of the thesis will be many, some of which are outlined below:

1. Generic feature generation algorithm for sequence based datasets to generate explicit discriminating features that help both experts and predictive models [22].

2. Generic machine learning algorithm that can scale and perform well for large dataset supervised learning tasks [23].

3. Explicit features and models for complex bioinformatics classification problems like splicing, promoter identification, regulatory region identification and Alu sites recognition that can help biologists [22,24,25].

4. Feature generation techniques in collaboration with Symbolic Aggregation Approximation (SAX) for time series classification.

5. A scalable large scale learning framework for genome-wide sequencing and time series datasets for high throughput and accurate predictions.

6. Theoretical insights and model for a generic parallel machine learning that can help researchers get more insights and understand constraints better [26,27].

7. A theoretical model that can be used for spatial evolutionary learning used in search or optimization in the metric spaces where geometric breeding operators are used.
The thesis gives a complete background on technology and related work first. Next, it
describes the EFG algorithm and its components in detail. Application of EFG and compar-
ison with the state-of-the-art algorithms on various bioinformatics classification problems
such as identifying hypersensitive sites (HSS), Alu elements recognition, promoter region,
and splice sites is presented in detail in the next chapters. EFG along with SAX is applied to
various time series domain problems to validate its general nature in the following chapter.
The parallel learning PSBML algorithm with theoretical analysis and empirical validation
is unveiled in the next chapters. Finally, EFG and PSBML are combined to solve some
complex bioinformatics and time series classification problems. Conclusion and future work
are detailed at the end.
Chapter 2: Background and Related Work

This chapter offers a brief review of sequence classification and large scale machine learning to summarize the current state of the art research in the respective fields. I then introduce and give a brief overview on evolutionary algorithms, genetic programming, and spatial EAs, which are employed and adapted in the proposed algorithms.

2.1 Sequence Classification

The sequence classification methodologies can be sub-categorized into the following different categories based on past surveys [7,28].

- Feature-based: These methods convert the sequence data into well known feature vectors that can be used with any underlying classifier.

- Similarity-based: These methods use distance or similarity measures between the sequences and exploit classification techniques that work well with these measures.

- Statistical Model-based: These methods use statistical models like hidden Markov models or weighted matrices to assume and learn a distribution model for the given sequences.

2.1.1 Feature-Based Classification

Having explicit features known to the domain experts and mapping them into real vectors gives these methods a distinct advantage. One common methodology of performing explicit sequence to feature vector space transformation is to use “spectrum features” [13]. The methodology enumerates all finite subsequences of length $k$, also referred to as $k$-mer motifs, and employs these motifs as spectrum features to transform input sequences into feature
vectors. Either the frequency of the short subsequences or the presence of the subsequences as boolean are used in the feature vectors. This simple methodology has been successful in a number of sequence classification problems, such as promoter region identification, hypersensitive site location, and language recognition [13, 29, 30]. Another explicit feature based method, is to find the motifs in the training data using iterative sampling methods like Gibbs sampling. These overrepresented motifs that capture the compositional patterns in the training data have also been successfully applied to many sequence classification problems [31–33]. Again, the major disadvantage of these techniques is the inability to capture higher order signals like presence of correlated motifs, positions, and region specific sub-sequences in the training data.

Many of these sequence motifs obtained from enumeration are seldom useful and can be removed by using effective feature selection algorithms [34]. There have been extensions to the feature-based classification to enumerate more complex features that are position-based, region-based, and composition-based in an iterative way [8, 35]. The issue with these methods is that as features become complex, the feature space becomes enormous and puts constraints on the iterative process that enumerates them. These motifs, when used either as simple compositional or even positional overlook an important matching characteristics like gap constraints and internal disjunction of alphabets which have been very successful in string pattern matching.

2.1.2 Similarity-Based Classification

In similarity-based classification, a distance function or a similarity function is defined to measure the distance/similarity between a pair of sequences. Given such a distance function, one can use some existing classification methods, such as k-nearest neighbor classifier (KNN) and SVM with local alignment kernel [36, 37]. For KNN, the choice of k, the distance function used, and the size of training data make it extremely sensitive to sequence classification problems.

In contrast, SVM has been proved to be a widely used and effective method for sequence
classification [14,38]. The basic idea of applying SVM on sequence data is to transform the input sequences into a feature space and then find the maximum-margin hyperplane to separate the classes. A kernel function is used to perform the transformation of the data to the high dimensional feature space. The kernel function can be viewed as the similarity measure between any two sequences [39]. There have been various kernel functions proposed for sequence classification which have one to one correspondence with the features based methodology. Examples include the weighted position kernel, which is similar to positional features, and the spectrum kernel, which is similar to the spectrum features [14,40]. Dynamic programming using local alignment as the similarity function between two sequences has also been successfully used in the protein sequence classification [41]. Concepts from evolutionary computation have been proposed to learn effective, possibly more complex kernels for a particular sequence classification problem at hand [24].

### 2.1.3 Statistical Model-Based Classification

Statistical learning methods can be broadly classified into discriminative or generative models. Discriminative models are limited to a supervised learning setting where the joint distribution \( P(C,X) \) is learned from labeled vector data \( X \) with class labels in \( C \). Class label predictions for an unlabeled sequence \( x \in X \) are then made by evaluating the conditional probability \( P(c|x) \) from the learned joint distribution. Generative methods can additionally exploit unlabeled data, but lose this ability when trained discriminatively [42]. Nonetheless, discriminative models are preferred in many classification settings, as they provide a more direct way at modeling the posterior without first addressing a more general setting (as demanded by modeling the joint probability) [43].

Heuristic procedures have been proposed to combine discriminative and generative models [44] as a way to address the issue that generative methods lose their ability to exploit unlabeled data when trained discriminatively [42]. The resulting hybrid methods have been shown to result in superior performance on recognition of transcription factor-binding sites on DNA [45]. Representative methods include the position-specific scoring matrix
(PSSM)—also known as the position-weight matrix (PWM)—method, which assumes nucleotides at all positions are drawn independently [46, 47], the weight array model (WAM) which relaxes assumptions of independence by additionally modeling dependencies on a previous position [48], higher-order Markov models, which model more dependencies and outperform PSSMs [49, 50], and even more complex models like Bayesian networks [51, 52] and Markov Random Fields (MRFs) [53, 54]. A mixture of Bayesian trees and PSSMs in [55], smooth interpolations of PSSMs, and empirical distributions [56] have also been proposed to model arbitrary dependencies. Maximum Supervised Posterior (MSP) is a Bayesian discriminative algorithm that has performed superior to many other generative and discriminative methods including kernel techniques on biological sequence data [57].

2.2 Big Data Machine Learning

As stated in the introduction, most sequence-based datasets are large in volume, and learning from these large training datasets is normally referred to as Big Data or large scale machine learning. The Big Data learning is a large field by itself, and there have been many advances and research in this area. In this section, I focus on advances in the area of large scale learning to mitigate the pressing issue of scalability.

In statistical learning theory, a formal relationship between the notion of margin and the generalization classification error has been established [36]. As a result, classifiers that converge to a large margin have better performance in terms of generalization errors. One of the most popular examples of such classifiers is support vector machines (SVMs). The classification boundary provided by an SVM has the maximum-minimum distance from the closest training point. SVMs have training times of $O(n^3)$ and space complexity of $O(n^2)$, where $n$ is the size of the training set [58]. SVMs have been modified to scale to large data sets [59–63]. Many of these modifications introduce a bias caused by the approximations like sampling the large data or assuming a linear model, that can lead to a loss in generalization while trying to achieve speed.
Another approach in this area has been to use distributed architectures and network computing by performing algorithm-specific parallelizations [64–67]. These modifications are also conducted on other algorithms like decision trees, rule inductions, and boosting algorithms are optimized [68,69]. Most of these algorithms are changed to parallelize computations like matrix transforms in SVM or tree node learning, and many of them use communication infrastructure like message passing interfaces (MPI) for exchanges. MapReduce gives a generic framework for a divide-and-conquer-based approach and has been used in conjunction with learner algorithms to scale for large datasets [66]. Ensemble based learning on parallel networks have also been employed on various tree based algorithms for learning on enormous datasets [69].

2.3 Boosting and Large Margin Classifiers

Since the large scale learning work presented in this thesis is based on the idea of boosting and ensemble learning, background about this area is presented. The AdaBoost technique, and boosting in general, is an example of a learning methodology known as ensemble learning, in which multiple classifiers are generated and combined to make the final prediction. Ensemble learning has been shown to be effective with unstable classifiers, by improving upon the performance of the base learners [70]. AdaBoost induces a classification model by estimating the hard-to-learn instances in the training data [71]. A formal analysis of the AdaBoost technique has derived theoretical bounds on the margin distribution to which the approach converges [21]. In addition to AdaBoost, support vector machines (SVMs) are another popular family of large margin classifiers [72].

2.4 Evolutionary Computation

Evolutionary Algorithms mimic biological evolution to evolve a population of candidate solutions towards the true solutions of a difficult optimization or search problem [17]. EAs have been used in wide variety of NP problems, to perform global search when basic search
heuristics are rendered ineffective [17]. The superiority of EAs was recognized early [73] in
the field of feature generation. Since then, many studies have demonstrated the advantages
of EAs for feature generation and selection in different domains [9,10,12,74–79]. Work on
predicting enzymatic activity in proteins additionally shows the power of EAs in feature
generation in the bioinformatics related field [9].

2.4.1 Genetic Programming

Unlike standard GAs in which individual are fixed-length strings of symbols, an individual
in Genetic Programming (GP) is a variable-length tree composed of functions and variables.
The functions are internal nodes also referred to as non-terminals, whereas the variables
are the leaves also known as terminals. Originally introduced to evolve computer programs
and complex functions [80–83], GP algorithms allow evolving S-expressions that can be
represented as parse trees [17].

Since their introduction, GP algorithms have seen an increase in their usage in diverse
problems in bioinformatics [9,84–88]. Abundant applications can be found in bioinformatics
on quantitative structure-activity analysis in drug design, cancer classification from gene
expression data, classification of genetically-modified organisms, and classification of cog-
nitive states from fMRI data [84–92]. GP with some modification is used as a primary EA
for generating discriminating features in this work.

2.4.2 Spatially-structured evolutionary algorithms (SSEAs)

Spatially-structured evolutionary algorithms (SSEAs), which use topologically distributed
populations and local neighborhood selection have been well analyzed in the EC litera-
ture [93]. Selection pressure in an SSEA is determined by two design choices: the selection
method used by the local EAs running on each grid point, and the size and shape of the
neighborhood structure. SSEAs have been shown to maintain a diverse set of better indi-
viduals longer, resulting in improved performance in many applications [94]. However, the
key feature that the framework takes advantage of is its “embarrassingly parallel” architecture in that at each topological grid point a local algorithm is running that has only local interactions with its immediate neighbors.
Chapter 3: Evolutionary Feature Generator (EFG)

In this chapter, I present a detailed account of the EFG algorithm. First, I outline the input to feature-space transformations as a basis to motivate the discussion of the purpose of EFG in sequence mining; I then lay out the methodology and working of the evolutionary feature generation algorithm in assisting classification.

3.1 Input-Feature Space Transformation

Finding a decision boundary or class separation becomes very difficult for a complex non-linear training dataset in the input space for most machine learning algorithms. Kernel methods [63] are based on transforming the inputs to a high-dimensional feature space and finding a simpler linear class separator in the feature space. This is illustrated in Figure 3.1, where two dimensional non-linear circle fitting data is easily converted to three dimensional by using a known transformation, $\Phi$, which converts the points $(x, y)$ to $(x, y, x^2 + y^2)$. The data gets easily separated using a simple linear classifier in the three dimensional feature space.

Generally, effective transformation for classifying, as in the above example, depends on knowledge of the input space. Well-known transformations like polynomial, Gaussian, Laplace, and spline functions are used as transformations that convert the input space to high-dimensional feature space without using any knowledge from the domain or input data. The advantage of this technique is that it makes the problem fairly independent of datasets and independent of the input space dimension, as only the dot product is required in the feature space. The disadvantage is that the knowledge of discrimination is embedded in the transformation function and is limited to only kernel-based classifiers. Also many of these transformations do not work for complex non-numeric datasets like sequence
datasets. Transformations specific to input sequences are needed or naïve enumerations of a combination of sequence elements are used as features.

Another way to approach the classification problems in sequences is to embed the knowledge of the generic sequence domain in the transformations. The knowledge of the sequence domain can be meta-level building blocks that find patterns in the sequences, and some algorithm creates concrete features or learning elements from these meta-level building blocks. These meta-level building blocks are abstract concepts that are independent of datasets but are generally true in sequence mining. An example of such a meta-level construct is the presence of a “short subsequence” or a “motif” common in all the sequences, but what those “motifs” are specifically depends on the dataset and have to be found.

If such features are known, a simple classifier like the linear classifier as in the circle dataset example in the Figure 3.1 can be used to learn a complex model. If this can be done in a generic and automatic way, then we have a customized transformation from input to feature space for the sequence domain. The transformation using the meta-level concepts
creates data-specific features that assist in classification. This is precisely what the Evolutionary Feature Generation algorithm does. It uses basic domain knowledge of sequences and lets an evolutionary algorithm generate features that have good discriminatory power. These complex features evolved from some basic building blocks that act as explicit features in a new high dimensional space. Thus, EFG can be seen as an explicit sequence-domain transformation that converts the input sequences to high-dimensional explicit features.

3.2 EFG as an EA

EFG uses a Genetic Programming (GP) algorithm to explore a large, complex space of potentially useful features from the given training dataset. Features are represented as standard GP trees, and a population of features is evolved over time using standard GP mechanisms of mutation and crossover. EFG uses a surrogate filter-based fitness function to estimate the usefulness of the GP-generated features, since the wrapper methodology to find effectiveness of the features is costly. A hall of fame mechanism incrementally collects the best estimated features for subsequent use with a classifier. Next, we present details of all the evolutionary elements and constructs that are used in the EFG algorithm.

3.2.1 Feature Representation

Various researchers, as highlighted in the related work section, have individually discovered many building blocks that can be very effective in finding the patterns in sequence classification. The novelty of the EFG algorithm is that it not only defines many new building blocks, but it also gives a structure through strongly-typed GP evolution, combining various building blocks in an effective and human-understandable manner. This structured way of searching a vast feature space involves building a complex structure given the constraints defined from simpler ones. The strongly-typed GP plays the role of giving structure and guidance to the vast search space of features. Next, we highlight the building blocks from the simplest short subsequence known as a motif, which becomes the common building block to the complex higher-order signals that can be constructed through the algorithm.
We have arranged the explanation at various levels of complexity starting from level 1 (the simplest) to level 3 (the most complex).

<table>
<thead>
<tr>
<th>Name</th>
<th>Args</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>matches</td>
<td>motif</td>
<td>Boolean</td>
</tr>
<tr>
<td>matchesAtPosition</td>
<td>motif, position</td>
<td>Boolean</td>
</tr>
<tr>
<td>positionalShift</td>
<td>motif, position,Shift</td>
<td>Boolean</td>
</tr>
<tr>
<td>correlational</td>
<td>motif, motif, position, close</td>
<td>Boolean</td>
</tr>
<tr>
<td>regional</td>
<td>motif, region</td>
<td>Boolean</td>
</tr>
<tr>
<td>and</td>
<td>2 non-terminal boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>or</td>
<td>2 non-terminal boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>not</td>
<td>2 non-terminal boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>motif-*</td>
<td>ERC-chars</td>
<td>Motif</td>
</tr>
<tr>
<td>position</td>
<td>ERC-int</td>
<td>Integer</td>
</tr>
<tr>
<td>shift</td>
<td>ERC-int</td>
<td>Integer</td>
</tr>
<tr>
<td>close</td>
<td>ERC-int</td>
<td>Integer</td>
</tr>
<tr>
<td>region</td>
<td>ERC-int, ERC-int</td>
<td>Integer</td>
</tr>
<tr>
<td>ERC-char</td>
<td>{Symbols}</td>
<td>Character</td>
</tr>
<tr>
<td>ERC-int</td>
<td>{1, \ldots, length}</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**Level 1: Motif**

The most common building block is the presence of a short subsequence of strings of a given length, which are constructed as parse trees from the given alphabets. These motifs are used as a building block in all the second level constructs.

**Level 2: Compositional Features**

The `matches` operator allows constructing simple compositional features. An example is provided in Fig. 3.2, using alphabets from DNA sequences. The nucleotides that make up the motif serve as leaves. The evaluation involves obtaining the occurrence of the motif in a
given sequence. The motif function length depends on the application; work on splice sites in [7,8] shows that no longer than 8-mers are useful for short-subsequence based matching, so we can limit motif length to between 1 and 8. The domain expert can give the bounds based on prior knowledge or they can be tuned in an iterative way by looking at the quality of the motifs and features they generate.

Level 2: Regional Features

The regional operator can be considered an extension of compositional features, restricted to certain bounds given by the domain experts. For example, in the splice site sequence identification problem, it would be important to find compositional patterns in the coding and non-coding regions in the sequence. The coding region and non-coding region changes based on whether it is a donor or acceptor site, as depicted in Fig. 3.3. Region-specific
features were found to be important functional signals in many sequence classification problems [8,95].

![Diagram of regional domain information to capture regional features.](image)

**Figure 3.3:** Region-specific domain information to capture Regional Features.

### Level 2: Positional Features

The `matchesAtPosition` operator allows constructing simple positional features from the motifs at a given position. An example is provided in Fig. 3.4.
The positional features correspond to local features often employed in classification of sequences. In these features, the goal is to find a specific motif at a specific position in the sequence.

![Diagram of positional features](image)

Figure 3.4: Position (left) and sequences with those motifs (right) features graphically represented.

**Level 2: Positional Shift Features**

The `positionalShift` operator allows constructing positional features that may be displaced in either direction by a small shift given as a parameter. An example is provided in Fig. 3.5. The positional shift features were discovered to be very effective in complex sequence/series classification problems [14].

**Level 2: Correlational Features**

The `correlational` operator captures the presence of positional features adjacent to each other, within a distance. Correlational features are generated from two simpler motifs:
position in the sequence and closeness capturing the adjacency, as shown in Fig. 3.6. Correlational features could be evolved from two positional features using a logical operator like \textit{and}. However explicit guiding structures, like the presence of this construct, helps the search to be more effective.

\textbf{Level 3: Complex Higher Order Signals}

Many statistical learning approaches, such as Bayesian networks and Markov-chain models, rely on higher order elements formed from lower order signals as the discriminative features [96]. The approach of having logical combinatorial operators like \textit{and}, \textit{or}, and \textit{not} acts in a similar way to construct more complex features combining the simple Level 1 motifs or Level 2 elements, such as positional, compositional, correlational, shift-positional, and regional features, or even the Level 3 features to form any level of complex chains,
from simpler conjuncts or disjuncts. Figure 3.7 illustrates a conjunction feature combining one positional and one compositional feature. Figure 3.8 illustrates a complex disjunction feature combining one positional and one negated compositional feature.

### 3.2.2 Genetic Operators

As in most evolutionary algorithms, individuals have to undergo some modifications through genetic breeding operators to generate a new representation from the existing population individual(s). Studies have shown robust evolutionary algorithms incorporate both mutation and crossover as the breeding operators [97].
Mutation

The role of mutation in evolutionary algorithms is to make small, incremental change to an individual to form a new individual with a small change. GP-styled evolution normally has a more disruptive mutation operator that randomly generates a subtree and replaces a node in the given individual with that subtree [83]. In this research, we explored forming problem-specific mutations as small, incremental operators. These mutation operators are motif mutation, positional mutation, shift mutation and adjacency mutation. Figure 3.10 illustrates how a parent individual subjected to some of these mutations creates a new individual as a child. Motif mutation has its affects on the compositional and regional...
Figure 3.8: Disjunction features combining two positional and one negated positional feature.

features. Any one node with the **motif** return type in the tree is chosen with probability $P_n = p$, typically in the smaller range like 0.1. A character in the motif is then selected at random and replaced with any letter of the alphabet. This process is similar to the standard bit-flip operation in EAs. If the alphabet character set has any mismatch characters, this form of change helps increase the range of string matching. Positional mutation affects the **positional**, **positionalShift** and **correlational** features. One of the nodes has its positional value changed to a random integer within the bounds of the sequence length to explore the presence of the motifs in some other positions in the sequence. Shift mutation affects the **positionalWithShift** feature. One of the nodes has its shift value changed to a random integer within the bounds of the shifting range specified by the designer. Finally, adjacency mutation affects the correlational node in the tree structure to find alternative adjacency or closeness as determined by the designer.
Crossover

In this work, the standard subtree crossover, one of the most common genetic recombination operators used in GP [83] is employed. Subtree crossovers have been very effective in GP, as
they form complex trees and explore vast search spaces more effectively. The tournament selection scheme is carried out twice to obtain two individuals from a population. Subtree crossover, as shown in Fig. 3.9, finds a random node in both the tree structures and swaps the subtrees forming a new individual, given two selected individuals as parents. Once two parents are selected, their crossover proceeds as in the standard Koza-style subtree crossover [83]. If the selected nodes don’t match in return type, and swapping the subtrees rooted at these nodes does not violate the maximum depth constraint, then the swap is performed. Otherwise, the process is repeated for a maximum of 10 times to find another random position in the tree to swap.

3.2.3 Bloat Control

One common problem with evolving variable-length or tree-structured individuals in EAs is that as the generation progresses, the individuals become complex in structure or length without any changes to fitness, commonly referred to as “bloat”. Bloat also affects the understandability of the features because of inconsequential nodes added into the trees. One of the ways to control bloat is to have structural elements that reduce the chance of forming larger trees without much improvement to the fitness. By making specific building blocks, such as the correlational feature, rather than leaving it for evolution to form complex trees with two positional features capturing adjacent positional information is one such example. Another method that is used in EFG to combat increase in length and complexity is to employ a lexicographic tournament selection: if multiple individuals have the same fitness, the selection chooses the individual with the smaller tree depth [83].

3.2.4 Population and Generation Mechanism

The EFG algorithm creates individuals in generation 0 consisting of $N$ randomly generated features using the well-known ramped half-and-half generative method [83]. The population size for GP is generally large, and we have employed a size of 5000. Half of the individuals of the population are created using the full method and the other half are created using the
grow method. The maximum depth of the trees generated is ramped, such that individuals are created in a range of sizes. The full method recursively adds non-terminal nodes to the tree until the maximum depth is reached and then puts the terminal nodes as the leaves. The grow method is similar to the full method in its construction, but it does not restrict the choice of nodes being just the non-terminal nodes until the maximum depth. The full method results in fixed-shape trees while the grow methods results in arbitrary-shaped structured trees. Employing the ramped half-and-half method, a diverse initial population, both in structure and computational complexity, is achieved \[83\]. Instead of keeping the population size fixed for every generation, we employed the well known strategy of population implosion to reduce the size of population by 10% in every generation \[98\].

3.2.5 Fitness Function

A surrogate fitness function, or a “filter” approach, which is considered to be fast and effective way for feature evaluation \[99\] is employed in EFG. Since most sequence classification data are imbalanced and have very few positives and a large number of negatives, the goal is to improve precision while managing the discriminating power of features. We formulate the fitness function: 

\[
\text{Fitness}(f) = \frac{C_{+,f}}{C_+} \times \left| \frac{C_{+,f} - C_{-,f}}{C_+ + C_-} \right|
\]

In this equation, \( f \) refers to a feature, \( C_{+,f} \) and \( C_{+,f} \) are the number of positive and negative training sequences that contain the feature \( f \), respectively. \( C_+ \) and \( C_- \) are the total number of positive training sequences. This fitness function tracks the occurrence of a feature in positive sequences, as negative sequences may not have any common features or signals, while penalizing non-discriminating features equally found in positive and negative sequences.

The goal is to maximize the fitness function, but the Koza fitness formulated for GP aims for minimization \[83\]. So, the Koza fitness of a feature is defined by \( f \) as Koza(\( f \)) = \( 1/(\text{Fitness}(f)) \). EFG then converts the Koza fitness back into the GP-adjusted fitness \( 1/(1 + \text{Koza}(f)) \) to select fit individuals. Note that the GP-adjusted fitness takes values in \([0, 1]\).
3.2.6 Hall of Fame

Since GP is a generational EA, i.e., the parents die after producing the offspring, there can be a “genetic drift” and convergence to a local optimum [17]. This can result in the loss of some of the best individuals, which can be useful discriminating features for classification. Introducing elitism, i.e., the ability to keep some of the best individuals in the population helps to overcome this at the expense of introducing strong selection pressure. To maintain this balance of not losing the best individuals in every generation and not introducing elitism for strong selection pressure, external storage has been found to be the ideal design decision [17]. The EFG algorithm will use this external storage of features known as hall of fame, and at the end of the EA run, these highly fit feature sets chosen from each generation become the feature set that constitutes the solution. The choice of the number and size of individuals will be empirically determined and explored for the sequence-based problems. Keeping the fittest individuals in a hall of fame guarantees optimal performance [100].

3.3 Discussion

One of the issues with GP-generated features is poor readability and high complexity introduced because of the bloat. Even though the bloat is controlled by the mechanism discussed above, the feature expression still needs to be simplified for readability. EFG employs simple Boolean simplification rules as illustrated in Table 3.2 as a post-processing technique to further simplify complex expressions. This can be further improved by various enhancements, such as using the simplifications during the evolutionary cycle or using more sophisticated boolean simplifier packages in future.

Also, the method of constructing complex features from simpler building blocks is very similar to the concept of “constructive induction”, which has been successful in machine learning [101]. Combining simpler features in complex combinations of logical Boolean operators using the search process has similarity to Bayesian networks used effectively in many machine learning techniques [102]. The relationship to these well known learning
Table 3.2: Boolean simplification rules for EFG post-processing and improving the readability of the features.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>not not</code> booleanExpr</td>
<td>booleanExpr</td>
</tr>
<tr>
<td>true and booleanExpr</td>
<td>booleanExpr</td>
</tr>
<tr>
<td>false and booleanExpr</td>
<td>false</td>
</tr>
<tr>
<td>true or booleanExpr</td>
<td>booleanExpr</td>
</tr>
<tr>
<td>false or booleanExpr</td>
<td>booleanExpr</td>
</tr>
<tr>
<td><code>(not booleanExpr1) or (not booleanExpr2)</code></td>
<td>not (booleanExpr1 and booleanExpr2)</td>
</tr>
<tr>
<td><code>(not booleanExpr1) and (not booleanExpr2)</code></td>
<td>not (booleanExpr1 or booleanExpr2)</td>
</tr>
</tbody>
</table>

The aim of this chapter was to give details of the EFG algorithm and various EA constructs that are needed. Many elements, like the choice of alphabets, range of position, and shift in representation, are totally dependent on the applications. Similarly, the choice of mutations and crossover probability balance is also problem dependent. In the chapters where we implement EFG for specific problems, these choices are made explicit and reasons are provided.
Chapter 4: EFG for Hypersensitive site and Alu sequence classification

Identifying regulatory sequences promises to reveal underlying reasons for phenotypical differences among cells and for diseases associated with pathologies in protein expression. Hypersensitive (HS) sequences are considered reliable markers of regulatory sequences and are currently the focus of classification methods. Another interesting problem for DNA sequence analysis is the mapping of Alu elements. Such sites belong to a family of primate-specific repetitive DNA elements estimated to make up more than 10% of the human genome. The main question considered in this chapter is, “Can the EFG algorithm be used in the context of these two sequence classification problems?” I will give background information on both problems, explain adaptation of EFG for these applications, explore experimental validation, and offer some analysis of the features.

4.1 Background and Introduction

4.1.1 Hypersensitive Sites Recognition

Cells of different tissue have been found to have the same set of genes, yet display significant phenotypical differences. Experimental evidence now demonstrates that cell development, proliferation, apoptosis, aging, differentiation, and even certain pathologies depend on the spatial and temporal expression of genes [103,104]. Protein expression in cells is controlled by regulating which genes are transcribed into mRNA. DNA-binding proteins known as transcription factors bind specific DNA sequences known as regulatory elements to activate or repress transcription of genes.

Understanding what governs expression patterns and their alterations in eukaryotic cells relies on identifying regulatory elements associated with an annotated gene. Full
annotation of the human genome with regulatory elements is an emerging challenge in genomics research [103, 104]. Evidence has emerged over the past twenty years that the eukaryotic cell hides regulatory elements by wrapping and packaging DNA into chromatin fibers [105–108].

DNA-modifying enzymes like non-specific endonuclease DNase I disrupt chromatin architecture to expose regulatory sequences. Experimental research shows that DNA sites that are hypersensitive to DNase I, referred to as HS sites, are reliable markers of regulatory elements. As shown in Figure 4.1, HS sites are found to be co-localized with regulatory elements in all known regulatory classes (transcriptional enhancers, promoters, silencers, insulators, splicing regulators, and chromatin boundary elements) [103–106,109–111].

Figure 4.1: DNase Hypersensitive sites in a genome [112]. Reproduced by the permission of copyright owner.

Until the discovery of HS sites, wet-lab identification of regulatory elements relied on
detecting transcription factor-binding sites [113]. Mapping HS sites now underpins the discovery of most regulatory elements in the human genome [111]. Bolstered by experimental success, mapping HS sites is considered the gold standard for discovering regulatory elements. Recent years have seen a burst of high-throughput experimental methods that are recovering a large number of HS sequences [111, 114].

The large number of discovered HS sequences have created the opportunity to develop computational methods that can learn to recognize these sequences and assist in genome-wide annotation of regulatory elements. Early approaches, which suffered from the scarcity of available HS sequences, focused on identification and combinatorial analysis of short-sequence motifs presumed to represent transcription factor-binding sites in regulatory sequences. The motifs were often obtained from experiment, analysis of upstream regions of co-regulated genes [115, 116], or employment of phylogenetic data [117]. Performance was generally poor [118] and did not improve until a large repository of known HS sequences was made available to machine learning approaches. Recent computational approaches to recognition of HS sequences employ Support Vector Machines using the well known spectrum kernel (SVM) [6].

4.1.2 Alu Elements Predictions

Mobile elements, which make up 45% of human genome, play key role in shaping eukaryotic genomes. Mobile elements use extensive cellular resources for their replication, insertion and amplification. Because of their amplification, they are considered to play an important role in various cell-related diseases [119]. ‘Repetitive elements’ are types of mobile elements present in multiple copies in the human genome and are further classified as tandemly arrayed or interspersed. Interspersed elements are further sub-divided based on the size and the short interspersed elements (SINEs) correspond to the sequences less than 500bp in length [120]. Alu SINEs are one of the SINEs present in the genomes. Alu elements correspond to the members of this family of repeats that contain a recognition site for an enzyme known as “AluI” [120].
Recent human genome analysis has shown that the mobile elements that are present in abundance are the Alu elements, considered to be 10% of the mass of the human genome, which accumulate in gene-rich areas [121]. Detailed analysis has shown that Alu RNAs were ancesterally derived from the 7SL RNA gene, a important part of the ribosome complex [122]. Thus the 1.1 million Alu elements that are dispersed in the human genome can be traced back to the initial gene duplication in the primates. These types of gene duplication have been sporadic in the mamalian and non-mamalian genomes.

The amplification of Alu elements is hypothized to occur by the reverse transcription of an Alu-derived RNA polymerase III transcript in a process called retrotransposition [123]. Only a few human Alu elements, the source genes, seem to be retrotransposition competent [124]. Mutations that accumulate in the source genes also get inherited by their copies. Thus, the human Alu family comprises several distinct subfamilies of different genetic ages formed by a hierarchical series of mutations. Figure 4.2, top, shows the alignment of the Alu subfamily Alu-Sx consensus sequences with younger sequences progressively following [125]. The dots represent the same nucleotides, the dashes represent the deletion, and highlighted color boxes show mutations.

The rate of amplification of human Alu elements has been non-uniform. Figure 4.2, bottom, illustrates the pattern of expansion of the Alu family in primate genomes in relation to the subfamily size. The duplication of Alu repeats began more than 40 million years ago. Early in primate evolution, there was approximately one new Alu insertion in every primate birth and the current rate of Alu amplification is estimated to be on the order of one Alu insertion in every 200 births [126, 127]. Because of the wealth of knowledge Alu sequences bring in understanding the ancestral inheritance and its relation to important diseases, interest from the machine learning community for detecting Alu sites is rather recent [128].

### 4.2 EFG

Since the HSS and Alu sequence datasets are relatively small, features generated from the EFG algorithm needs to be reduced further to find effective and non-redundant features. In
this research, an Evolutionary Feature Selection (EFS) using a GA-based algorithm with a filter technique to further reduce the features to meaningful subset is employed. The overall framework is shown in the schematic Figure 4.3 on how the training/testing sequences get converted to features and further reduced for classification using a well-known classifier.

Following are the changes or the parameters required to run EFG for both hypersensitive sites and Alu sequences.

4.2.1 EFG algorithm

Feature Representation

The alphabets in the motif representation of EFG terminals correspond to characters from the IUPAC code for DNA sequences [129]. The alphabet in this case was the larger IUPAC code detailed in Table 4.1, which contains characters that represent more than one nucleotide. These symbols employ ambiguous pattern-like designations, which allow motifs in which specific positions are not constrained to specific nucleotides but rather to a group of nucleotides with shared properties.

Genetic Operators

EFG uses positional mutation, shift mutation and motif mutation each with 0.1 probability as asexual perturbation operators. EFG uses subtree crossover with max-depth of 50 as the recombination operator, with a probability of 0.7.

EFG parameters

EFG uses a population size of $N = 8,000$ and hall of fame as an external memory pad of $\ell = 300$ for every generation.

4.2.2 EFS Algorithm

The objective of the EFS algorithm is to find a subset with a high feature-class correlation to retain discriminating power but low feature-feature correlation to reduce redundancy
Table 4.1: IUPAC code, adapted from [129].

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Description</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>G</td>
<td>Guanine</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>Adenine</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>Thymine</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>Cytosine</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>G or A</td>
<td>puRine</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>T or C</td>
<td>pYrimidine</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>A or C</td>
<td>aMino</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>G or T</td>
<td>Ketone</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>G or C</td>
<td>Strong interaction</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>A or T</td>
<td>Weak interaction</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>A or C or T</td>
<td>H follows G in alphabet</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>G or T or C</td>
<td>B follows A in alphabet</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>G or C or A</td>
<td>V follows U in alphabet</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>G or A or T</td>
<td>D follows C in alphabet</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>G or A or T or C</td>
<td>aNy</td>
<td></td>
</tr>
</tbody>
</table>
Figure 4.2: Alu-subfamily Sx, showing younger sequences having more mutations progressively [119](top). The expansion of Alu subfamilies (Yc1, Ya5a2, Yb9, Yb8, Y, Sg1, Sx and J) is superimposed on a tree of primate evolution [119](bottom). Reproduced by the permission of copyright owner.
Figure 4.3: EFG algorithm with EFS algorithm
among features. EFS achieves this by employing a correlation-based fitness function.

**Feature Representation**

EFS evolves feature subsets. This is achieved by having an individual in the evolving population in EFS correspond to a feature subset represented as a binary string. The length of the string is equal to the hall of fame. A string of all ‘1’s would correspond to the maximum subset, the hall of fame itself, and a string of all ‘0’s would correspond to the empty subset.

**Genetic Operators**

EFS uses a standard bit-flip mutation operator with mutation rate of $1/l$, where $l$ is the length of the genome (number of individuals). Additionally, standard uniform crossover is used; each bit is considered a crossover point with a given probability of 0.5. It has been shown that employing uniform crossover along with bit-flip mutation is very effective at balancing exploration and exploitation of search landscapes [97]. Parent(s) for the reproductive operators are selected using standard fitness-proportionate selection; that is, a probability of selection $p_i = f_i / \sum_{j=1}^{N} f_j$ is assigned to an individual with fitness $f_i$ ($N = 20$ here).

**Fitness Function**

EFS employs a correlation-based fitness evaluation [130]. Using a measure of feature correlation $r$ based on Pearson’s correlation, a set of features $A$, a feature subset $F \in A$, and a class-to-be-predicted $C \notin A$, let the average feature-class correlation be

$$\overline{r_{Cf}} = \frac{1}{|F|} \sum_{f_i \in F} r_{Cf_i}$$

(4.1)
Feature-feature correlation is given by

\[ \overline{r_{ff}} = \frac{1}{|F| \times |F - 1|} \sum_{f_i \in F} \sum_{g_i \in (F - f_i)} r_{f_i,g_i} \]  

(4.2)

Combining the two for maximizing class-feature correlation while minimizing feature-feature correlation and weighing it with the number of features results in the following fitness function:

\[ \text{Fitness} = \frac{k \times \overline{r_{Cf}}}{\sqrt{k + k \times (k - 1) \overline{r_{ff}}}} \]  

(4.3)

**Evolution Parameters**

The initial population is of a fixed size of 20 individuals who are created using random binary strings (effectively, 20 subsets of randomly selected features from the hall of fame constitute the initial population). The GA implementation in EFS is generational; that is, after the offsprings are created using mutation and crossover, the parents die. The population size of 20 remains constant throughout the generations in EFS. The number of generations is set to 20, as well. The best individual (feature subset) is tracked over the generations and constitutes the feature subset presented to any classifier for labeling new unlabeled (testing) sequences.

**Classifiers**

The best (highest fitness) individual obtained from EFS is the feature subset presented now to a classifier. Generally, any classifier can be used, and experimentation shows that there are no significant differences among the standard classifiers. Since the Naïve Bayes Classifier is the simplest, fastest, non-parametric, and most effective when features have low correlation among them but high correlation with class, I employ Naïve Bayes as the classifier of choice.
4.3 Datasets

In this section, details of the datasets employed for this research is given.

4.3.1 HSS Dataset

The dataset used had 280 HS sequences and 737 non-HS sequences. 280 hypersensitive sequences used in the experiments were identified from throughout the human genome using an experimental methodology that identifies HS sites employing cloning and in-vivo activity of K562 erythroid cells [111]. 737 were from the same genome but that were non-hypersensitive when tested in the same cell type. Both K562 HS and non-HS sequences were similar in sequence length, with mean length 242.1 and 242.8 bp, respectively. The complete dataset is available at noble.gs.washington.edu/proj/hs.

4.3.2 Alu Dataset

Data sets of 1922 319 Alu Sequences were obtained from the NCBI website. This small subset of sequences is considered to be representative of 99% of all Alu sequences in the GenBank[REF]. The average length of these Alu sequences is approximately 300 bp. I generated equal 319 random non-Alu sequences using the average length equal to that of the Alu sequences and the composition of each base pair having the same probability as in the whole Alu sequences.

4.4 Experiments

In the experiments, I use evaluation criteria based on prediction accuracy in terms of area under the Receiver Operator Characteristic Curve (auROC) and area under the Precision Recall Curve (auPRC) [131,132]. Since both HSS and Alu sequence datasets are smaller in size, 10-fold stratified cross-validation is run to sample the training/testing. All comparisons are done running the experiments 30 times and using paired-t tests for statistical significance at 95 % confidence interval. Mean auROC and auPRC are reported.
Many of the statistical learning algorithms and kernel methods using position have a limitation sometimes that all sequences should be of equal lengths. Some of the datasets mentioned have variable sequence lengths, which results in various choices of using random alphabets as fillers for smaller sequences or eliminating the sequences below a certain threshold from comparison.

The statistical methods like HMM, Bayes Network, and MSP were carefully tuned for getting the best result by selecting parameters such as the order in the Markov chain, optimization methods, and the epsilon for error optimization. The kernel methods were optimized for soft-parameter $C$ for both weighted degree (WD) and weighted degree with shift (WDS). Feature-based methods like K-mer and Gibbs sampling used same length of motifs as EFG, i.e., between 2 to 8.

### 4.4.1 Cross-validation on Training Data Sets for HSS

I performed 10-fold cross-validation on the HSS dataset, using all the comparative methods, 30 times and report the mean auROC and auPRC. The results that are significant are shown in bold. The experiment for classifying the HS sites shows that EFG performs statistically significantly using paired-t tests at 95% confidence intervals better in both auROC and auPRC at 89.7 and 89.2 respectively as shown in Table 4.2. This seems to be the best published result on the dataset that I know of. It is interesting to see that simple compositions using spectrum k-mers are very effective in the dataset. The ability of the present evolutionary algorithm to automatically find these signals from basic compositional to complex positional and correlational gives an edge over all the algorithms.

### 4.4.2 Cross-validation on Training Data Sets for Alu Sequences

I performed 10-fold cross-validation on the Alu dataset, and report only the average auROC as an accuracy metric, since the ratio of positive and negative sequences is the same for this experiment. EFG shows mean auROC at 98.9, statistically significantly greater than
Table 4.2: auROC (area under ROC curve) and auPRC (area under PRC curve) comparison with all methods for HSS

<table>
<thead>
<tr>
<th>Feature</th>
<th>auROC</th>
<th>auPRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-mer</td>
<td>82.20</td>
<td>82.6</td>
</tr>
<tr>
<td>Gibbs Sampling</td>
<td>79.3</td>
<td>50.3</td>
</tr>
<tr>
<td>EFG</td>
<td>89.7</td>
<td>89.2</td>
</tr>
<tr>
<td>Statistical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWM-HMM</td>
<td>70.8</td>
<td>47.8</td>
</tr>
<tr>
<td>BayesNetwork</td>
<td>72.5</td>
<td>49.5</td>
</tr>
<tr>
<td>HomogenousHMM</td>
<td>82.02</td>
<td>71.5</td>
</tr>
<tr>
<td>WAM-HMM</td>
<td>80.05</td>
<td>70.0</td>
</tr>
<tr>
<td>MSP</td>
<td>85.5</td>
<td>72.9</td>
</tr>
<tr>
<td>Kernel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WeightedPosition</td>
<td>80.01</td>
<td>62.3</td>
</tr>
<tr>
<td>WeightedPositionShift</td>
<td>80.93</td>
<td>64.9</td>
</tr>
</tbody>
</table>

All others comparison made using paired-t tests at 95% confidence intervals as shown in Table 4.3. The average number of features used by EFG for 30 runs is 110, with max of 120, while other feature generation techniques like K-mers and Gibbs sampling had 65, 536 and 1213 features respectively.

4.5 Feature Analysis

4.5.1 Statistical Analysis of Features

To perform analysis on features generated by EFG, I used the Information Gain metric to get a statistical metric on the quality of the features generated. For a dataset D, with
Table 4.3: auROC (area under ROC curve) and auPRC (area under PRC curve) comparison with all methods for Alu Sequences

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>auROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-mer</td>
<td>94.20</td>
</tr>
<tr>
<td>Gibbs Sampling</td>
<td>95.2</td>
</tr>
<tr>
<td>EFG</td>
<td><strong>98.9</strong></td>
</tr>
<tr>
<td>Statistical</td>
<td></td>
</tr>
<tr>
<td>PWM-HMM</td>
<td>77.45</td>
</tr>
<tr>
<td>BayesNetwork</td>
<td>86.82</td>
</tr>
<tr>
<td>HomogenousHMM</td>
<td>93.6</td>
</tr>
<tr>
<td>WAM-HMM</td>
<td>94.59</td>
</tr>
<tr>
<td>MSP</td>
<td>93.54</td>
</tr>
<tr>
<td>Kernel</td>
<td></td>
</tr>
<tr>
<td>WeightedPosition</td>
<td>96.9</td>
</tr>
<tr>
<td>WeightedPositionShift</td>
<td>97.8</td>
</tr>
</tbody>
</table>

classes ranging from $i = 1$ to $k$, the information theory metric for entropy is given by

$$Info(D) = - \sum_{i=1}^{k} P(C_i, D) \times \log(P(C_i, D))$$ \hspace{1cm} (4.4)

For a feature $F$, having $v$ different values, the weighted sum of expected information by splitting the dataset for these features is given by

$$Info_F(D) = - \sum_{j=1}^{v} \frac{|D_i|}{|D|} \times Info(S_i)$$ \hspace{1cm} (4.5)
The information gain for the feature as compared to the whole dataset is computed by

\[ \text{InfoGain} = \text{Info}(D) - \text{Info}_F(D) \]  

(4.6)

Figure 4.4 shows the mean \text{InfoGain} for EFG is 0.017, which is almost 3 and 9 times more than that of the Gibbs sampling and K-mers respectively for hypersensitive sites. Also, the number of features generated by EFG at 45 is smaller than 1030 and 65536, as compared to Gibbs sampling and K-mers respectively. Figure 4.5 shows the mean \text{InfoGain} for EFG is 0.115 which is again approximately 3 and 1000 times more than that of the Gibbs sampling and K-mers respectively for Alu sequences. Also, the number of features generated by EFG is only 103, which is smaller than 170 and 65536 by Gibbs sampling and K-mers respectively. This shows that EFG generates fewer but statistically more discriminating features which is one of the most desired quality required of feature construction algorithms.

Figure 4.4: Information Gain Distribution for EFG-EFS features in comparison with other feature related methods like K-mers and Gibbs Sampling.
4.5.2 Biological Analysis of Features

To analyze features generated and selected from EFG, I used the entire dataset mentioned above for each application to generate features. The hypersensitive sites for K562 generated many features such as the compositional motifs like CGCG, CGCGAGA, (A/G)GG(T/G). Many positional with slight shifts like presence of short two-mers CG to 8-mer CTTCCGCC, correlational ones such as GAT and ATCT, CATTT and (G/T)GGC were analyzed in many past researches which had important biological significance like maturation, silencers and enhancer effects [133, 134]. Also, various features generated by EFG had CG patterns such as CGMS, CGMSN, and CGSBN, which confirms the past studies that HS sites are rich in CG nucleotides [6].

The Alu Sequences also generated compositional motifs like AAAAAA, AAAAT, AGCCT,
CCCAG, CCTGT which are known signals in the Alu repeats.[REF]. As further analysis, note that EFG found some interesting combinations of disjunction features, composed of three block signals like correlational signals CCTR, AAT, shift 3, correlational features CA, GY, shift 3, and a compositional feature TGG. I ran clustal alignment on the whole dataset and found that these signals were indeed over-represented as illustrated in Figure 4.6. This also highlights the importance of using ambiguous symbols in the representation of EFG for matching pyridines. EFG also found interesting combinations of disjunctive features like CCTGG, CTGGGG, and GAGGC which were considered individually important signals, showing the strength of the algorithm to combine presence of lower level signals to form interesting higher order features.

4.6 Conclusion

This chapter shows that the EFG algorithm with some basic feature selection and generic classifiers like Naïve Bayes can perform significantly better as compared to complex statistical, feature-based, and kernel-based methods in the two complex sequence classification problems. It is also shown statistically using information gain that the features generated by EFG for both the problems have higher discrimination power. Using clustal and other well known biological feature analysis algorithms it was shown that the features generated by EFG capture more complex local and global interactions in the dataset without domain expertise.
Figure 4.6: Sample showing a complex feature from the EFG algorithm for Alu sequence (top) and the clustal alignment showing the same overrepresented signals (bottom).
Chapter 5: EFG for Promoter Region Identification

One of the outstanding challenges in mapping out eukaryotic DNA is the identification of promoter regions, whose identification is an important step towards understanding the regulation of differential transcription. The complex gene-specific architecture of promoter regions in the sequences makes this sequence classification problem a difficult one. Can EFG-based feature generation help find the complex signals required to identify promoter regions? In this chapter, I will give the background of the promoter region identification problem and the application of EFG to solve it; experiments and comparative analysis follow.

5.1 Promoter Region Identification

One of the most important yet poorly understood functional elements in DNA is the promoter region. Promoters play the important role of initiating and regulating transcription. The promoter regions precede the protein-encoding genes and contain specific signals tied to specific classes of genes and transcription regulation. Annotating promoter regions in genomic sequences, currently missing from annotated genomic sequences, is an important task to improve the detection of specific classes of genes [135].

The architecture of promoter regions is very complex in eukaryotic DNA. Eukaryotic promoters, often referred to as PolII promoters, bind RNA polymerase II, among many other transcription factors. Transcription of a gene starts only when all transcription factors bind onto the promoter resulting in a productive complex [136] as shown in Figure 5.1. What makes promoters unique to specific gene classes and programs of regulation is their content and arrangement of transcription factor binding sites. Elements such as the TATA box and transcription start signals (TSS) are shared by only 30–50% of promoters [135].
General purpose promoter prediction methods, summarized and compared in [135], can recognize only about 50% of promoters with a false positive rate of about 1 per 700–1000 base pairs (bp). Employing a Hidden Markov Model (HMM) and incorporating prominent motifs, such as the TATA box, DRE, INR, DPE, and MTE, results in recognition of 50% of eukaryotic promoter regions and a 1/849 bp false positive rate [138]. A combination of neural networks and genetic algorithms in the Promoter 2.0 method increases prediction accuracy to 67%. Non-linear time series descriptors and non-linear machine learning algorithms such as Support Vector Machines (SVM) in [139] result in 84% accuracy in 7-fold cross-validation. Other methods that focus on promoters of specific species achieve similar accuracies [140]. Can EFG-based feature generation algorithm find discriminative features that help classifying promoter regions? Is EFG-based feature generation technique competitive with other state of the art techniques? Can EFG algorithm find complex features which have been found through complex wet labs and other expert based systems? I will try to address these questions through a series of experiments in this chapter.
5.2 EFG Algorithm

In this section, I provide implementation details of EFG algorithm, customized for the promoter region identification problem. Since most datasets are smaller in size, there needs to be some feature selection/reduction technique. In this chapter, I implemented statistical subset feature selection detailed below.

5.2.1 Feature Representation

EFG uses the same IUPAC code characters as the symbols for motifs, for the same reason that using ambiguous character helps the matching patterns.

5.2.2 EFG Algorithm

Parameters  Population size is kept large at $N = 5000$, and hall of fame is $\ell = 250$.

Breeding Operators  Motif, Positional, and Shift Mutation all are employed with a probability of 0.1. Subtree crossover is employed, as no significant difference was found in performance when compared to homologous crossover. Subtree crossover was used at a probability of 0.7.

5.2.3 Post EFG Feature Selection

The set of features in the hall of fame is further reduced through the filter selection technique proposed in [141]. Briefly, given a set of features, the technique selects a subset of relevant but redundant features. A feature is deemed “good” and selected if it is highly correlated with the class but not highly correlated with the rest of the features. The technique combines the advantage of the classical linear correlation approach with information theory through an entropy-based approach. A symmetrical uncertainty (SU) measure evaluates the goodness of a feature: $\text{SU}(X, Y) = 2 \frac{\text{IG}(X|Y)}{H(X) + H(Y)}$. In this equation, $X$ and $Y$ refer to features. $H$ refers to entropy and measures the uncertainty of a random variable: $H(X) = - \sum P(x_i) \cdot \log_2(P(x_i))$ and $H(X|Y)$, the entropy of $X$ after observing values of $Y$, is measured as
\[ -\sum_j P(y_j) \cdot \sum_i P(x_i|y_j) \cdot \log_2(P(x_i|y_j)) \]. The information gain \( IG(X|Y) \), equivalent to \( H(X) - H(X|Y) \), measures the amount by which the entropy of \( X \) decreases from additional information about \( X \) provided by \( Y \). [142].

5.2.4 Support Vector Machines as Classifier

The features obtained after the feature selection technique allow transforming input sequences into feature vectors. The feature vectors associated with training sequences are then employed to train an SVM classifier and further estimate the discriminating power afforded by the top EFG-obtained features. The reason for employing an SVM classifier is due to SVMs becoming one of the most popular machine learning techniques. SVMs have shown great success in difficult classification problems and have a solid theoretical foundation in statistical learning theory [143,144].

5.2.5 Implementation Details

EFG is implemented using the standard GP infrastructure provided in ECJ software [145]. The feature selection technique was implemented using Weka [146]. Sequence matching and pattern recognition were implemented using BioJava [147]. SVM training and classification was implemented using LibSVM [148]. The experiments reported here use a Radial Basis kernel function (RBF). The kernel parameters and the SVM cost function are tuned through the standard grid search mechanism [149].

5.3 Datasets

The input dataset consists of promoter and non-promoter sequences obtained from various species. The validation of the method focuses on three species: plant, human, and drosophila.

Plant Dataset The plant promoter dataset consists of 305 promoter sequences obtained from the PlantProm database. The database contains annotated and non-redundant PolII
promoter sequences with experimentally-determined TSS from various plant species [150]. The length of these promoter sequences upstream of the TSS is 251 bp. Position 201 corresponds to the TSS, and promoter sequences occupy the region $[-200 : +51]$. The plant negative data also consists of 305 non-promoter sequences. To ensure that these sequences do not contain promoter regions, I extract them from known coding sequences of the same length, 251 bp, obtained from the Genome database of the NCBI (http://www.ncbi.nlm.nih.gov/). Coding sequences are known not to contain promoter signals. Since the plant promoter data consists of various plant organisms, I make sure that the plant non-promoter data reflects the same distribution of organisms.

**Homo Sapiens and Drosophila Datasets** Datasets of 1922 Homo Sapiens and 1863 Drosophila promoter sequences were obtained from the Eukaryotic Promoter Database [151]. Similarly, the non-promoter data for each species is constructed by extracting non-promoter sequences of the same length, 251 bp from coding sequences. Coding sequences for Homo Sapiens and Drosophila were obtained from the Berkeley Drosophila Genome Project [152].

### 5.4 Experiments

Four sets of experiments were conducted. First, three different SVM models are trained separately on the datasets described above for plant, Homo Sapiens, and Drosophila. 7-fold cross-validation is performed in order to evaluate the classification performance. Essentially, the dataset is randomly divided into 7 subsets of equal size. The model is trained on $6/7$ of the data and tested on the remaining subset. The area under the receiver-operating-characteristic (ROC) curve is reported as an average over the 7-fold validations.

The second and third sets of experiments test the SVM and EFG features on testing datasets extracted from the input datasets. In the second set of experiments, 50 promoter and 50 non-promoter sequences are withdrawn from the plant, Drosophila, and Homo Sapiens datasets to employ as datasets. The performance of the SVM models trained on the remaining dataset for each species is compared in detail with the K-MerSVM in [30]. The
third set of experiments then tests the features on a plant dataset of 10 promoter and 40 non-promoter sequences extracted from the plant dataset before training the SVM on the rest of the sequences. The goal is to measure performance when negative sequences significantly outnumber positive sequences. Finally, the fourth set of experiments compare performance to that of state-of-the-art methods on the plant testing dataset of 50 promoter and 50 non-promoter sequences extracted from the plant input dataset as described above.

Performance was quantified in terms of the standard measures of sensitivity, specificity, precision, recall, and false positive rate (FPR). Sensitivity is measured as \( \frac{TP}{TP + FN} \), and specificity is \( \frac{TN}{FP + TN} \), where \( TP, TN, FP, \) and \( FN \) refer to the number of true positives, true negatives, false positives, and false negatives. Recall is \( \frac{TP}{TP + FN} \), precision is \( \frac{TP}{TP + FP} \), and FPR is \( \frac{FP}{FP + TN} \).

Also, all experiments were run 30 times and the statistically significant with 95% confidence improvements are highlighted with bold-face.

5.4.1 Cross-validation on Training Datasets

Table 5.1 shows the area under the (ROC) curve (auROC) obtained after a 7-fold cross-validation on each of the three species’ promoter training datasets. The results obtained when employing the features under consideration are compared with those reported by the K-merSVM method in [30]. It is worth mentioning that the auROCs reported for the method are additionally averaged (shown as \( \text{auROC}_\mu \)) over 30 different sets of features obtained from 30 independent EFG runs. Standard deviations (shown as \( \text{auROC}_\sigma \)) are also shown. Since EFG is by nature a stochastic search, I execute it 30 different times in order to obtain 30 different halls of fame. Applying the feature selection technique results in 30 different sets of features.

The results shown in Table 5.1 make the case that EFG features allow obtaining consistent accuracies in the 94–95% regardless of the species. The range of accuracies for K-merSVM is larger, in the 84–94% range depending on the species. The performance increase by EFG is most significant for the plant dataset, followed by the Homo Sapiens
Table 5.1: Cross-validation auROC comparison between K-MerSVM and EFG.

<table>
<thead>
<tr>
<th>Organism</th>
<th>K-MerSVM auROC</th>
<th>EFG auROC $\mu$</th>
<th>EFG auROC $\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plant</td>
<td>83.8</td>
<td>94.6</td>
<td>0.4</td>
</tr>
<tr>
<td>Homo Sapiens</td>
<td>91.25</td>
<td>94.1</td>
<td>0.6</td>
</tr>
<tr>
<td>Drosophila</td>
<td>94.82</td>
<td>95.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

and Drosophila data sets. Obtained $\alpha$ values from EFG were consistently 0.0001 for each training dataset.

5.4.2 Evaluating Performance on Testing Datasets

The performance of EFG is further tested and compared with K-MerSVM over the testing datasets detailed above. Table 5.2 shows various statistical measurements for K-Mer SVM and EFG when applying trained SVMs over the testing datasets of each species (Plant, Drosophila, and Homo Sapiens). The results show improvements over all measurements when employing EFG features. For instance, testing the SVM on the Plant testing dataset shows a 12% increase in sensitivity, 6% increase in specificity, 7% increase in PPV, and 9% increase in NPV compared to the respective measurements obtained from K-merSVM. Additionally, reductions to plant FNR and FPR by 4% to 10% and 2% to 14% show low false positive and false negative predictions by the model on unseen sequences.

Performance is further tested when negative sequences outnumber positive sequences in the plant testing dataset (constructed as described above). Precision versus recall values are measured and shown in the curve in Figure. 5.2. The break-even point on the curve is 77% for K-merSVM and 88% for EFG, showing that a high number of true positives is predicted even with a large negative dataset. Figure. 6.4 also shows the FPR versus recall curves. At very high recall, the FPR of K-merSVM is higher than for EFG, suggesting that EFG features allow better classifying a dataset where negative sequences significantly
Table 5.2: Comparison between K-Mer and EFG over testing data sets for Plant, Drosophila, and Homo Sapiens with suffixes P, D and H respectively.

<table>
<thead>
<tr>
<th>Organism</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>TN</th>
<th>Sn</th>
<th>Sp</th>
<th>Cor.</th>
<th>FPR</th>
<th>FNR</th>
<th>PPV</th>
<th>NPV</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-K-Mer</td>
<td>43</td>
<td>5</td>
<td>7</td>
<td>45</td>
<td>0.86</td>
<td>0.9</td>
<td>0.761</td>
<td>0.1</td>
<td>0.14</td>
<td>0.89</td>
<td>0.86</td>
</tr>
<tr>
<td>P-EFG</td>
<td>49</td>
<td>2</td>
<td>1</td>
<td>48</td>
<td>0.98</td>
<td>0.96</td>
<td>0.94</td>
<td>0.04</td>
<td>0.02</td>
<td>0.96</td>
<td>0.98</td>
</tr>
<tr>
<td>D-K-Mer</td>
<td>48</td>
<td>4</td>
<td>2</td>
<td>46</td>
<td>0.96</td>
<td>0.92</td>
<td>0.881</td>
<td>0.08</td>
<td>0.04</td>
<td>0.92</td>
<td>0.95</td>
</tr>
<tr>
<td>D-EFG</td>
<td>49</td>
<td>1</td>
<td>1</td>
<td>49</td>
<td>0.98</td>
<td>0.98</td>
<td>0.96</td>
<td>0.02</td>
<td>0.02</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>H-K-Mer</td>
<td>44</td>
<td>4</td>
<td>6</td>
<td>46</td>
<td>0.88</td>
<td>0.92</td>
<td>0.801</td>
<td>0.08</td>
<td>0.12</td>
<td>0.92</td>
<td>0.8</td>
</tr>
<tr>
<td>H-EFG</td>
<td>48</td>
<td>5</td>
<td>2</td>
<td>45</td>
<td>0.96</td>
<td>0.9</td>
<td>0.86</td>
<td>0.1</td>
<td>0.04</td>
<td>0.90</td>
<td>0.95</td>
</tr>
</tbody>
</table>

outnumber positive sequences.

Figure 5.2: Precision versus recall for K-merSVM and EFG.
5.4.3 Comparison with State-of-the-Art Methods

State-of-the-art methods in promoter prediction, such as (NNP) [152], SoftBerry [153], ProScan [154], Dragon Promoter Finder [155], Promoter2.0 [156], and K-MerSVM [30] report various measurements on the plant testing dataset. This provides an opportunity to directly compare the performance of EFG with these methods, shown in Table 5.3, which make the case that EFG is consistently among the top-performing methods in the various measurements. For instance, compared to SoftBerry, EFG improves specificity by 16%.
Table 5.3: Comparison between EFG and other methods over the plant testing dataset.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP Threshold(0.8)</td>
<td>68</td>
<td>76</td>
<td>0.44</td>
</tr>
<tr>
<td>SoftBerry</td>
<td>88</td>
<td>90</td>
<td>0.78</td>
</tr>
<tr>
<td>ProScan (v1.7)</td>
<td>0</td>
<td>100</td>
<td>N/A</td>
</tr>
<tr>
<td>DFB (v1.4)</td>
<td>12</td>
<td>100</td>
<td>0.25</td>
</tr>
<tr>
<td>Promotor2.0</td>
<td>0</td>
<td>78</td>
<td>N/A</td>
</tr>
<tr>
<td>Kmer-SVM</td>
<td>86</td>
<td>90</td>
<td>0.77</td>
</tr>
<tr>
<td>EFG</td>
<td><strong>98</strong></td>
<td><strong>96</strong></td>
<td><strong>0.94</strong></td>
</tr>
</tbody>
</table>

5.5 Feature Analysis

5.5.1 Statistical Analysis of Features

The F-score that SVM models associate with support vectors provides another measure of the relative importance of features. The F-score measures the discrimination of two sets of real numbers. Given training vectors $x_k$, where $k \in \{1, \ldots, m\}$, with $n_+$ and $n_-$ denoting the number of positive and negative instances, respectively, the F-score of the $i^{th}$ feature is defined as:

$$F(i) = \frac{(\bar{x}_i^+ - \bar{x}_i)^2 + (\bar{x}_i^- - \bar{x}_i)^2}{\frac{1}{n_+-1} \sum_{k=1}^{n_+} (x_{k,i}^+ - \bar{x}_i^+)^2 + \frac{1}{n_-+1} \sum_{k=1}^{n_-} (x_{k,i}^- - \bar{x}_i^-)^2}$$

In the above equation, $\bar{x}_i$, $\bar{x}_i^+$, and $\bar{x}_i^-$ are the average of the $i^{th}$ feature of the whole, positive, and negative datasets, respectively. Similarly, $x_{k,i}^+$ is the $i^{th}$ feature of the $k^{th}$ positive instance, and $x_{k,i}^-$ is the $i^{th}$ feature of the $k^{th}$ negative instance. The numerator measures the discrimination between the positive and negative sets, whereas the denominator measures the one within each of the two sets.

The F-score is often used as a feature selection criterion, but I use it here as an additional...
measure by which to show the relative importance of the top 100 features obtained after the correlational feature selection technique. Fig. 5.4 compares the F-scores of the features obtained by the approach to the F-scores of the 128 4-mer motif features employed in the SVM classification in [30]. The mean F-Score for EFG is approximately 500% of the K-mers showing the presence of more complex signals in promoter region identification and the ability of EFG to find them.

![Figure 5.4: Left: F-scores of top 95 features obtained by the EFG. Right: F-scores of 128 4-mer features employed in [30].](image)

<table>
<thead>
<tr>
<th>Means and Std Deviations</th>
<th>Std Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>Number</td>
</tr>
<tr>
<td>EFG</td>
<td>95</td>
</tr>
<tr>
<td>KMER</td>
<td>128</td>
</tr>
</tbody>
</table>

5.5.2 Biological Analysis of features

Inspecting the top discriminating features reveals many well known 4-mer compositional motifs, such as `AAAA`, `TTTT`, `AAAT`, and `TATA`, in agreement with the analysis of 4-mers in [30]. In addition to these 4-mer motifs, many 5-mer compositional motifs are observed, such as `GAGCT`, `GATGA`, `GAAGG`, and `TCATG` that have not been observed.
by previous methods but are predominant among the top-ranking features. Other interesting features emerge. For instance, correlational features like \((\text{AND (Correlate(Motif-3 GCA,Motif-2 AT) 26,2)})\), are present among the top top 100 features. Such features encode interesting signals of the correlational presence of two or more motifs in promoter sequences. Some of the complex motifs observed such as the CGTCA and TCTCCCT matched the responsive element and the light response element in the dataset.

Due to the fact that non-promoters were obtained from coding sequences, many other interesting features are observed. For instance, \((\text{Not(AND(Matches(Motif-3 TTC) (Matches(Motif-3 CGA)))})\) were examples of discriminating features present in coding sequences but not present in promoter sequences. The top 100 features contained additional complex disjunctive features, such as \((\text{AND ( AND ( Not (MatchesAtPosition (Motif-3 GCG) 103)) (OR (Matches (Motif-2 CC))(OR (MatchesAtPosition (Motif-3 GCA) @ 26) (MatchesAtPosition(Motif-6 GCGTTA) 204))))}\).

5.6 Conclusion

In this chapter, application of EFG to complex promoter region identification was studied. It can be concluded that employing the EFG for feature generation results in features that significantly improve the problem of promoter prediction. The detailed comparison with well known state of the arts on various datasets show that EFG improves promoter prediction in various metrics like accuracy, sensitivity, specificity, and low false positives which are very important in the promoter prediction problem.

The detailed statistical analysis using FScore shows that features generated from EFG have the unique characteristics of being small and carrying larger discriminative power. The biological analysis of EFG features shows that EFG can not only automatically construct simpler well known motifs for promoter identification but much more complex signals capturing positional, correlational, and conjunctive properties present in the raw data.
Chapter 6: EFG for Splice Site Prediction

Splice site prediction is considered one of the most difficult sequence classification problem because of the complexity of the features needed to obtain high classification accuracy and precision. This chapter will try to answer at a very high level whether EFG can be applied to this complex gene finding problem, beginning by introducing the splice site prediction problem, then giving details of EFG customizations for this application, experimental validation on various known datasets with state-of-the-art algorithms comparisons, and finally some analysis on the EFG features with respect to splice sites.

6.1 The DNA Splice Site Prediction Problem

Transcription of a eukaryotic DNA sequence into messenger RNA (mRNA) occurs only after enzymes splice away non-coding regions (introns) from the precursor (pre-mRNA) sequence to leave only coding regions (exons) as shown in Figure 6.1. For this reason, prediction of splice sites is a fundamental component of the gene-finding problem [157]. An acceptor splice site marks the start of an exon; a donor splice site marks the end. The sites have different consensus sequences. AG is a consensus dinucleotide among canonical acceptor splice sites, whereas GT is a consensus among canonical donor splice sites. AG and GT cannot be used as features due to their abundance in non-splice site sequences. Nucleotide composition and coding and non-coding length and composition also do not make for discriminating features [95]. Early approaches employing positional probabilities fared poorly [46].

The state-of-the-art methods in splice site prediction are algorithms that are kernel based, feature based or statistical model based. Kernel-based methods like the ones in [14, 158, 159] achieve some of the best performance in recognition of splice sites in a diverse
Feature-based methods focus on identifying discriminating features and using known classifiers to generate classification models. The feature generation algorithm (FGA) in [7] is one of the most successful feature-based classification methods for splice site prediction. FGA expands upon the list of features of an earlier hallmark method, Gene-Splicer [95], which included only position-specific nucleotides and upstream/downstream 3-mers. FGA conducts an iterative and exhaustive search over features of different types but have to limit the combinations because of the exponential search space. Statistical methods like position-specific scoring matrix (PSSM), the weight array model (WAM), complex models like Bayesian networks and Markov Random Fields (MRFs) have been applied for splice sites [51–54].

Can EFG improve upon the FGA’s iterative search method in finding features for splice sites? Can EFG improve classification of splice sites as compared to the state-of-the-art algorithms in this widely studied problem? Can the EFG algorithm be used for gene annotation from the predictive model learnt? Can EFG generate features that are representative...
of splice site signals normally found through wet lab or expert analysis? These are the questions the chapter will try to answer.

6.2 The EFG Algorithm

Various choices of parameters, representations and methodology are explained in this section.

**Feature Representation**  EFG uses the same IUPAC code characters as the symbols for motifs, for the same reason that using ambiguous character helps the matching patterns.

**EFG parameters**  EFG uses population size of $N = 5,000$ and hall of fame as an external memory pad of $\ell = 250$ for every generation.

**Genetic Operators**  EFG uses positional mutation, shift mutation, and motif mutation each with a 0.1 probability as the asexual perturbation operators. EFG uses subtree crossover with max depth of 50 as the recombination operator with probability of 0.7.

6.2.1 Post EFG Feature Selection

The set of features in the hall of fame can be further narrowed through Recursive Feature Elimination (RFE) in the context of SVM classification [8, 160, 161]. The main idea in RFE is to start with a large feature set and gradually reduce this set by removing the least successful features (according to some metric) until a stopping criterion is met. RFE is employed in the context of SVM classification, as in [162], using precision as the metric by which to determine whether a feature can be removed.

6.2.2 Support Vector Machines as Classifier

The EFG obtained feature allows transforming input sequences into feature vectors. The feature vectors associated with training sequences are employed to train an SVM classifier and estimate the discriminating power afforded by the top EFG features.
6.3 Datasets

The first dataset, known as NN269, consists of 1324 confirmed acceptor site sequences, 1324 confirmed donor site sequences, 5552 false acceptor site sequences and 4922 false donor sites sequences obtained from 269 human genes [163]. Each of the false acceptor/donor sites also has AG/GT in the splicing junction with approximately same distribution of alphabets. The window size used for acceptor sequence is 90 nt $-70$ to $+20$ with consensus AG at positions $-69$ and $-70$ and includes the last 70 nucleotides of the intron and first 20 nucleotides of the next exon. The window size for donor sequence is 15 nt, from $-7$ to $+8$, with consensus GT at positions $+1$ and $+2$ and includes the last 9 bases of the exon and first 6 bases of the next intron. The most common way of splitting data set into a training set and a testing set [163] was followed. The training data set had 1116 true acceptor, 1116 true donor, 4672 false acceptor, and 4140 false donor sequences. The test data set had 208 true acceptor, 208 true donor, 881 false acceptor, and 782 false donor sequences.

The next dataset used is from 5057 human pre-mRNA sequences in the NCBI RefSeq collection that are annotated with exon start (acceptor) and end (donor) positions. The annotations are used to extract 51,008 positive (containing splice sites) and 200,000 negative sequences as in [7,8].

Acceptor and donor splice site sequences (25,504 acceptor and 25,504 donor) consist of 162 nucleotides each, 80 nucleotides upstream of the annotated AG or GT dinucleotide, respectively, and 80 downstream (80+AG/GT+80). Negative sequences are 162 nucleotides long and centered around randomly selected AG/GT dinucleotides not annotated as splice sites. The significant difference in size between the negative and positive training data sets makes it harder for a classifier to obtain a high number of positive matches at random [7,8].

The EFG features are further validated on a testing data set, the B2hum 1115 human pre-mRNA sequences employed to train GeneSplicer [95].

The worm data set is extracted from the worm genome and prepared in [14]. The genome is aligned through blat with all known cDNA sequences available at http://www.
wormbase.org and all known EST sequences in [164]. A splicing graph representation built over the clustered alignments reveals a list of acceptor and donor splice sites. Using this list, 64,844 donor and 64,838 acceptor splice site sequences are extracted. Each sequence is 142 nucleotides long (60+AG/GT+80) and centered around splice sites. Negative training sequences are also 142 nucleotides long and centered around non-splice sites in intronic regions.

6.4 Experiments

Performance Measurements I measure performance in terms of 11-point average precision (11ptAVG), false positive rate (FPR), area under receiver-operating-characteristic curve (auROC), and area under precision-recall curve (auPRC). An SVM labels and orders data from most to least confident. Given a confidence threshold, only the data above that threshold can be considered correctly labeled. For any recall ratio, precision can be calculated at the threshold that achieves that recall ratio [165] for a definition of recall and precision. The 11ptAVG is the average of precisions calculated at 11 recall values \{0\%, 10\%, \ldots, 100\%\}. In addition to 11ptAVG, Precision-Recall Curves (PRCs) are employed to show the ability of EFG to discriminate true splice sites from other sequences. FPR is also computed for recall values by varying the confidence threshold to employ FPR-recall curves and show that EFG makes very few mistakes. All the runs are repeated 30 times and statistical significance is computed at the 95% confidence interval and highlighted in bold when applicable.

6.4.1 NN269 Splicing Sequences Comparisons

NN269 is a widely used dataset for splice site comparisons used to compare the EFG algorithm with different state-of-the-art mentioned in the related works chapter. The three top-level categories that EFG algorithms are compared with are feature-based, statistical-algorithm-based and kernel-based algorithms. For feature based, spectrum features and over-represented motifs from alignments using Gibbs sampling with simple base classifier
such as Naïve Bayes. For statistical learning, a comprehensive comparison with all the state-of-the-art techniques like PWM, Bayes Tree Network with PWM, Markov Chain (MC) and Maximum Separable Posterior (MSP) is made. Finally, kernel methodologies like weighted degree positional kernel (WD) and weighted positional kernel with shift (WDS) are used with SVM.

Table 6.1: Comparison of different methods for NN269 Splicing Sequences

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ACCEPTOR</th>
<th>DONOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature</td>
<td>auROC</td>
<td>auPRC</td>
</tr>
<tr>
<td>K-mer</td>
<td>63.3</td>
<td>75.5</td>
</tr>
<tr>
<td>Gibbs Sampling</td>
<td>62.8</td>
<td>72.4</td>
</tr>
<tr>
<td>EFG</td>
<td>97.7</td>
<td>94.3</td>
</tr>
<tr>
<td>Statistical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWM</td>
<td>97.1</td>
<td>90.6</td>
</tr>
<tr>
<td>BayesNetwork</td>
<td>97.25</td>
<td>90.6</td>
</tr>
<tr>
<td>HomoHMM</td>
<td>59.2</td>
<td>26.3</td>
</tr>
<tr>
<td>InHomoHMM</td>
<td>96.78</td>
<td>88.41</td>
</tr>
<tr>
<td>MSP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WD Kernel</td>
<td>98.16</td>
<td>92.53</td>
</tr>
<tr>
<td>WD-S Kernel</td>
<td>98.65</td>
<td>94.36</td>
</tr>
</tbody>
</table>

The NN269 splice-site classification experiments didn’t show any one method to perform statistically significantly better, which is consistent with many past researches on that
dataset [166]. The EFG performance on the auROC metric was comparable to the state-of-the-art WDS, WD, and MSP with a high value among the top at 97.7 and 98.2 for acceptors and donors respectively. The EFG algorithm was second best in average auPRC, which is an important metric when the dataset is unbalanced at high value of 94.33 and 82.81 for acceptor and donors respectively.

6.4.2 Precession-Recall on Human Sequences

Next, EFG algorithm performances with state-of-the-art feature generation method FGA and probabilistic model based Genesplicer are compared. Figure 6.2 plots and compares precision values corresponding to 11 recall points among GeneSplicer, FGA, and EFG. Precision values of EFG are averaged over 30 runs. Standard deviations are shown as error bars. Figure 6.2 shows significant differences between EFG, GeneSplicer, and FGA in all precision values calculated at the 11 recall points. The break-even points on the PRCs for acceptor data are 54.9%, 67.8%, and 91.3% for GeneSplicer, FGA, and EFG, respectively. The break-even points for donor data are 58.7%, 66.7%, and 91.2% for GeneSplicer, FGA, and EFG, respectively. EFG shows significant improvements of 23.5% and 24.5% in the break even values for acceptor and donor splice sites, respectively. Table 6.2, which summarizes the PRCs by comparing 11ptAVG values, shows similar results. EFG outperforms GeneSplicer and FGA with 11ptAVG values of 94.89% and 93.69% for acceptor and donor data, respectively. Paired t-test shows the 11ptAVG values are statistically significant ($\alpha = 0.005$).

6.4.3 Training-Testing on B2HUM Sequences

I next analyze the performance over the B2hum testing data set using the trained model from last section. The testing is done by running the test set using window of 162 nucleotides is scanned with overlap of 161 nucleotides over each pre-mRNA sequence to obtain shorter sequences for classification. The SVM trained over the human splice site data set is then employed to classify each of the shorter sequences. AUC, the area under the receiver
Figure 6.2: Precision values, plotted over recall points (left: acceptor, right: donor). Values are averages over 30 EFG runs. Error bars are 95% confidence intervals of standard deviations.

operating characteristic (ROC) curve for EFG over acceptor sequences is 99.41% compared to 99.37% and 98.71% for FGA and GeneSplicer, respectively. The EFG AUC score over donor sequences is 99.39% compared to 99.25% and 98.58% for FGA and GeneSplicer, respectively.

Precision-Recall Curves are shown in Figure 6.3. The break-even points on the curves for acceptor data are 55.2%, 67.9%, and 77.7% for GeneSplicer, FGA, and EFG, respectively. The break-even points for donor data are 58.53%, 67.2%, and 78.11% for GeneSplicer, FGA, and EFG, respectively. EFG shows significant improvements of 23.5% and 24.5% in the break even values for acceptor and donor splice sites, respectively. EFG shows improvements of 9.8% and 10.9% in the break even values for acceptor and donor splice sites, respectively. These results show that all three methods achieve lower precision on the testing data compared to the results on the training data. On both training and testing data, EFG achieves higher precision.

Figure 6.4 compares the FPR vs. recall curves among EFG, FGA, and GeneSplicer.
Table 6.2: Comparison of 11ptAVG data. The average and standard deviation in column 4 are obtained over 30 EFG runs.

<table>
<thead>
<tr>
<th></th>
<th>GeneSplicer</th>
<th>FGA</th>
<th>EFG ($\mu$, $\sigma$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptor</td>
<td>81.89</td>
<td>92.08</td>
<td>94.89, 0.35</td>
</tr>
<tr>
<td>Donor</td>
<td>80.1</td>
<td>89.08</td>
<td>93.86, 0.57</td>
</tr>
</tbody>
</table>

Figure 6.3: Precision over recall (left: acceptor, right: donor), plotted for the B2hum testing data set.

At 95% sensitivity, EFG performs similar to FGA with an FPR of 3.7% over FGA’s FPR of 3.3%. Both FPR values are significantly better than the 6.2% achieved by GeneSplicer. Having low FPR at high recall is particularly important when classifying testing data where the negative sequences significantly outnumber positive sequences.
6.4.4 Gene Annotation

The results of the classification can also be employed to annotate the pre-mRNA sequences with splice site information. For the gene annotation experiment, I trained acceptor and donor models using EFG on the 5,057 human pre-mRNA sequences and tested it on all the chromosomes from the B2HUM dataset. For each chromosome for annotation, 162 window length of sequence was considered and run through both acceptor/donor models to get the probability score. The false positives as illustrated in two samples shown in Figures 6.5, 6.6 were very low and the true positives had significantly high scores as compared to decoy splice sites in these sequences. These results further support the prediction power of the method and the general applicability of EFG for the purpose of annotation.

6.4.5 Cross-validation on Worm Sequences

Figure 6.7 compares EFG to state-of-the-art kernel methods WD and WDS in [14] on 40,000 randomly sampled sequences from the worm data set in terms of PRCs obtained after the
five-fold cross validation (acceptor and donor results are shown separately). The break-even points on the curves for acceptor data are 81.37%, 86.89%, and 91.1% for WD, WDS, and EFG methods, respectively. The break-even points for donor data are 86.2%, 86.4%, and 90.34% for the three methods, respectively. EFG shows improvements of 4.21% and 3.94% in the break-even values obtained over the acceptor and donor data, respectively. These results make the case that similar or slightly better results are obtained with EFG algorithm.

These results make the case that EFG performs very well even when trained over small-size data sets. This is also demonstrated in Table 6.3, which summarizes the performance through measurements of auROC and auPRC values comparing all the feature, kernel and statistical methods.
6.5 Feature Analysis

6.5.1 Statistical Analysis of Features

It is interesting to analyze the type distribution of the top features obtained by EFG and measure the contribution of each type. The hall of fame features were divided into three types or subsets. One subset consists of all compositional features. The second subset consists of all region-specific compositional, positional, and correlational features. The third and final subset contains all remaining features and consists of conjunctive and disjunctive features. Table 6.4 breaks down the distribution of features into these three subsets.

The contribution of each feature subset to the performance detailed above is estimated
by associating an IG value to each subset. The IG value of a subset sums the IG values of
the features in a subset, assuming naive Bayes independence. The distribution of IG values
is also shown in Table 6.4. Evaluation of the features on acceptor and donor data is kept
separate.

A closer inspection of the hall of fame reveals the fittest features are complex rule sets
combining various positional, shift-positional features in conjunction and disjunction.

### 6.5.2 Biological Analysis of Features

The fittest EFG features contain useful biological signals reported around splice sites [7, 8, 95]. Known signals in a typical pre-mRNA include the branch site, the pyrimidine-rich
region, splice site consensus signals, and exonic splicing enhancers.

The mammalian branch-site signal is degenerate and shows low levels of purifying selection [167]. To identify such signals, I search for compositional features of 6 nucleotides 40 to
20 nucleotides upstream of the acceptor splice site. The hall of fame contains such composi-
tional features over motifs **CTGACC, CCTGAC, CTTTT**, which were also reported
in [8] as top discriminative motifs. EFG features additionally capture the acceptor splice site pyrimidine tract interval. Well-known positional tetramers, such as CTGA, CTTT, CTTA, and TTTT, are present in the hall of fame and have high fitness values when evaluated over the acceptor training data.

Studies have suggested a potential role for the GGG and GGGG motifs in splicing [168]. The role of these motifs is validated by EFG. The hall of fame contains compositional and positional features over these motifs. These features have high fitness values when evaluated over donor training data. Additionally, many A/C-rich motifs, such as CACA, GCCCAA, CATTCA, CCTACA, can be found among EFG fittest features. Such motifs, originally described in [169] and additionally discovered in [8], have not been extensively characterized.
Table 6.4: IG sums of subsets of features evaluated over acceptor (top) and donor data (bottom).

<table>
<thead>
<tr>
<th>Subset</th>
<th>Acceptor Nr.</th>
<th>IG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compositional</td>
<td>600</td>
<td>2.53</td>
</tr>
<tr>
<td>Positional, Correlational, Regional</td>
<td>2451</td>
<td>10.34</td>
</tr>
<tr>
<td>Conjunctive and Disjunctive</td>
<td>1949</td>
<td>21.78</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subset</th>
<th>Donor Nr.</th>
<th>IG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compositional</td>
<td>738</td>
<td>4.22</td>
</tr>
<tr>
<td>Positional, Correlational, Regional</td>
<td>2791</td>
<td>11.43</td>
</tr>
<tr>
<td>Conjunctive and Disjunctive</td>
<td>1471</td>
<td>36.23</td>
</tr>
</tbody>
</table>

The NN269 dataset generated positional features (C/A)AGGTAAG and (T/C)(T/C) CCAGGT matching the donor and acceptor consensus sequences exactly. Interesting complex conjunction feature combining three positional features CG, GA and AG at around 10 nt to 17 nt in acceptor region matched exactly with known acceptor region signals in previous research [170]. The Worm dataset generated many regional features which are characteristic of splicing region like the 7mer motifs GGTAAGT, AGGTAAG, GGTAGGT around -43 nt, matching the donor consensus AGGTAAGT. Another important positional feature in the region −18 nt to −14 nt consisting of TAAT which is a well known branch site signal [170]. Shift-Positional features around −3 nt consisting of motifs TTTCAGG and TTTCAGA matching the acceptor consensus TTTCAG(A/G) exactly.
Chapter 7: EFG for Time series classification

Many real world applications and problems have time dependency and need time series analysis. Learning classification models for time series based applications is one of the challenges facing the machine learning community. For effective classification and pattern recognition, most time series applications employ some form of transformations. Symbolic Aggregate approXimation (SAX) is one well-known technique to transform time-series based data to discrete alphabet based symbolic data. SAX is often used with K-NN to be the black-box model for classification. This chapter tries to verify the generality of EFG framework by applying it to the time series classification problems in combination with SAX.

7.1 Introduction

Large numbers of datasets and applications can be considered to be some variants of time series. Recognizing action sequences in robotics, understanding body metrics in medical applications, stock market variations in financial applications, and sensor data in various industrial applications are some of the real-world data in time series format. Time series data is different from conventional data in terms of the presence of implicit ordering in the data. Time series data itself can be considered as univariate or multivariate and can be fixed or variable length. In this research, I focus on the univariate fixed length time series classification problems.

Most data mining techniques such as classification on a time series usually involves some form of transformation to the existing real valued time ordered data. The complex structural characteristics of time series data such as the high dimensionality, feature correlation, measurement-induced noises that are associated with real-world time series data, render classic data mining algorithms ineffective and inefficient for these representations.
Symbolic-based transformations such as SAX have been shown to be very effective transformations for a large number of time series applications [171]. SAX, in addition to performing time series to discrete symbol mapping, also performs dimensionality reduction.

In this chapter, answers pertaining to time series are sought. Can EFG in collaboration with SAX be considered an effective technique in time series classification? Can EFG be applied as an effective feature generation technique for time series to capture complex motifs in time series? Can EFG based features give results comparable and yet more of transparent model for time series? Can using EFG with DNA alphabets in conjunction to IUPAC ambiguous symbols improve the pattern recognition in SAX? Can EFG with SAX be comparable to other statistical, feature based and kernel methods? Can EFG be applied to time series with multi-class instead of binary classification? These are the key questions that are being explored. In this chapter, I try to use EFG with SAX and apply it to various real-world applications to see the generality of the EFG framework. I first give a basic introduction to time series and representations, SAX, and the theory, then introduce the datasets, I then showcase how EFG and SAX can be combined together and finally perform experiments on well-known real-world datasets.

7.2 Background

7.2.1 Time Series Classification

Time-series-based data have been analyzed in various data mining tasks such as classification, clustering, indexing, and summarization. In this work, I will focus only on the classification task, but the study is more general and can be applied to other tasks like clustering, etc. The definition of time series classification is given below

Definition: Time Series: A time series \( T = \{t_1, t_2, \ldots t_n\} \) is an ordered set of \( n \) real-valued variables.

Definition: Time Series Classification: A training Dataset \( D \), comprising various classes or labels from the set \( C = \{c_1, c_2, \ldots c_m\} \), time series instance consists of tuple
of timeseries as defined above and a class label from $C$. If there is unseen data from the
same distribution as training dataset the the task of predicting the class label given the
learnt model from the training dataset $D$ is the typical time series classification task.

Most time series classification tasks first need to perform some transformation for an
approximate representation of the real valued ordered data. Many transformations like
Discrete Fourier Transform (DFT) [172], the Discrete Wavelet Transform (DWT) [173],
Piecewise Linear, and Piecewise Constant models (PAA) [174], (APCA) [175,176], Singular
Value Decomposition (SVD), and symbolic-based [171] have been proposed. Each of these
techniques can be considered to be approximating the signal using linear combination of
some basis function. One of the major issues with most of the transformations is the curse
of dimensionality posed by large time series datasets.

7.2.2 Symbolic Aggregate approXimation

SAX is a transformation technique that allows a time series of arbitrary length $n$ to be
reduced to a string of arbitrary length $w$, $(w \ll n)$. The alphabet size is also an arbitrary
integer $a$, where $a \gg 2$. SAX performs this transformation using two distinct steps 1)
transform the data into the Piecewise Aggregate Approximation (PAA) representation,
and 2) symbolizing the PAA representation into a discrete string representation and 3)
Numerosity Reduction of the data.

Piecewise Aggregate Approximation (PAA)

A time series $T$ of length $n$ can be represented in a $w$-dimensional space by a vector $T$. The
$i^{th}$ element of $C$ is calculated by the following equation:

$$
\bar{c}_i = \frac{w}{n} \sum_{j=\frac{w}{n}(i-1)+1}^{\frac{w}{n}i} c_j
$$

First, each time series is normalized to have mean zero and standard deviation of 1.
This time series data $T$ is then divided into $w$ equal sized “frames”. The mean value of the data falling within a frame is calculated and a vector of these values becomes the reduced representation. The representation can be visualized as an approximation of the original time series with a linear combination of box basis functions as shown in Figure 7.1. PAA has been considered to be simplest yet an effective technique as compared to more sophisticated ones like DWT and DFT [174,177].

![Figure 7.1: PAA transformation visualized as a linear combination of the box-basis function, reducing 128 long sequences to 8 dimensions [171.](image)](image)

**Discrete Symbolization**

Most normalized time series have a Gaussian distribution, which results in the discretization step generating equiprobable symbols from a discrete symbol list, which is a desired property [178]. Breakpoints are defined as the points that will produce a equal-sized areas under the Gaussian curve [179]. A symbolic transformation table could be created by defining breakpoints that would result in regions of equal-probability on the Gaussian distribution. These breakpoints (or the $z$-values) may be determined by looking them up in a statistical tables. Once the breakpoints are obtained, the mapping to symbol just happens to be
mapping the ordered symbol list to the values of PAA coefficients as shown in Figure 7.2.

![Figure 7.2: PAA approximations with predetermined breakpoints mapped to symbols. In the example above, with \( n = 128 \), \( w = 8 \) and \( a = 3 \), the time series is mapped to the word baabccbc [171]](image)

**Numerosity Reduction**

Most time series data have a large number of sequences, and one common technique is to consider a sliding windows of length \( n \) (user defined parameter) subjected to SAX. Each subsequence of length \( n \) is normalized with mean zero and unit standard deviation and converted to a SAX string. Thus a set of SAX strings are obtained which correspond to the original time series. It was found that a SAX subsequence \( S_i \) is likely to be very similar to its neighboring subsequences \( S_{i+1} \) and \( S_{i-1} \), especially when the sequence is in a smooth region, as depicted in the figure. Normally the first of the repeating subsequence is considered to avoid artificial over-representation of symbols and the technique is called numerosity reduction and is illustrated in Figure 7.3.

As an example, suppose we have the following sequence of SAX strings with the sliding
Figure 7.3: Timeseries of length 128, showing overlapping signals starting at different positions from p 1 to 8 [171].

window technique: $S = \{aacaacabcabbabbabbbacbaa\}$. With the numerosity reduction option, we would get the following sequence instead: $S_{\text{red}} = \{aac1abc3abb4bac8baa9\}$.

In some datasets and applications, including these subsequences is useful, as they might carry important signals. But in most datasets, it was found that excluding these repeating subsequences gave a more accurate representation and classification accuracy [171].

### 7.3 EFG and SAX as a Framework for Time Series Classification

The overall framework for employing EFG along with SAX for time series is shown in the figure. SAX performs the preprocessing to convert the time series data to a string representation and EFG does the feature generation task while any discriminating classifier like Naïve Bayes can be used to learn models from these features.

Time series to SAX strings needs various user defined parameters like the sliding-window length $n$, PAA frame reduction size $a$, and total number of alphabets $\alpha$ for discretizing. Since the goal was to employ EFG with already widely-experimented DNA alphabets with ambiguity, I chose SAX representation to be standard in all the experiments to have 4
alphabets, i.e., A,C,G,T. The sliding-window size and PAA size were used from standard SAX based runs from previous research and is mentioned along with results in the table.

The EFG algorithm has been customized to use IUPAC symbols for the base motif representation. The motif length is set to default between 2 to 8 nt long. The mutation and crossover parameters are also set to default of 0.1 each and 0.7 respectively. The hall of fame capture of features per generation is set to 250. The EFG algorithm is run for 30 generations.

7.4 Experiments

In the sections below, different experiments are performed to answer the questions posed in the introduction section.

7.4.1 Comparing State-of-the art algorithms

In the first set of experiments, the goal is to compare EFG with IUPAC representation of motifs for feature generation and a standard classifier like Naïve Bayes with the state of the art algorithms in time series classification. The datasets chosen for this task are some standard time series datasets with binary class labels from the real-world time series data. I will give a short description of each dataset below.

Datasets

**GunPoint**  This real-world dataset comes from the video surveillance domain and has two classes, containing 50 training and 150 testing examples with time series length of 150 [180]. The two classes are the gun drwaing and gun pointing, both motions being very close to each other and easily distinguishable to human eye. The task is to predict the classes from the training data of similar nature. The time series for gun drawing consists of actors trying to draw a replica gun from a hip-mounted holster, point it at a target for approximately one second, then return the gun to the holster, and their hands to their sides. The time series for gun pointing consists of actors trying to point with their index fingers to a target.
Figure 7.4: The EFG algorithm with the SAX algorithm
for approximately one second, and then return their hands to their sides. For both classes, centroid of the right hand in both the X- and Y-axes; however, only X-axis movement is used as time series.

**ECG** The electrocardiogram (ECG) dataset contains measurements of cardiac electrical activity as recorded from electrodes at various locations on the body; each instance in the ECG database contains the measurements recorded by one electrode during one heartbeat [181]. The two classes are normal or abnormal conditions analyzed by the domain experts. The training and testing instances each contain 100 samples of time series which are of length 96.

**Coffee** The Coffee dataset comes from food spectrograms domain [182]. Food spectrograms are used in chemometrics to classify food types, a task that is useful in food safety and quality assurance. The task is to automatically differentiate types of coffee given the food spectrograms as time series measurements. The two classes of coffee datasets are readings from Arabica and Robusta coffee variants. The spectograph readings presented as time series are of length 286 and the split of training/testing data is 28 each.

**Lightning2** The Lightning2 dataset comes from the geographic and satellite-based monitoring domain [183]. The time series measurements are the RF signals obtained from satellites and are of length 637. The task is to classify the two distinct signals, the intra-cloud based lightning and the cloud-ground based lightning. The datasets for training and testing are 60 and 61 respectively.

**SonyAIBOSurface** The SonyAIBOSurface dataset comes from the robotics and surface detection domain [184]. The quadruped robot, equipped with multiple sensors, including a tri-axial accelerometer, is used to record walk cycles on the X-axis and is of length 65. The task is to identify the surfaces, i.e., carpet and cement, based on the walk cycles of length 70 at 125 Hertz. The dataset contains 20 training instances of two classes and 601 testing instances.
Comparison Methodology

For each dataset, I performed the training using the training set with parameters of SAX changed for each dataset, but the EFG parameters remaining the same. The SAX parameters like sliding-window length $n$ and the PAA reduction length $a$ are set based on previous studies done in SAX and outlined in the table. The classifier used in all the experiments is the simplest Naïve Bayes classifier. During the testing phase, the same parameters of SAX are applied to the time series for SAX conversion and the trained model is used for predicting. The accuracy measured as the number of correct predictions, which is normally the metric used in all time series, is used as the comparison metric.

Comparison with different distance metrics like Euclidean with $L_2$ and DTW is performed. Comparison is also shown with basic SAX, with 1NN, with same number of alphabets, and the one tuned for increased number of alphabets. Since the research goal is to verify the mapping of EFG with using DNA alphabets with ambiguity, the choice of 4 alphabets is a self imposed restriction. In future work, this can be changed and explored with higher number like in normal SAX.

The implementation used for Euclidean with $L_2$, DTW, and SAX were from Jmotif and FAP. All the experiments are run 30 times and the statistically significant results with 95% confidence are reported in bold in Table 7.1.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Euclidean</th>
<th>DTW</th>
<th>SAX-BEST</th>
<th>SAX</th>
<th>EFG-SAX</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>GunPoint</td>
<td>0.11</td>
<td>0.12</td>
<td>0.18 (32,4,10)</td>
<td>0.20 (32,4,4)</td>
<td><strong>0.032</strong> (32,4,4)</td>
<td>1</td>
</tr>
<tr>
<td>ECG200</td>
<td>0.15</td>
<td>0.22</td>
<td>0.12(32,8,4)</td>
<td>0.12 (32,8,4)</td>
<td>0.13(32,8,4)</td>
<td>2</td>
</tr>
<tr>
<td>Coffee</td>
<td>0.16</td>
<td>0.12</td>
<td>0.46(48,4,3)</td>
<td>0.18(48,4,4)</td>
<td><strong>0.01</strong> (48,4,4)</td>
<td>1</td>
</tr>
<tr>
<td>Lightning2</td>
<td>0.29</td>
<td>0.17</td>
<td>0.313 (128,8,4)</td>
<td>0.313 (128,8,4)</td>
<td>0.22(128,8,4)</td>
<td>2</td>
</tr>
<tr>
<td>SonyAIBOSurface</td>
<td>0.30</td>
<td>0.30</td>
<td>0.38 (10,8,5)</td>
<td>0.38 (10,8,4)</td>
<td>0.32 (10,8,4)</td>
<td>2</td>
</tr>
</tbody>
</table>

To compare EFG with the rest of algorithms, one is in terms of error-rate and other
is to rank its error rate as compared to the rest. In a couple of datasets like GunPoint and Coffee, EFG-SAX gives the best results with an error rate whose mean is 0.032 and 0.01 respectively, the lowest as compared to all the state-of-the-art algorithms as shown in Table 7.1. For ECG200, the EFG-SAX is comparable to most and ranks ahead of DTW and Euclidean-based KNN. For the Lightning2 dataset, EFG-SAX performs comparable to SAX, but the DTW-based classification is significantly better than the rest. Finally, for the SonyAIBOSurface, EFG-SAX performs almost comparable but bit worse than DTW and Euclidean, but better than SAX by itself, given the same parameters. SonyAIBOSurface has only 20 instances of training data, so the curse of dimensionality may have played a role here and the distribution of training/testing data may be very different. To verify the later, 10-fold cross-validation was performed on the training dataset and it yielded accuracy of 0.97, confirming the theory of distribution being slightly different. Also, EFG-SAX overall rank in almost all the dataset is 1.8 amongst 5 methodologies shows the relative strength of EFG-SAX in classifying time series datasets.

7.4.2 Sequence Classification Comparison

If SAX is used for discretizing the time series classification datasets, can EFG be comparable to other techniques known for handling discrete sequence-based data? A comparison of EFG with feature-based, statistical and kernel methods on some subset of datasets with the same preprocessing from time series to symbolic discrete set using SAX will be the next research goal. Two datasets, GunPoint and SonyAiBoSurface from above, which were statistically best and sightly worse respectively were considered for this comparison. I used SVM with WeightedDegreePosition Kernel as the string kernel implementation, K-mer for feature based, and Homogeneous HMM and MSP as statistical discriminative and generative techniques. The kernel parameters like motif length or order were kept same as that of EFG at 8, while other parameters like C for SVM, order of markovs were chosen using cross-validation using a default ranges in the grid search. The methods that have randomness are run 30 times, and the mean error is noted and the significance is calculated.
using paired-t tests with 95% confidence intervals and shown in bold in Table 7.2.

Table 7.2: Error Rate comparing EFG-SAX with feature, statistical, and kernel methods

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Feature</th>
<th>Kernel</th>
<th>Statistical Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datasets</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GunPoint</td>
<td>0.07</td>
<td>0.032</td>
<td>0.05</td>
</tr>
<tr>
<td>SonyAIBOSurface</td>
<td>0.52</td>
<td>0.32</td>
<td>0.52</td>
</tr>
</tbody>
</table>

Interestingly, it can be seen that SAX based discretized symbolic representation works well with EFG as compared to other techniques. SonyAIBOSurface, which showed bit worse performance as compared to other state-of-the-art, shows even worse performance with all other techniques like WD-S kernel, K-mer, HMM and MSP.

7.4.3 Multi-class Time Series Classification

Finally, some of the time series applications are multi-class in nature. EFG in general is a binary-classification-based framework as it tries to find features which are discriminating to one class as compared to the other. There have been many machine learning strategies to adapt the binary-class problems to multi-class problems as one-vs.-one or one-vs.-rest problems, creating many binary classification problems. I adapt the one-vs.-one strategy in the methodology to create many binary classification problems, create binary classification models using EFG features for these and combine the models using simple vote mechanism in an ensemble.

The dataset adopted for experimentation is the cylinder-bell-funnel (CBF) time series data. The time series for CBF is defined from the following equations, where $c(t)$, $b(t)$ and $f(t)$ define cylinder, bell, and funnel respectively.

$$c(t) = (6 + \eta) \times \chi_{[a,b]}(t) + \varepsilon(t) \quad (7.2)$$
\[ b(t) = (6 + \eta) \times \chi[a, b](t) \times (t - a)/(b - a) + \varepsilon(t) \quad (7.3) \]

\[ f(t) = (6 + \eta) \times \chi[a, b](t) \times (b - t)/(b - a) + \varepsilon(t) \quad (7.4) \]

\[ \chi[a, b](t) = \begin{cases} 0, & t < a \\ 1, & a < t < b \\ 0, & t > b \end{cases} \quad (7.5) \]

Figure 7.5 illustrates instances of CBF, showing the cylinder class having a plateau from \( a \) to \( b \), the bell class having a gradual increase from \( a \) to \( b \) and the funnel class having sudden increase at \( a \) and gradual decrease to \( b \). The time series is of length 128 and is considered a model characterizing the properties of temporal domains. Various characteristics of CBF such as random amplitude variation as a result of \( \eta \), random noise as a result of \( \varepsilon \) and large variations at start and end make it really suitable as complex classification problem.

The implementation of EFG-SAX was trained on three models for cylinder-bell, cylinder-funnel, and bell-funnel using EFG-SAX as before with a Naïve Bayes classifier. The voting ensemble using average function for the probability estimate was chosen to combine the outputs for each models and give classification and confidence to the unseen test data.

The comparison of EFG-SAX with different algorithms like DTW, 1NN with Euclidean distance, 1NN with SAX, with the same parameters as EFG, and one with increased alphabets is shown in Table 7.3. EFG-SAX outperforms all the traditional time series classification algorithms in significant way, recording the lowest error rate.
Figure 7.5: Showing Cylinder, Bell and Funnel as three classes from top to bottom as time series.
## Table 7.3: Comparison of EFG-SAX with different algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2</td>
<td>0.14</td>
</tr>
<tr>
<td>DTW</td>
<td>0.05</td>
</tr>
<tr>
<td>SAX-BEST (32,4,10)</td>
<td>0.10</td>
</tr>
<tr>
<td>SAX (32,4,4)</td>
<td>0.11</td>
</tr>
<tr>
<td>EFG-SAX (32,4,4)</td>
<td><strong>0.02</strong></td>
</tr>
</tbody>
</table>

### 7.5 Conclusion

In this chapter, the EFG-SAX framework for time series classification was implemented and studied. It is clear that for binary classification applications, the EFG-SAX framework performs relatively better or close to the best algorithms for time series classification. The choice of using ambiguity symbols and complex motifs generate much more complex patterns that help SAX based discretization for better classification. It was also interesting to note that EFG-SAX outperforms the traditional feature, kernel, and statistical methods with same SAX based discretization confirming the generic nature of EFG to find more complex patterns in sequences. The reason that EFG outperforms normal feature based methods is clearly because of its ability to exploit much complex patterns in the symbolic sequences as shown in Figure 7.6. Finally, applying EFG for multi-class algorithms using ensemble methodology with one vs. one shows relative strength of EFG in finding discriminative features and enhancing the ensemble classification accuracy.
Figure 7.6: Showing complex patterns in a time series, that when translated as symbols can help the EFG capture positional, correlational, and compositional elements as motifs.
Chapter 8: PSBML and Theory

In this chapter, I discuss a meta-learning algorithm (PSBML) that combines features of spatial evolutionary algorithms with concepts from ensemble and boosting methodologies to achieve the desired scalability property. I present both theoretical and empirical analyses confirming the theory that show that PSBML preserves a critical property of boosting, specifically, convergence to a distribution centered around the margin.

8.1 Big Data Learning Problem

Many real-world applications, such as web mining, social-network analysis, and bioinformatics, involve large datasets with millions, or even billions, of data items. Many traditional supervised learning algorithms such as support vector machines (SVM) have training times of $O(n^3)$ and space complexity of $O(n^2)$, where $n$ is the size of the training set [58]. In order to handle the large training sets required by these applications, one of two approaches is typically taken: (1) perform some form of sampling on the dataset to reduce its size, or (2) customize the learning algorithm to improve the running time via parallelization. In the first approach, sampling techniques frequently introduce unintended biases that reduce the accuracy of the results. Similar reductions in accuracy often result from modifications to a learning algorithm to improve its speed. There are very few general procedures that under minimal assumptions can parallelize any given machine learning algorithm keeping the desired balance between speed and accuracy.

In this chapter, I describe an alternative approach that combines features of spatially-structured evolutionary algorithms (SSEAs) with the well-known machine learning techniques of ensemble learning and boosting. The result is a powerful and robust framework
for parallelizing ML methods in a way that does not require changes to the underlying ML methods.

8.2 The PSBML Algorithm

The parallel spatial boosting algorithm (PSBML) has at its core a standard SSEA [93], where the individuals of the population are the training instances, uniformly distributed using the same class-to-instance ratio over a two-dimensional toroidal grid. Each node in the grid is associated with a classifier, and, as in SSEA, interacts only with its neighbors to select the individuals during each learning epoch. Commonly used neighborhood structures are shown in Figure 8.1.

During each epoch of training, a node in the grid performs a standard training process using its local data. For testing, a node’s training data are combined with the instances assigned to the neighboring nodes. In addition to predicting the test instances, each classifier outputs a confidence value for each prediction, which is then used for weighting the corresponding instance. Every node updates its local training data for the successive training epoch by probabilistically selecting instances based on the assigned weights.

Figure 8.1: Two dimensional grid with various neighborhood structures.
The confidence values are used as a measure of how difficult it is to classify a given instance, allowing the local EA in the node to select during each iteration the most difficult instances from all its neighbors. Since each instance is a member of the neighborhood of multiple nodes, an ensemble assessment of difficulty is performed, similar to the boosting of the margin in AdaBoost [21].

In PSBML the confidence $c_{si}$ of an instance $i$ is set equal to the smallest confidence value obtained from any node and for any class:

$$c_{si} = \min_{n \in N_i} c_{ni},$$

where $N_i$ is a set of indices defined over the neighborhoods to which instance $i$ belongs, and $c_{ni}$ is the confidence credited to instance $i$ by the learner corresponding to neighborhood $n$. These confidence values are then normalized through linear re-scaling:

$$c_{si}^{norm} = \frac{c_{si} - c_{s\text{min}}}{c_{s\text{max}} - c_{s\text{min}}},$$

where $c_{s\text{min}}$ and $c_{s\text{max}}$ are the smallest and the largest confidence values obtained across all the nodes, respectively.

A weight $w_i$ assigned to instance $i$ is then set to:

$$w_i = 1 - c_{si}^{norm}$$

to indicate its relative degree of classification difficulty (low confidence).

The $w_i$ are used to define a probability distribution over the set of instances $i$, and used by the local EA’s stochastic sampling technique (roulette wheel selection) to update its local set of training instances. The net effect is that the smaller the confidence credited to an instance $i$ is (i.e., the harder it it to learn instance $i$), the larger the probability will be for instance $i$ to be selected.
With most EA applications, more effective results are obtained by incrementally updating only a fraction of the current population instead of replacing the entire population at each generation. For PSBML a replacement probability \( p_r \) is defined. Its optimal value was estimated experimentally, and set to 0.2 \([23]\), and thus each local EA updates about 20% of its population of training examples each generation.

The pseudo-code of PSBML is given in Algorithm 8.3.1. The parameters for grid configuration, such as width and height, replacement probability, and maximum iterations (or epochs), are all included in \( \text{GridParam} \).

### 8.3 Theoretical Analysis of PSBML

I use Gaussian Mixture Models (GMMs) combined with the mean-shift algorithm to model the behavior of the PSBML algorithm. Specifically, I formally show that PSBML, through the roulette wheel selection process, iteratively changes the data distribution, and converges to a distribution whose modes are centered around the margin, i.e., around the hardest points to classify.

Each node of the grid in the PSBML algorithm, along with its neighborhood structure, represents a sample of the whole dataset, where each point is weighted according to how difficult it is to be classified. The present analysis fits a Gaussian mixture model on the weighted points, and applies the mean-shift procedure to locate the modes of the resulting distribution. Observe that, throughout the iterations of PSBML, as more data closer to the boundary are selected, the data distribution grows higher modes centered around the margin, as shown in Figure 8.2. These modes will be the ones visited by the mean-shift procedure, irrespective of the starting point.
Algorithm 8.3.1: PSBML(Train, Validation, GridParam)

\textbf{initializeGrid}(Train, GridParam)
\textbf{comment}: distribute the instances over the nodes in grid
\text{currentMin} \leftarrow 100
\text{Pr} \leftarrow \text{GridParam.pr}
\textbf{comment}: Probability of replacement
\textbf{for} i \leftarrow 0 \textbf{to} GridParam.iter
\textbf{do}
\text{marginData} \leftarrow \text{Train}
\textbf{comment}: marginData initialized to all training data
\begin{algorithmic}
\STATE \text{TRAINNODES}(GridParam)
\textbf{comment}: Train all nodes
\STATE \text{TESTANDWEIGHTNODES}(GridParam);
\textbf{comment}: Test using neighborhood and assign weight
\STATE \text{PrunedData} \leftarrow \{\}
\textbf{for} j \leftarrow 0 \textbf{to} GridParam.nodes
\textbf{do}
\begin{algorithmic}
\STATE \text{NeighborData} \leftarrow \text{COLLECTNEIGHBORDATA}(j);
\STATE \text{NodeData} \leftarrow \text{NodeData} \cup \text{NeighborData}
\STATE \text{ReplaceData} \leftarrow \text{ROULETTEWHEELSEL}(\text{NodeData, Pr})
\STATE \text{PrunedData} \leftarrow \text{UNIQUE} (\text{PrunedData, ReplaceData})
\textbf{comment}: Unique keeps 1 copy of instances in set
\end{algorithmic}
\STATE \text{error} \leftarrow \text{TESTCLASSIFIER}(\text{PrunedData, Validation})
\textbf{comment}: Use Validation set to track model learning
\textbf{if} \text{error} < \text{currentMin}
\textbf{then}
\begin{algorithmic}
\STATE \text{currentMin} \leftarrow \text{error}
\STATE \text{bestClassifier} \leftarrow \text{SAVECLASSIFIER}(\text{PrunedData})
\STATE \text{marginData} \leftarrow \text{PrunedData}
\textbf{comment}: marginData set reduced
\end{algorithmic}
\STATE \text{return} (\text{bestClassifier, marginData})
\end{algorithmic}
\end{algorithmic}

Since each node in the toroidal grid has the same behavior, they all fit a Gaussian mixture model on the respective neighborhood. By consolidating the micro-behavior of the mean-shift procedure at each node, an overall convergence to a distribution with peaks centered around the boundary is expected. The analysis below, and the empirical results presented in Section 8.4, confirm this argument.
8.3.1 Distribution of a node at time $t = 1$

After the completion of the first iteration of PSBML, each classifier in the grid has been trained with its own data, and is tested on the instances of the neighbors, to which it assigns confidence values. A common approach to assess the confidence of a prediction for an instance is to measure its distance from the estimated decision boundary: the smaller the distance, the smaller the confidence will be. The resulting weight values drive the probability for a point to be selected for the successive iterations. Below we modify the Gaussian mixture model to incorporate this process.

Consider a Gaussian mixture density of $M$ components:

$$p(x) = \sum_{m=1}^{M} p(m)p(x|m) \quad \forall x \in \mathbb{R}^D \tag{8.1}$$
where the \(p(m)\) are the mixture proportions such that \(p(m) > 0, \forall m = 1, \ldots, M,\) and \(\sum_{m=1}^{M} p(m) = 1\). Each mixture component is a Gaussian distribution in \(\mathbb{R}^D\), i.e. \(x|m \sim \mathcal{N}_D(\mu_m, \Sigma_m)\), where \(\mu_m = \mathbb{E}_{p(x|m)}[x]\) and \(\Sigma_m = \mathbb{E}_{p(x|m)}[(x - \mu_m)(x - \mu_m)^T]\) are the mean and covariance matrix of the Gaussian component \(m\).

Let us first consider a known result for the mean-shift procedure applied to Gaussian mixture models to find the modes of the distribution [185]. No closed-form solution exists to this problem, so numerical iterative approaches have been developed. In particular, the fixed-point iterative method gives the following fixed-point solution [185]:

\[
x^{(t+1)} = f(x^{(t)})
\]

(8.2)

where

\[
x = f(x) = (\sum_{m=1}^{M} p(m|x)\Sigma_m^{-1})^{-1} \sum_{m=1}^{M} p(m|x)\Sigma_m^{-1} \mu_m
\]

(8.3)

Let us assume now that we model the sample data assigned to a node and to its neighbors using a Gaussian mixture distribution of \(M\) components in \(\mathbb{R}^D\). In our analysis, we consider only the distribution of one class; the argument stays the same for the other class due to the symmetry with respect to the boundary. We need to embed the weighted sampling process performed by PSBML in our Gaussian mixture modeling. Let us assume the optimal boundary between classes is known. Let \(s \in \mathbb{R}^D\) be a point on the boundary. We estimate the distance of a point \(x\) from the boundary by considering its distance from \(s\). At each iteration of the PSBML algorithm, the weights bias the sampling towards those points which are closer to the boundary: the larger the weight of a point is, the larger is the probability of being selected. To embed this mechanism in the Gaussian mixture modeling, we set \(p'(x|m) = w(x) \cdot p(x|m)\), where \(w(x)\) is a Gaussian weighting function centered at \(s\):

\[
w(x) = (2\pi)^{-D/2}|\Sigma_s|^{-1/2}e^{-1/2(x-s)^T\Sigma_s^{-1}(x-s)}
\]

(8.4)
and

\[ p(x|m) = (2\pi)^{-D/2} |\Sigma|^{-1/2} e^{-1/2(x-\mu_m)^T \Sigma_m^{-1}(x-\mu_m)} \]  

(8.5)

We compute the gradient of \( p'(x|m) \) with respect to the independent variable \( x \), while keeping the parameters \( \mu_m \) and \( \Sigma_m \) fixed:

\[
\frac{\partial p'(x|m)}{\partial x} = \frac{\partial p(x|m)}{\partial x} + \frac{\partial w(x)}{\partial x} \frac{p(x|m)}{\partial x} \]

(8.6)

Considering each derivatives:

\[
\frac{\partial p(x|m)}{\partial x} = p(x|m) \Sigma_m^{-1}(\mu_m - x) \]

(8.7)

\[
\frac{\partial w(x)}{\partial x} = w(x) \Sigma_s^{-1}(s - x) \]

(8.8)

Substituting these results in equation (8.6), we obtain:

\[
\frac{\partial p'(x|m)}{\partial x} = w(x)p(x|m) \Sigma_m^{-1}(\mu_m - x) + p(x|m)w(x) \Sigma_s^{-1}(s - x) \]

(8.9)

We now turn to the mixture of \( M \) Gaussian distributions defined in (8.1). By the linearity property of the differential operator, we obtain:

\[
\frac{\partial p(x)}{\partial x} = w(x) \sum_{m=1}^{M} p(m)p(x|m) \Sigma_m^{-1}(\mu_m - x) + (s - x)w(x) \sum_{m=1}^{M} p(m)p(x|m) \Sigma_s^{-1} \]

(8.10)

By setting the above gradient to 0 and simplifying \( w(x) \), we derive a fixed point iteration...
procedure that finds the modes of the distribution [185]:

\[
\sum_{m=1}^{M} p(m)p(x|m)\Sigma_m^{-1}(\mu_m - x) = (x - s) \sum_{m=1}^{M} p(m)p(x|m)\Sigma_s^{-1}
\]  

(8.11)

Solving for \( x \), we obtain:

\[
x = \frac{\sum_{m=1}^{M} p(m)p(x|m)\Sigma_s^{-1}s + \sum_{m=1}^{M} p(m)p(x|m)\Sigma_m^{-1}\mu_m}{\sum_{m=1}^{M} p(m)p(x|m)\Sigma_s^{-1} + \sum_{m=1}^{M} p(m)p(x|m)\Sigma_m^{-1}}
\]

(8.12)

Using Bayes rule and simplifying \( p(x) \):

\[
x = \frac{\sum_{m=1}^{M} p(m|x)\Sigma_s^{-1}s + \sum_{m=1}^{M} p(m|x)\Sigma_m^{-1}\mu_m}{\sum_{m=1}^{M} p(m|x)\Sigma_s^{-1} + \sum_{m=1}^{M} p(m|x)\Sigma_m^{-1}}
\]

(8.13)

Rearranging the terms, we obtain our fixed-point solution:

\[
x = \left( \sum_{m=1}^{M} p(m|x)\Sigma_s^{-1} + \sum_{m=1}^{M} p(m|x)\Sigma_m^{-1} \right)^{-1} \times 
\]

\[
\left( \sum_{m=1}^{M} p(m|x)\Sigma_s^{-1}s + \sum_{m=1}^{M} p(m|x)\Sigma_m^{-1}\mu_m \right)
\]

(8.13)

Comparing equations (8.3) and (8.13) we can see that, by weighting the points according to their distance from the boundary, the modes of the resulting distribution become the weighted average of the means \( \mu_m \) and \( s \). That is, each local classifier in PSBML, by
assigning weights to points according to the confidence of the prediction, causes the shifting of the modes towards the points closest to the estimated boundary, i.e. towards its margin.

8.3.2 Distribution of the Grid at time $t = 1$

The whole grid itself is modeled as a Gaussian mixture (given by the collection of GMMs at each node). Thus, the same derivation given above, applied to the grid, shows that the overall data distribution will have the same modes emerging from the individual nodes, i.e. centered around the margin of the boundary.

8.3.3 Final Distribution of the Grid

After a number of iterations, at each node, data will be sampled according to the current distribution. We can show that all the nodes will converge to the same mode. Suppose that a node $i$, at time $t$, has a neighborhood with means $T(t) = \{\mu_1^{(t)}, \ldots, \mu_l^{(t)}\}$, and one of these means, say $\mu_g^{(t)}$, is the closest (globally) to the boundary. During successive iterations, the sampling process causes the elimination of modes that are far from the boundary. Thus, after $k > 0$ steps, the local distribution of node $i$ will have a smaller number of modes: $T(t + k) = \{\mu_1^{(t+k)}, \ldots, \mu_{l-m}^{(t+k)}\}$, with $l - m > 0$. Due to the weighted sampling mechanism (note that the sample size remains constant at each iteration), $\mu_g^{(t)} = \mu_g^{(t+k)} \in T(t + k)$. The whole process converges when $T(t + 1) = T(t)$, or the mean shift is negligible, and at convergence $T(t) = \{\mu_g^{(t)}\}$. We observe that spatially structured replication-based evolutionary algorithms show a similar behavior, where the global best is spread deterministically across the nodes, until all the nodes in the grid converge to the same individual according to logistic takeover curves [186].
8.4 Empirical Analysis of PSBML and GMMs with Mean-shift

I performed a number of experiments to verify the established relationship between PSBML and GMMs with mean-shift and generated synthetic data on which to run the following experiments.

1. I ran the PSBML algorithm using a $5 \times 5$ spatial grid with the C9 neighborhood (see Fig. 8.1) and a large margin classifier and observed the population distribution change over training epochs.

2. I replaced each local classifier with a GMM with mean-shift, while keeping the grid structure and neighborhood interaction unchanged. Each data instance is weighted \textit{a priori} using the Gaussian weighting function as defined in the theoretical analysis. I ran GMM with mean-shift on each node and performed sampling iteratively at every training epoch exactly as in PSBML. I observed the population distribution change over time.

3. I removed the grid structure and ran GMMs with mean-shift estimation on the whole dataset, with each instance weighted according to its distance from the known boundary as above. I observed the data distribution and final modes at convergence, and compared them with those obtained in the previous setting.

8.4.1 A Non-linearly Separable Dataset

Instances were drawn at random within a square centered at the origin and with side of length two. Points with a distance smaller than 0.4 from the origin are labeled as negative, and those with a distance greater or equal than 0.4 are labeled as positive (see Figure 8.6). The three experiments were run as described in Section 8.4. For experiment 1, the large margin classifier used at each node fits a circle to its training set by setting its radius to the average distance of the origin from the smallest positive and the largest negative instances.
For testing, the learner outputs “−” when the instance falls within the circle, and “+” otherwise. The confidence of the prediction is the distance of the instance from the circular boundary.

To compare the data distributions obtained in experiments 1 and 2, I recorded the number of points at various intervals of distances from the origin at training epochs 25 and 50. The resulting histograms are given in Figure 8.3. The two methodologies, PSBML and GMMs with mean-shift, provide a nearly identical distribution at both generations, and they converge to a distribution with modes centered on the points closest to the boundary.

For experiment 3, I ran GMMs with mean-shift estimation 30 times on the whole weighted data. The means of the modes at convergence were (−0.01, 0.38) and (0.01, −0.41), with a very small standard deviation of 0.03. The distribution at convergence was very close to those obtained in experiments 1 and 2. Interestingly, I observed that, when the weights were removed, the modes at convergence moved to (−0.03, 0.51) and (0.03, −0.49).

**Weight Distribution Changes.** One important property of boosting is to scale the
weights of data as a function of its distance from the margin. SSEAs have similar behavior where the *takeover* curves exhibit a logistic function with time [186]. To observe the effect of weight changes, in Figure 8.4, I plotted the weights of all points at different radii and for different generations for the circle dataset. The exponential decay and the logistic increase based on the vicinity to the margin of the data can be clearly seen. For positive points, when the radius is between 0.3 and 0.4, and for negative points, when the radius is between 0.4 and 0.5, an increase is seen with time, and for the rest there is an exponential decay, confirming a behavior analogous to boosting.

### 8.4.2 Linearly Separable Bivariate Gaussians

I created a synthetic dataset consisting of 5 Gaussians for each class, with roughly the same density but different shapes (see Figure 8.6). The Gaussians with means (14, 8) and (24, 8) are the closest to the boundary, given by the line $x = 20$. They simulate the “global modes”. I again ran the three experiments described in Section 8.4. The large margin classifier was
Figure 8.5: Linearly separable Gaussian dataset: Data distribution at epochs 25 (Left) and 50 (Right) using PSBML and GMMs.

Figure 8.6: (Left) Circle dataset; (Right) Bivariate Gaussian dataset.
simulated by estimating the average distance between the smallest positive and the largest negative instances.

Again I observed that the data distributions produced by PSBML and GMM with mean-shift and grid structure are very much alike, as illustrated in Figure 8.5. For experiment 3, with 30 runs on the weighted dataset, the means of the modes converged to (14.02, 7.89) and (24.09, 7.88), with deviation of 0.002, matching exactly the results for experiments 1 and 2.

### 8.4.3 Hard Instances and Support Vectors

I also analyzed the data distribution at convergence by comparing the hard instances identified by PSBML with the support vectors of a trained SVM. Table 8.1 shows the percentage of overlap for the two simulated datasets. The support vectors of the trained SVMs with the highest $\alpha$ (i.e., weight) values correspond to the hard instances with the top 10% largest weights identified by the PSBML algorithm for both the datasets.

<table>
<thead>
<tr>
<th></th>
<th>2D Circle</th>
<th>2D Gaussians</th>
</tr>
</thead>
<tbody>
<tr>
<td>SV overlap</td>
<td>90%</td>
<td>94%</td>
</tr>
</tbody>
</table>

### 8.5 Conclusion

In this chapter, a novel approach for parallelizing machine learning methods that combines the features of spatially-structured evolutionary algorithms with the well-known machine learning techniques of ensemble learning and boosting was introduced. The key advantage of PSBML is that it does the scaling in a way that does not require changes to the underlying
machine learning algorithm. The theoretical analysis was obtained by creating a model for a stochastic parallel supervised learning algorithm in terms of a well-known statistical distribution model. The empirical analysis confirmed the veracity of the theoretical model.
Chapter 9: PSBML Validation

This chapter tries to answer various questions related to PBML in empirical sense. Is PSBML a competitive meta-learning framework? Is PSBML scalable in time and space? Is the PSBML accuracy sensitive to parameters? Is PSBML robust to noise as compared to other boosting algorithms?

9.1 Empirical Validation

9.1.1 Software, Hardware and Methodology

All the scalability experiments (in which running times were measured) were run on a dual, 3.33 GHz, 6 core Intel Xeon 5670 processor. PSBML was implemented both as a single threaded Weka [187] classifier and as a multi-threaded standalone Java implementation that can run on any JVM version above 1.5. All experiments with PSBML were run using a maximum heap size of 8GB and a number of threads equal to the number of nodes in the grid. All SVMs and boosting implementations, where running times were compared, used either the native Matlab or C++ code, except for AdaBoostM1, where Weka 3.7.1 was used. All statistical significance tests were performed using the Matlab paired-t test function.

9.1.2 Parameter and Sensitivity Analysis

To study the effects that SSEA neighborhood structure has on the performance of PSBML, the UCI Chess (King-Rook vs. King-Pawn) dataset is chosen for the experiments. It has 3196 instances, 36 attributes and 2 classes. PSBML was run on this problem using various neighborhood structures as shown in Fig. 9.1(a). A 5×5 grid with a Naïve Bayesian classifier
as the ML method with discretization for numeric features was used. The ensemble classifier is evaluated by combining the reduced datasets from all the nodes, training a single classifier with these and comparing the test set predictions for classification accuracy or the error-rate. Although the average reduction in the training data was quite similar for all the neighborhoods, their classic “over fitting curves” were different. The stronger selection pressures of L9 and C13 produced the more rapid initial decrease in test classification error rates, which subsequently increased more rapidly as the training data became too sparse. The simplest L5 neighborhood reduced classification error rates too slowly. The best results were obtained with C9.

In the next set of experiments, PSBML was run on the UCI Chess dataset with different \( p_r \) values to observe the effect that different rates of replacement have on the performance of PSBML. Figure 9.1(b)–(c) illustrates that increasing \( p_r \) results in faster convergence but a less accurate learner, with the best results obtained when \( p_r \) is about 0.2.

Finally, to see the impact of grid size on accuracy, Chess and Magik datasets, both with different training data sizes, were used for experiments. Keeping the neighborhood configuration fixed at previously recorded best value at C9 and replacement rate at 0.2, various grid sizes ranging from 3 \( \times \) 3 to 7 \( \times \) 7 in steps are used, and auROC for PSBML is recorded for 30 runs. The NaïveBayes classifier is used in all the PSBML nodes and is same for all the configurations. The Table 9.1 summarizes the results, showing that there is no statistically significant difference in the measured metrics of auROC across various grid sizes.

These results are not surprising, given the wraparound nature of the grid and “diffusion” of hard instances by fitness proportional selection, only the rate at which the convergence to margin data happens changes with grid size. In the Chess dataset, which has only 3196, as number of nodes increases there is slight degradation in performance. As number of nodes increases, the training data reduces significantly and as a result a classifiers VC bound comes into the effect. Thus, for smaller datasets, the choice of grid configuration may be dependent on this lower bound. With slightly larger dataset like Magik, there is no
degradation even without considering statistical significance. This is an important insight for the practitioner, as scaling based on the number of cores available in the hardware can be used to decide on number of nodes or grid configuration.
Table 9.1: Performance measured as auROC with PSBML on different grid sizes given in the respective columns.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>3 × 3</th>
<th>4 × 4</th>
<th>5 × 5</th>
<th>6 × 6</th>
<th>7 × 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>98.5</td>
<td>98.5</td>
<td>98.3</td>
<td>98.2</td>
<td>98.1</td>
</tr>
<tr>
<td>Magik</td>
<td>89.4</td>
<td>89.5</td>
<td>89.4</td>
<td>89.4</td>
<td>89.5</td>
</tr>
</tbody>
</table>

9.1.3 Meta-learning Experiments

The goal of this experiment is two-fold: first to validate that PSBML provides a general framework for meta-learning, and therefore can be used in combination with a variety of learners; second, to verify that it’s an effective parallel algorithm, i.e., it performs at least as well as the sequential counterpart. To illustrate this, we performed experiments using three base classifiers: Naïve Bayes, Decision Trees (C4.5), and Linear SVMs (LibLinear v1.8) (the corresponding Weka implementations were used). We used five medium to large UCI datasets [188], commonly used for performance comparisons. Table 9.2 provides a description of the data. For each dataset, we normalized the features in the range [0,1], and converted multi-class problems to binary, using the one-vs-all strategy optimized for the LibSVM system, as described in [148]. The PSBML algorithm was run with the C9 neighborhood, a 3 × 3 grid, a replacement probability of 0.2, 20 training epochs, and a validation set size of 10%. We first optimized the base classifiers for performance, and then used the optimized settings in PSBML. Naïve Bayes was used with the option of kernel estimation instead of using the default normal estimation; C4.5 was used with the default settings; and LibLinear was used with L2 loss function in both experiments. Each run, with the exception of Cover and C4.5, was repeated 30 times, and paired-t tests were used
for statistical significance computation using the Area Under the Curve (AUC) [189] as the metric. The experiments involving Cover and C4.5 were run only 10 times, due to the long processing time. Thus, significance is not recorded in this case. Results are reported in Table 9.3. All statistically significant results are marked in bold-face.

It can be observed that PSBML, combined with the Na"{i}ve Bayes classifier, performs statistically significantly better than the Na"{i}ve Bayes classifier itself on all the datasets. Similar results were observed, and theoretical insights were provided, with regular boosting and Na"{i}ve Bayes [190]. Another important result to note is that the ensemble effect of PSBML makes the accuracy of a linear SVM significantly better (in three cases), while parallelizing the LibLinear SVM, which was already optimized for speed.

Table 9.2: UCI datasets training size, testing size and dimensionality used in the experiments.

<table>
<thead>
<tr>
<th></th>
<th>Adult</th>
<th>W8A</th>
<th>ICJNN1</th>
<th>Cod</th>
<th>Cover</th>
</tr>
</thead>
<tbody>
<tr>
<td># Train</td>
<td>32560</td>
<td>49749</td>
<td>49990</td>
<td>331617</td>
<td>581012</td>
</tr>
<tr>
<td># Test</td>
<td>16279</td>
<td>14951</td>
<td>91701</td>
<td>59535</td>
<td>58102</td>
</tr>
<tr>
<td># Features</td>
<td>123</td>
<td>300</td>
<td>22</td>
<td>8</td>
<td>54</td>
</tr>
</tbody>
</table>

9.1.4 Scalability Experiments

The goal of this experiment is to validate whether PSBML performs competitively against custom optimized learning algorithms, in terms of training time, as a measure of speed, and in terms of accuracy, as a measure of performance. PSBML shares an important feature with SVMs: it reduces the training data to the points which are close to the boundary. Thus, PSBML is compared with a number of SVM implementations: a fast Newton method-based LP-SVM [60], a structural optimization-based SVM-PERF [63] (linear because with
Table 9.3: Meta-learning results (AUC) comparing the base classifiers and PSBML combined with the same.

<table>
<thead>
<tr>
<th></th>
<th>Adult</th>
<th>W8A</th>
<th>ICJNN1</th>
<th>Cod</th>
<th>Cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>NB</td>
<td>90.1</td>
<td>94.30</td>
<td>81.60</td>
<td>87.20</td>
<td>84.90</td>
</tr>
<tr>
<td>PSBML</td>
<td><strong>90.69</strong></td>
<td><strong>96.10</strong></td>
<td><strong>81.79</strong></td>
<td><strong>91.79</strong></td>
<td><strong>87.31</strong></td>
</tr>
<tr>
<td>C4.5</td>
<td>88.01</td>
<td>87.80</td>
<td>94.60</td>
<td>95.90</td>
<td>99.50</td>
</tr>
<tr>
<td>PSBML</td>
<td>88.78</td>
<td>84.80</td>
<td>97.30</td>
<td>97.24</td>
<td>97.44</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>54.60</td>
<td>80.20</td>
<td>64.60</td>
<td>88.80</td>
<td>72.20</td>
</tr>
<tr>
<td>PSBML</td>
<td><strong>60.01</strong></td>
<td>80.70</td>
<td>64.80</td>
<td><strong>95.10</strong></td>
<td><strong>79.10</strong></td>
</tr>
</tbody>
</table>

an RBF kernel it crashed), most commonly used LibSVM [148], fast optimized LibLinear [59], stochastic gradient approximation-based SGDT [61], and fast ball enclosure-based BVM [62]. PSBML is also compared against a parallel AdaBoost algorithm [191] and the standard AdaBoostM1. All of the above mentioned implementations of SVMs incorporate some form of custom changes to boost the speed, like incrementally sampling the dataset, or simplifying the quadratic optimization, or assuming linearly separable data. In the following, first the results with synthetic datasets, and then with real ones are presented.

**Synthetic Datasets**

The first dataset was a two dimensional decision boundary based on a sine wave generated by the function $f(x) = 2\sin(2\pi x_1)$ (see Figure 9.2). The dimension $x_1$ was sampled from $[0, 6.28]$ and the $y = f(x)$ dimension was randomly sampled from $[0, 2]$. The second dataset is a $4 \times 4$ rotated checkerboard data with alternate positive and negative classes as shown in Figure 9.2. Each dataset has one million instances, and all the experiments were repeated 30 times. Training time for each of the runs was measured, and the average training time is reported. 10 fold cross-validation was performed for accuracy and the average accuracy is reported. Each algorithm was tuned to some level of optimality for comparisons, i.e., the
soft margin parameter and the radius of the RBF kernel for SVMs were optimized using a grid search in the intervals [-5,15] and [3,-15], respectively.

The PSBML algorithm was run with the C9 neighborhood, a 3 × 3 grid, replacement probability of 0.2, 10 training epochs, and a validation set size of 10% for each training fold. The C4.5 classifier with default parameters was used as it had an intermediate training speed between the fast LibLinear and the kernel estimated Naïve Bayes. Results are shown in Table 9.4. For both the synthetic datasets, PSBML gives the most accurate results with comparable training speed. The synthetic datasets, being highly non-linear, exaggerate the trade-offs implemented by the algorithms.

![Figure 9.2: Synthetic datasets: (Left) Sine wave; (Right) Checkerboard.](image)

**Real-world Dataset**

The KDD Cup 1999 intrusion detection dataset was used to compare the performance of the algorithms. The dataset contains 4,898,431 training instances. The problem was converted into a binary classification problem because many SVM implementations do not support multi-class labels. The feature set was also scaled within the range [0,1], which improved
Table 9.4: Training speed (in seconds) and accuracy for the Checkerboard and the Sine Wave datasets.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Checkerboard</th>
<th></th>
<th>Sine Wave</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Speed</td>
<td>Acc</td>
<td>Speed</td>
<td>Acc</td>
</tr>
<tr>
<td>SVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LP-SVM (Linear)</td>
<td>44.20</td>
<td>50.23</td>
<td>33.20</td>
<td>68.80</td>
</tr>
<tr>
<td>LP-SVM (RBF)</td>
<td>33.20</td>
<td>57.11</td>
<td>105.56</td>
<td>70.11</td>
</tr>
<tr>
<td>LibLinear</td>
<td>133.20</td>
<td>50.08</td>
<td>203.12</td>
<td>68.60</td>
</tr>
<tr>
<td>SVM-PERF (Linear)</td>
<td>1.10</td>
<td>51.01</td>
<td>2.01</td>
<td>61.90</td>
</tr>
<tr>
<td>BVM (RBF)</td>
<td>1.80</td>
<td>50.03</td>
<td>1.20</td>
<td>49.03</td>
</tr>
<tr>
<td>LibSVM (RBF, 0.1% data)</td>
<td>136.20</td>
<td>98.20</td>
<td>423.23</td>
<td>70.80</td>
</tr>
<tr>
<td>Boosting</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdaBoostM1</td>
<td>38.21</td>
<td>51.25</td>
<td>30.71</td>
<td>74.25</td>
</tr>
<tr>
<td>ParallelAdalBoost</td>
<td>17.90</td>
<td>51.22</td>
<td>13.90</td>
<td>78.30</td>
</tr>
<tr>
<td>(9 threads,10 iterations)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSBML</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSBML (C4.5)</td>
<td>123.10</td>
<td>99.49</td>
<td>193.10</td>
<td>99.56</td>
</tr>
</tbody>
</table>

the performance of many SVMs almost 10 times. The PSBML algorithm was run with the C9 neighborhood, a $3 \times 3$ grid, replacement probability of 0.2, 10 training epochs, and a validation size of 0.1% of the training data. The C4.5 classifier was used with default parameters again for the same reasons mentioned earlier.

In previous work, it was noted that many algorithms have a very similar error rate on this dataset. Hence, the number of mis-classifications was suggested and used as comparison metric [192]. The same metric is used in the comparisons here. In addition, since the dataset is unbalanced, areas under the ROC and under the Precision Recall Curve (PRC) are measured. Each of the experiments was run 30 times, except AdaBoostM1 (only 10 times, due to large training time). The mean training times and the mean mis-classification averages are reported in Table 9.6. Some of the algorithms, e.g., LP-SVM, couldn’t run with
a 12GB RAM machine, because the loading of the data matrix itself failed. Also, for SGDT and BVM the output probabilities to measure ROC and PRC couldn’t be computed due to the kernel choice. It is observed that most algorithms that were optimized for speed had to compensate for classification rate. Also, the training time of LibSVM increased considerably when the sampled data went from 1% to 10%, with a small change in classification rate. The ROC value for PSBML was statistically significantly better; the value of the PRC area was comparable to that of SVM-PERF. In conclusion PSBML, while working on the entire dataset, finds a good classification rate at a considerable performance speed.

To see the impact of data sizes on the PSBML algorithm, the training data was sampled in various sizes from 50K, 100K, 500K, to one million. 10 runs were performed with standard PSBML with decision trees, a $3 \times 3$ grid, and the C9 neighborhood. 9 threads were used in this experiment. Training time (log scale) is plotted against data size in Figure 9.3. The graph clearly shows that time scales almost linearly with data size.

To see the impact of the multi-core processor described above on scalability, the number of threads were changed and computed the corresponding average training times. The result is given in Figure 9.4, which shows again that time scales almost linearly with the number of threads.

Another important aspect of a large scale learning algorithm is the memory space requirements. To see the impact of space, the training data was sampled in various sizes from 10K, 100K, 500K, and one million. 10 runs were performed with standard PSBML with decision trees, a $3 \times 3$ grid, and the C9 neighborhood. 9 threads were used in this experiment. The Table 9.5 shows comparison of training data when in memory and average peak memory used by the algorithm with standard deviations for varying data sizes. Figure 9.5 illustrates a linear model fit for the data size against the peak memory usage, showing a clear linearity with training data size. PSBML memory space complexity of $O(n)$ as compared to $O(n^2)$, where $n$ is the size of the training set [58] gives it an important key advantage.
Table 9.5: Memory with various data sizes, peak training memory, and standard deviation for the KDD Cup 1999 dataset.

<table>
<thead>
<tr>
<th>Training Data</th>
<th>Full Data (KB)</th>
<th>Mean Peak Training Memory (KB)</th>
<th>Std. Dev (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10K</td>
<td>187,826</td>
<td>189,423</td>
<td>112</td>
</tr>
<tr>
<td>100K</td>
<td>729,620</td>
<td>742,684</td>
<td>223</td>
</tr>
<tr>
<td>500K</td>
<td>1,400,976</td>
<td>2,119,252</td>
<td>657</td>
</tr>
<tr>
<td>1 Million</td>
<td>1,725,726</td>
<td>2,995,112</td>
<td>2015</td>
</tr>
<tr>
<td>4.5 Million</td>
<td>3,646,792</td>
<td>5,618,084</td>
<td>7831</td>
</tr>
</tbody>
</table>

Table 9.6: Training speed in seconds, mis-classification, area under ROC, and PRC for the KDD Cup 1999 dataset.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speed</th>
<th>MisClass</th>
<th>ROC</th>
<th>PRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LibLinear</td>
<td>80.20</td>
<td>25474.3</td>
<td>94.4</td>
<td>6.3</td>
</tr>
<tr>
<td>LibSVM (RBF, 1% data)</td>
<td>90.20</td>
<td>25517.8</td>
<td>94.1</td>
<td>76.9</td>
</tr>
<tr>
<td>LibSVM (RBF, 10% data)</td>
<td>1495.20</td>
<td>25366.1</td>
<td>94.1</td>
<td>13.1</td>
</tr>
<tr>
<td>SGDT (10 iterations)</td>
<td>211.10</td>
<td>121301</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SVM-PERF (Linear)</td>
<td>4.90</td>
<td>25877.1</td>
<td>93.1</td>
<td>90.3</td>
</tr>
<tr>
<td>BVM (RBF)</td>
<td>3.20</td>
<td>25451.3</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boosting</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaBoostM1</td>
<td>13296.42</td>
<td>190103.3</td>
<td>88.4</td>
<td>17.2</td>
</tr>
<tr>
<td>ParallelAdaBoost</td>
<td>202.30</td>
<td>26170.2</td>
<td>36.2</td>
<td>70.2</td>
</tr>
<tr>
<td>(9 threads, 10 iterations)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PSBML</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PSBML(C4.5)</td>
<td>2913.10</td>
<td>20898.8</td>
<td>95.6</td>
<td>91.2</td>
</tr>
</tbody>
</table>

9.1.5 Comparison Against AdaBoost and Impact of Noise

Here PSBML is compared against AdaBoost testing the robustness in the presence of noise. Previous work found that boosting is more susceptible to noise as compared to other ensemble methods like bagging and stacking [193,194]. Class label noise is added by randomly
changing different percentages of labels in the training data. We used AdaBoostM1 both with decision stumps and with Naïve Bayes (optimized using kernel estimators), and compared it against PSBML combined with the same underlying Naïve Bayes classifier. PSBML was used with the default C9 neighborhood, replacement probability of 0.2, and validation set of 10%.

The datasets used for the meta-learning experiments are used here in the noise experiments and the same preprocessing was applied. To compare the three algorithms, 30 runs were performed without noise and in presence of 10% and 20% of noise. The results are shown in Table 9.7. Statistically significant results using paired-t testing with 95% confidence are highlighted in bold-face.

In absence of noise, PSBML with Naïve Bayes performs significantly better than AdaBoostM1 with decision stumps or with the same optimized Naïve Bayes in three of the five
datasets. To measure how robust a method is across all the datasets being considered, the following quantity, which we call *impact* was computed:

\[
\text{impact} = \frac{1}{N} \sum_{i=1}^{N} (\text{auc}^i_{\text{no-noise}} - \text{auc}^i_{\text{noise}}),
\]  

(9.1)

where \(N\) is the number of datasets. The smaller the value of the impact is for an algorithm, the more robust that method is on average.

The impact values of AdaBoostM1 (DecisionStump), AdaBoostM1 (NaïveBayes) and PSBML (NaïveBayes) with 10% noise are 4.41, 3.32, and 1.71, respectively. Similarly, with 20% noise, the impact values for AdaBoostM1 (DecisionStump), AdaBoostM1 (NaïveBayes)
and PSBML (NaïveBayes) are 5.02, 4.62, and 2.02, respectively. This shows that the PSBML algorithm is more robust to noise as compared to standard boosting. This is likely due to two reasons. First, in PSBML, the weighted sampling procedure is driven by the confidence of predictions only (prediction errors are not used), while AdaBoost credits larger weights to instances which are erroneously predicted. Second, PSBML makes use of a validation set to estimate the best classifier to be used for prediction of test instances, thus preventing overfitting.

### 9.2 Conclusion

In this chapter, detailed empirical analysis of PSBML trying to answer various questions pertinent to a scalable classifier has been performed. It is shown that the PSBML algorithm
Table 9.7: Performance of AdaBoostM1 (DS: Decision Stump), AdaBoostM1 (NB: Naive Bayes) and PSBML (NB: Naive Bayes) with no, 10%, and 20% noise.

<table>
<thead>
<tr>
<th></th>
<th>Adult</th>
<th>W8A</th>
<th>ICJNN1</th>
<th>Cod</th>
<th>Cover</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No Noise</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdaBoostM1/DS</td>
<td>87.10</td>
<td>77.80</td>
<td>93.40</td>
<td>92.80</td>
<td>75.70</td>
</tr>
<tr>
<td>AdaBoostM1/NB</td>
<td>87.20</td>
<td>93.30</td>
<td>84.30</td>
<td>95.70</td>
<td>85.30</td>
</tr>
<tr>
<td>PSBML/NB</td>
<td><strong>90.69</strong></td>
<td><strong>96.10</strong></td>
<td>81.79</td>
<td>91.79</td>
<td><strong>87.31</strong></td>
</tr>
<tr>
<td><strong>10% Noise</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdaBoostM1/DS</td>
<td>85.70</td>
<td>58.90</td>
<td><strong>92.82</strong></td>
<td>92.20</td>
<td>75.10</td>
</tr>
<tr>
<td>AdaBoostM1/NB</td>
<td>85.80</td>
<td>83.40</td>
<td>79.80</td>
<td><strong>95.10</strong></td>
<td>85.10</td>
</tr>
<tr>
<td>PSBML/NB</td>
<td><strong>90.46</strong></td>
<td><strong>96.01</strong></td>
<td>77.46</td>
<td>88.06</td>
<td><strong>87.14</strong></td>
</tr>
<tr>
<td><strong>20% Noise</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdaBoostM1/DS</td>
<td>85.10</td>
<td>57.10</td>
<td><strong>92.30</strong></td>
<td>92.10</td>
<td>75.10</td>
</tr>
<tr>
<td>AdaBoostM1/NB</td>
<td>84.88</td>
<td>79.01</td>
<td>79.70</td>
<td><strong>94.90</strong></td>
<td>84.20</td>
</tr>
<tr>
<td>PSBML/NB</td>
<td><strong>90.10</strong></td>
<td><strong>95.97</strong></td>
<td>77.42</td>
<td>86.98</td>
<td><strong>87.11</strong></td>
</tr>
</tbody>
</table>

is a generic meta-learner, boosting the performance across different classifiers independent of datasets used. The property of sensitivity of algorithm parameters on learning, which plays an important role in practice is analyzed and various bounds of grid sizes, was discussed. It was shown that PSBML scales linearly with training time and memory space, an ideal quality required for a large margin classifier. Comparison with the state-of-the-art optimized algorithms shows that PSBML is comparable or better than most in terms of practical metrics like accuracy and training time. Finally, the robustness of the PSBML algorithm in the presence of noise in training data as compared to traditional AdaBoost gives the required empirical evidence of the desired behavior in a classifier.
Chapter 10: EML Framework and Validation

In this chapter, I will be combining the two algorithms, EFG and PSBML, to form the proposed EML framework. I describe the EML framework in detail and validate it with real-world Big Data sequence based applications.

10.1 Introduction

In the previous chapters, two functionally orthogonal algorithms, EFG for feature generation in sequences and PSBML for large scale learning were introduced. Both algorithms were carefully analyzed and applied to a wide variety of applications involving bioinformatics, time series, and large scale learning independently. This chapter focuses on combining the important aspects of these algorithms in a single framework. The combined framework will be validated on large-scale sequence applications.

The main focus of this chapter is answering the question: How can the two algorithms, EFG and PSBML, be combined into a single framework for solving large scale learning in sequences? Some additional questions also arise when two algorithms are combined. Some of the other relevant questions that this chapter tries to answer are: Can EFG be parallelized to work in combination with PSBML to be an effective parallel learning algorithm? If EFG is parallelized, how can the features generated in parallel be combined for PSBML, which needs common set of features and parallel data? Can combining the two algorithms in a single framework retain their respective strengths without creating any negative side-effects?

Also, instead of using synthetic datasets, the validation of the framework is done on real-world Big Data sequence applications. Genome wide sequencing for splice site recognition, a complex and an open-ended problem in bioinformatics, is used as a typical computational
biology application to validate the claims of EML. A brain-computer interface based application, capturing the real-world subject’s brain signals as EEG time series to learn the models of various interaction, is used as the time series application to validate the EML framework. Detailed analysis, in terms of predictive models accuracy and speed of training, comparing EML with various state of the art in each of the applications are performed.

In this chapter, I will first describe the EML framework and the modifications made to the EFG and PSBML algorithms to combine them in an effective manner to form a single EML framework. The real-world datasets that are used for validation are described in detail next. Lastly, the experimental validation of EML and comparison with other algorithms is presented for each dataset.

10.2 EML Framework

This section gives the details of EML framework and how the two algorithms are combined in an effective manner. The entire framework with the components involved are shown in Figure 10.1. Each of the components or the processes are described in the following subsections.

10.2.1 Data Parallelization

The first process in the EML framework, is to split the training data using stratified folds, i.e., the same balance of labels and divided equally amongst all the nodes. The splitting of data into equal parts will be used both in the feature generation and parallel classification. The design choice on the upper and lower bounds on this is effected by processor/cores and the VC dimensions respectively. The grid configuration like the number of nodes running further depends on the processors/cores supported by the underlying hardware running the EML framework. For example, on a 256 POWER7 4.25 GHz IBM machine, a 256 nodes in $16 \times 16$ arrangement, will be dividing the large datasets in stratified manner into 256 samples. If the training data is not really too big, then dividing it by large number of nodes can lead to a configuration where each classifier/node in the learning mode has less
data than the minimum requirement as given by the VC dimension for that classifier [195]. Thus the balance between parallel execution and the VC dimension has to be achieved for effective learning and parallelization.

10.2.2 Parallel EFG

Parallel EFG in the grid consists of each node running the EFG algorithm on its stratified data, partitioned as discussed above, in parallel. Each node in the EFG grid generates the features in parallel in its own hall of fame storage described in the EFG chapter. Note that there is no communication between the EFG running in the nodes with other nodes in the
grid. However, in the future, highly fit individuals migrating among the neighborhood nodes or breeding individuals from neighbors may be incorporated to give more robust features. Many island model based evolutionary algorithms often employ migrations or neighborhood interactions for an effective evolutionary behavior [196].

10.2.3 Feature Combination and Selection

Since each node runs the EFG algorithm, there will be many disjoint sets of features generated by the whole grid, equal to the number of the nodes in the grid. These independent sets of features have to be combined to form a single set of features to be used in the PSBML algorithm. In this research, they are combined to find a unique set by forming a union among all the disjoint sets generated by nodes in the grid. One can easily contemplate more sophisticated mechanisms of feature weighting in the future. If a certain feature is truly a representative of an over-represented pattern, then it will emerge in more EFG nodes, and just reducing it to a single feature without taking into the account the normalized count of its appearance may seem bit contrived. But, imposing the feature weights puts a constraint on the classifier chosen by the PSBML algorithm to be one that handles weighted features and is a limited set. Also, complexity of feature weighting and instance weighting during PSBML iterations may lead to a positive feedback loop, introducing a biased induction and may perform badly on noisy data that has over-represented features!

Since there will presumably be a large number of features even after removing duplicates, a feature selection algorithm that further selects only relevant and non-redundant features becomes a mandatory dimensionality reduction step. In previous chapters, various feature selection algorithms such as information gain with ranking, correlational feature selection with genetic algorithms, and fast correlation feature selection with parallel greedy forward search were researched. Each of these methodologies have a trade-off between accuracy and the speed of searching for quality features. In the current EML framework, fast correlation based forward (FCBF) search which was the fastest and had very good accuracy in general has been employed as the dimensionality reduction technique [141].
10.2.4 Parallel Feature Interpretation

The feature set generated by the above feature reduction step is passed to each node. Each node in the grid has a feature interpreter that parses the features and maps it to numeric value 0/1. Thus parallel feature interpretation is accomplished and the raw sequence data gets transformed to high dimensional vector space in this process.

10.2.5 PSBML

There are no changes made to the PSBML algorithm to use it in the EML framework, as it is already designed as a parallel learning algorithm that works on the stratified samples of large training data on a grid configuration with some known features. The parallel data stratified and interpreted in the above step with the parallel features generated by feature selection respectively will be used in the PSBML algorithm. The choice of the classifier is generally a design decision based on the VC dimension, number of features, noise level in the data, and speed of the classifier in learning the model to name a few. Using the feature selection and reduction techniques, the features available to PSBML have more independence, lower correlation between them, and higher correlation with the label. Also because of the fast training speed and robustness to noise as compared to other classic classifiers analyzed in the PSBML analysis, Naïve Bayes is chosen as a default classifier in PSBML training nodes. The PSBML algorithm, as outlined in Chapter 8, reduces the training data to substantially smaller but relevant margin data. These reduced margin data with EFG features become the inputs for the classifier for learning the model to predict the future unseen testing/validation data.

10.3 Experimental Validation

Validation of the EML framework will be performed by applying it to two diverse applications in the sequence classification domain where training data sizes are large, a) genome
wide sequence splice site recognition and b) brain-computer interaction time series prediction.

**Hardware and Software** All the scalability experiments (in which the running times were measured) were run on a dual, 3.33 GHz, 6 processor Intel Xeon 5670 processor, accounting for 12 hardware threads. The EFG algorithm running on each node has basic settings of using motifs between length 2–8, IUPAC code ith ambiguity symbols for the motif, population size of 5000, external storage of 250 features, crossover rate at 0.7 and three different mutations at 0.1 each.

PSBML is implemented as a multithreaded standalone Java implementation that can run on any JVM version above 1.5. All the experiments with PSBML are run using a maximum heap size of 8GB and a number of threads equal to the number of nodes in the grid. Since the hardware had only 12 threads, all the experiments are run using $3 \times 3$ grid with C9 configuration of neighborhood.

The kernel methods, the weighted degree positional kernel (WD) and the weighted positional kernel with shift (WDS) method, are run using the publicly-available Shogun toolkit [197] along with the publicly-available LibSVM as the SVM implementation [198]. The SAX algorithm and KNN with Euclidean distance were run using the Matlab implementation [cite]. The SVM with RBF kernel is run using WEKA [187] and Markov-Chain is run using the JSTACS software [57].

### 10.3.1 Validation of EML: Genome-wide Sequence Classification

Genome projects are involved in annotating the entire genomes sequence for protein coded regions and other genome encoded features in different organisms. Whole genome sequence analysis have given important insights into genetic variations, explaining the evolutionary changes and mapping the gene functions to the genome sequence regions in these organisms [199]. These genomic sequences from various organisms are carefully annotated into various important regions such as regulatory, promoters, and splice sites regions. Instead
of using costly wet-lab experiments, the idea is to use computational methods to find interesting functional signals that become the predictor of these complex regions in the organisms. The computational biology community is constantly looking for a cost efficient, high throughput, and accurate predicting tools in the genome annotation task [200,201]. Training the model based on large data available in genome sequencing lends itself to the Big Data problem. Various learning algorithms such as Markov-Chain-based and kernel-based have performed relatively well in this area [14]. For validation of the EML framework, genome-wide recognition of splice site locations from two different organisms, namely *Caenorhabditis elegans* (“worm”) and *Drosophila melanogaster* (“fly”) are used.

**Datasets**

The dataset is extracted from the worm and fly genomes, prepared as in [14]. Briefly, the genome is aligned through blat with all known cDNA sequences available at [http://www.wormbase.org](http://www.wormbase.org) and all known EST sequences in [164] to reveal splicing sites confirming the introns and exons. Sequencing errors were corrected by removing minor insertions and deletions. In a next step, clustering of the alignments is performed initializing both cDNA and EST sequences to be in different clusters. Iteratively clusters are joined, if any two sequences from distinct clusters match to the same genomic location, showing alternate splicing. From the clustered alignments, a compact splicing graph representation is obtained, which can be easily used to generate a list of positions of true acceptor and donors.

Within the boundaries of the alignments, all positions exhibiting the AG, GT or GC dimer and which were not in the list of confirmed splice sites became the decoy or false acceptor and donor sites. The sequence length in all sets is 141 nt, for acceptor splice sequences the consensus dimer AG is at position 61, for donor GT/GC at position 81. Details of the dataset and distribution of true positives is given in Table 10.1.
Methodology and Comparison

The training/testing of methods for the two organisms on the acceptor and donor recognition tasks was done by 5-fold cross-validation. The reported auROC and auPRC are averaged scores over the five cross-validation unbiased splits. Each of the training cycles contains 4/5 of training data going through entire EML and 1/5 for testing repeated 5 times to get the average metrics. The Statistical Generative method, Markov-Chain (MC), and kernel-based methods Weighted Degree (WD) and Weighted Degree with Shift (WDS) are compared with EML for the auROC and auPRC metrics. The comparison is done by running the algorithm 10 times due to the time it takes and a paired-t test is used for statistical significance.

Results and Discussion

Tables 10.2 and 10.3 shows EML in general to be comparable in the accuracy metrics of auROC with state of the art statistical algorithm such as Markov-Chain and kernel methods. Kernel methods like WD and WD-S are still statistically significantly better in auROC measures as previously shown with just EFG algorithm, so combining with PSBML didn’t have any negative effect in this metric. As discussed in previous validation of EFG on bioinformatics applications, since the fitness function of EFG is biased towards finding more positive sequences, the auPRC is statistically significantly better than all the three methods.

The average time taken by the kernel method WD and WD-S was 6.5 and 7.0 hours, while the EML framework took 3.5 hours on average on both the datasets. The Markov-Chain method took more than 10 hours on average for both the datasets. Thus, on both of the large training datasets, it confirms the hypothesis that the EML framework, using the EFG and PSBML, combines the best of both worlds in terms of finding discriminating features and parallel learning respectively.

For the worm dataset, inspecting the features generated by EML confirmed the presence of 7mer motifs \textbf{GGTAAGT, AGGTAAG, GGTAGGT} around $-43$ nt, matching the donor consensus \textbf{AGGTAAGT}. Also an important positional feature was present in the
Table 10.1: Whole organism Genome training datasets size and true positives fractions in the total set.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Worm</th>
<th>Fly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acceptor</td>
<td>Donor</td>
</tr>
<tr>
<td>Training Data size</td>
<td>1,105,886</td>
<td>1,744,733</td>
</tr>
<tr>
<td>True Positive Fraction</td>
<td>3.6%</td>
<td>2.3%</td>
</tr>
</tbody>
</table>

Table 10.2: Area Under ROC (A) and Area Under PRC (P) comparisons of various state of the art algorithms with EML framework on the Worm dataset for both acceptor and donor splice sites.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Worm</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MC</td>
<td>WD</td>
<td>WD-S</td>
<td>EML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Acceptor</td>
<td>Donor</td>
<td>Acceptor</td>
<td>Donor</td>
<td>Acceptor</td>
</tr>
<tr>
<td>measures</td>
<td>A</td>
<td>P</td>
<td>A</td>
<td>P</td>
<td>A</td>
</tr>
<tr>
<td>average</td>
<td>99.6</td>
<td>90.2</td>
<td>99.4</td>
<td>90.1</td>
<td>99.36</td>
</tr>
</tbody>
</table>

Table 10.3: Area Under ROC (ROC) and Area Under PRC (PRC) comparisons of various state of the art algorithms with EML framework on the Fly dataset for both acceptor and donor splice sites.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fly</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MC</td>
<td>WD</td>
<td>WD-S</td>
<td>EML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Acceptor</td>
<td>Donor</td>
<td>Acceptor</td>
<td>Donor</td>
<td>Acceptor</td>
</tr>
<tr>
<td>measures</td>
<td>A</td>
<td>P</td>
<td>A</td>
<td>P</td>
<td>A</td>
</tr>
<tr>
<td>average</td>
<td>98.7</td>
<td>80.27</td>
<td>99.12</td>
<td>78.4</td>
<td>99.02</td>
</tr>
</tbody>
</table>
region −18 nt to −14 nt consisting of TAAT, which is a well known branch site signal [170].
Shift-positional features around −3 nt consisting of motifs TTTCAGG and TTTCAGA matching the acceptor consensus TTTCAG(A/G) exactly were also present.

For the fly datasets, overrepresented features such as compositional patterns of GGTAA, GGTGAG, GGTGAGT, which are close to the consensus donor patterns, are top features in the EML framework. Acceptor features showed compositional features such as ATTTTCAG, TTACAGA, CTTGCAGA close to the consensus acceptor patterns. Interesting regional patterns like presence of GTAAGT in the downstream and GTA in the upstream matches the results from past researches [170].

10.3.2 Validation of EML: Brain Computer Interaction as Time Series

Brain-Computer Interface (BCI) research is about the automatic translation of neural commands into control signals that can be used to control applications such as input programs, wheelchairs or neuroprostheses [202]. A BCI system has been effective as a communication option for severely disabled patients or as an additional interaction channel for the healthy users. The Berlin Brain-Computer Interface project (BBCI), has developed an electroencephalogram (EEG) based system that captures brain signals as EEG for different human subjects [203]. The EEG signals can be viewed as time series data inputs and the interactions as the labels for these data. BBCI holds a competition every year giving the labeled datasets to promote research in the BCI area [204]. These time series classification datasets are some of the largest available real-world datasets in terms of training data sizes. This time series dataset has all the components of complex time series such as mixed signals used in EEG, longer time intervals, noise, etc., to make it an ideal case study for time series classification [204]. As an application of the EML framework for time series classification, this research will use one of the latest datasets from the competition and compare it with the top winners and other time series algorithms for validation.
The Dataset

The dataset used for the experiment is from the BBCI time series competition, and contains data from 3 normal human subjects during 4 non-feedback sessions. The subjects sat in a normal chair, relaxed arms resting on their legs. There are 3 tasks:

1. Imagination of repetitive self-paced left hand movements, (left, class 2),
2. Imagination of repetitive self-paced right hand movements, (right, class 3),
3. Generation of words beginning with the same random letter, (word, class 7).

The raw EEG potentials were first spatially filtered by means of a surface Laplacian. Then, every 62.5 ms, i.e., 16 times per second, the power spectral density (PSD) in the band 8–30 Hz was estimated over the last second of data with a frequency resolution of 2 Hz for the 8 centro-parietal channels C3, Cz, C4, CP1, CP2, P3, Pz, and P4. As a result, an EEG time series sample is a 96-dimensional vector (8 channels times 12 frequency components). Also, to keep the EML algorithm working on a binary level, the left and right hand movements are combined as a single label, with the other being the word uttered. The data has around 50K training and 10K testing data.

Methodology and Comparison

The training and testing data of all three subjects were combined for the two class labels, i.e., hand movements and random letters. The preprocessing of time series data is done using SAX to generate sequences in DNA characters of A, C, G, and T. The parameters chosen for the SAX representation were the ones which gave optimum performance running just SAX. BBCI’s top competitor was a model using SVM with RBF kernel with tuned kernel parameters, which are used as baseline for comparison. To make the comparison even more comprehensive, comparisons is also made against SAX and KNN with Euclidean distance. The runs were done 10 times only due to the time required for running each tests, mean accuracy as measured as percentage correct is used as metric and paired-t test is used for statistical significance comparisons.
Results and Discussion

Table 10.4: Comparison of EML using SAX preprocessing with the other methodologies.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAX</td>
<td>62.27</td>
</tr>
<tr>
<td>EML(SAX)</td>
<td>67.28</td>
</tr>
<tr>
<td>KNN</td>
<td>66.63</td>
</tr>
<tr>
<td>SVM-RBF (BBCI top)</td>
<td>68.50</td>
</tr>
<tr>
<td>Fisher Discriminant Analysis Kernel (BBCI top2)</td>
<td>65.67</td>
</tr>
</tbody>
</table>

Table 10.4 shows the comparison of EML using SAX preprocessing with the other methodologies. There were no significant difference between SVM-RBF and EML, though SVM-RBF had a better mean than EML. SAX had the lowest performance on the dataset, and since EML works on top of these discretized and dimensionally reduced datasets, the chances of it sometimes get lowered when the SAX performance is lower. Note that SAX performance depends on parameters such as sliding-window length and word length. A comprehensive parameter sweep needs to be done to validate against various combinations. Since the goal here was not to be the best, but compare relatively well with SAX and others, a small range of parameters ranging from [8,8] to [24, 16] in steps of 4 were performed.

EML using SAX improved on SAX performance by 5.1, was better than the discriminant kernel by 1.61, better than KNN by 0.65, and was very close to the top perform in BBCI competition in mean. The improvement over using just SAX in terms of accuracy reaffirms the importance of the complex features that EFG algorithm is capable of finding. The top performers using non-linear kernels had training times on average of 1.5 and 1.25 hours, while EML on average had 0.5 hours training time, showing the improvement because of parallel processing both at feature learning and model learning.
10.4 Conclusion

In this chapter, two algorithms—EFG and PSBML—were combined with fast feature selection to form the EML framework. The EML framework was first validated on an open problem in bioinformatics, i.e., splice-site recognition in genome-wide sequences on two separate datasets. The promise of EML, showing great accuracy and precision with smaller training times, can be considered to be an important step forward in the next generation high throughput predictive genome sequencing tools. The general benefits of EML were validated by using it in a separate time series application, again with better accuracy and lower training times.
Chapter 11: Conclusion and Future Work

11.1 Conclusion

Current challenges in sequence classification are most highlighted by the need of transparent models based on well known features and the scaling issues caused by the size of datasets. In this dissertation, I have proposed two algorithms—Evolutionary Feature Generator (EFG) and Parallel Spatial Boosting Machine Learner (PSBML)—to address these issues.

In chapter 3, using the building blocks theory and GP as the implementation, EFG was designed to construct discriminating features in the sequence classification applications. In chapters 4–6, EFG was validated with different complex bioinformatics sequence classification problems such as regulatory region finding, Alu sequence identification, promoter region identification and splice site region recognition. EFG showed its strength in not only finding classification models that have equal or better classification accuracy but also in automation of finding features known to have biological significance without any help from domain experts. The generic nature of EFG was validated by combining it with SAX and using it in different time series classification applications in chapter 7. EFG in combination with SAX was very effective and equal or better than more complex state of the art methods in time series classification.

The PSBML algorithm, based on parallel spatial evolutionary algorithm, was independently analyzed and validated for its large scale learning properties in chapters 8 and 9. In chapter 8, the PSBML algorithm was shown theoretically to be a large margin classifier, an important property for a learning algorithm, and it was empirically shown to be a superior meta-learning algorithm similar to a boosting algorithm. The PSBML algorithm was also shown to scale almost linearly in training time, memory usage, and threads, giving it a distinct advantage over other the traditional large margin classifiers like AdaBoosting and
SVM. The PSBML algorithm also had a key advantage of being noise resilient as compared to AdaBoost because of the choice of using confidence for weighting the dataset iteratively and having a separate validation set for preventing overfitting.

EFG and PSBML were combined to form the EML framework for parallel feature generation and parallel learning. The EML framework and its detailed description was laid out in chapter 10. It was validated on the well known bioinformatics large data problem of genome-wide splice site recognition. On both the splice site datasets, the EML framework exhibited the best precision while maintaining the key advantage of almost doubling the training speed as compared to the other state of the art algorithms. The EML framework also showed its potential in large scale time series classification by not only improving the accuracy in SAX and being comparable to other algorithms but also in reducing the training time substantially as compared to the winner algorithm using an SVM.

11.2 Future Work

To address some of the future work on this dissertation, I have created two high level topics of algorithm-related enhancements and application oriented future work. The details are given in the following two sections:

11.2.1 Algorithmic Enhancements

1. The EFG algorithm in future can have Boolean simplification in the generation rather than as a post processing step. This might further reduce the bloat issue and improve the search capability even more.

2. The EFG algorithm may employ other implementation techniques like grammatical evolution instead of GP. This may be interesting work to see the effect of fixed size encoding of GP on both bloat and features discriminating power.

3. The EFG algorithm may employ multi-class discrimination as an inbuilt fitness function rather than combining the features externally in an ensemble.
4. The PSBML algorithm can be enhanced for a distributed network based deployment. Using hierarchical blocks where a $3 \times 3$ can run on a 12 core machine and $2 \times 2$ can run on 4 core machine for the original larger $5 \times 5$ grid can be an interesting parallel deployment for assymetrical cloud based setups.

5. The EML algorithm, which combined features parallely from EFG, can introduce weights on over-represented features.

6. The EML algorithm can be tried with other feature selection algorithms which can use feature weights discussed above in addition to maximizing the feature-class relevance and minimizing the feature-feature redundancy.

11.2.2 Future Applications

1. The EFG algorithm can be applied to other domains like music or sound based classification, using music notes and sound frequencies as the alphabets respectively.

2. The EFG algorithm may be adapted to other biological sequences like amino acids and proteins with the right alphabets and motif structures.

3. The PSBML algorithm may be adapted to use in other machine learning problems like semi-supervised learning and unsupervised learning. Using either the semi-supervised learners or using only the confidence based and a nearest neighbor classifier, a parallel semi-supervised learning seems plausible. Using inter-cluster distances to maximize the separation, a parallel $k$-means, like unsupervised learner, can be developed using PSBML concepts.

4. The PSBML algorithm with some enhancements can be also adapted to the other interesting applications such as the subspace clustering and co-clustering problems in machine learning. The ability to solve the high dimensional clustering problem by breaking down the problem into smallest features and using PSBML to finding the emerging clusters in parallel seems like a viable design.
5. The PSBML algorithm may be used in other applications such as protein structure prediction by parallelizing and combining structure learners in the PSBML nodes giving the local-global search for energy minima.
A.1 Annotation of Splice Sites

Here we provide more results of annotating human genome for splice sites as discussed in chapter 6.

**Sequence 3**  Human chromosome with id “45868_HSCDIR2” from genesplicer dataset is run through trained model from human splicing as detailed in the chapter 6. The observations are as below

1. Exon2, Exon3, Exon4 rightly identified
2. Initial Exon acceptor and donor not considered due to boundary condition (first 80 are considered outliers)
3. Final Exon donor not considered due to boundary condition.
4. One False Positive Donor at 81 corresponds to GT and one false positive acceptor around 80.

**Sequence 4**  Human chromosome with id “45378_AB005548” from genesplicer dataset is run through trained model from human splicing as detailed in the chapter 6. The observations are as below

1. exon2 rightly identified
2. Initial exon acceptor and donor not considered due to boundary condition (first 80 are considered outliers)
3. Final exon acceptor and donor not considered due to boundary condition.

4. No false positives in acceptor and donors.

**Sequence 5** Human chromosome with id “45378_AB005548” from genesplicer dataset is run through trained model from human splicing as detailed in the chapter 6. The observations are as below

1. exon2, exon3, exon4 rightly identified

2. Initial exon acceptor not considered due to boundary condition (first 80 are considered outliers)

3. Final exon donor not considered due to boundary condition.

4. One false positive in Donor 82 position.
Figure A.2: Genome Splice Predictions for 45378_AB005548 showing predictions and true sites

Figure A.3: Genome Splice Predictions for 4546_AB016492 showing predictions and true sites
A.2 PSBML Experiments

A.2.1 Scalability Comparisons

Here I have performed more experiments with synthetic datasets to compare the performance of PSBML. Here are the two additional experiments.

A three dimensional decision boundary was generated using the Rastrigin function

\[ f(x) = 20 + (x_1^2 - 10\cos(2\pi x_1)) + (x_2^2 - 10\cos(2\pi x_2)) \]  
(A.1)

The two dimensions \(x_1\) and \(x_2\) were sampled between \([-1, 1]\), and the \(y = f(x)\) dimension was sampled between \([0, 40]\).

Another three dimensional decision boundary was generated using the Rosenbrock function

\[ f(x) = (1 - x_1^2)^2 + 100(x_2 - x_1^2)^2 \]  
(A.2)

The two dimensions \(x_1\) and \(x_2\) were sampled between \([-3, 3]\) and the \(y = f(x)\) dimension was sampled between \([0, 2500]\).

In each case we generated a training dataset with a million instances. However, this was too large for an SVM (LIBSVM) with a non-linear kernel (RBF kernel). We kept reducing the data and found that at 10% (i.e. around 100K sample points) the non-linear SVM could run in reasonable time.

A.2.2 Accuracy Comparison on a Large Real-world Dataset

The goal of this experiment is to perform a statistical analysis comparing PSBML to known classifiers and meta-classifiers on a sufficiently large real-world dataset with non-overlapping train/test distributions. We used the NSL-KDD dataset with approximately 120K training data and 22K test data [205]. The NSL-KDD dataset retained all the important statistical characteristics of KDDCup-99, but it’s smaller so that performance can be evaluated on
Table A.1: Training speed (in seconds) and accuracy for the Rastrigin and the Rossenbrock datasets.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Rastrigin</th>
<th></th>
<th>Rossenbrock</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Speed</td>
<td>Acc</td>
<td>Speed</td>
<td>Acc</td>
</tr>
<tr>
<td>SVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LP-SVM (Linear)</td>
<td>25.20</td>
<td>77.23</td>
<td>43.20</td>
<td>59.80</td>
</tr>
<tr>
<td>LibLinear</td>
<td>13.20</td>
<td>69.3</td>
<td>20.12</td>
<td>51.1</td>
</tr>
<tr>
<td>SGDT (10 iterations)</td>
<td>5.20</td>
<td>58.49</td>
<td>4.20</td>
<td>59.89</td>
</tr>
<tr>
<td>SVM-PERF (Linear)</td>
<td>8.10</td>
<td>76.01</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BVM (RBF)</td>
<td>4.80</td>
<td>43.03</td>
<td>5.20</td>
<td>44.53</td>
</tr>
<tr>
<td>LibSVM</td>
<td>2512.2</td>
<td>85.20</td>
<td>2713.4</td>
<td>97.10</td>
</tr>
<tr>
<td>(RBF, 10% data)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boosting</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdaBoostM1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ParalleAdalBoost</td>
<td>30.1</td>
<td>84.22</td>
<td>31.2</td>
<td>81.30</td>
</tr>
<tr>
<td>(9 threads, 10 iterations)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSBML</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSBML (C4.5)</td>
<td>28.6</td>
<td><strong>92.2</strong></td>
<td>22.6</td>
<td><strong>99.6</strong></td>
</tr>
</tbody>
</table>

a variety of algorithms. We ran the PSBML algorithm with a $3 \times 3$ grid, the C9 neighborhood, and Naive Bayes as the base classifier. We obtained similar performance using other classifiers like decision trees, but to reduce experimental computation time we used Naive Bayes. We used the top ranking competitive methods (as shown in [205]), performed 30 runs of 10-fold cross validation, and compared the results using a paired $t$-test for significance. The results are shown in Table A.2. Boosting and PSBML are very close in performance. However, in this case, PSBML is statistically significantly better, illustrating the ability of PSBML to perform well on a large dataset, while maintaining the advantage of parallelization.
Table A.2: Performance on NSL-KDD.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Accuracy (auROC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecisionTee (J48)</td>
<td>81.02</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>80.67</td>
</tr>
<tr>
<td>NB-Tree</td>
<td>82.8</td>
</tr>
<tr>
<td>NeuralNetwork</td>
<td>77.42</td>
</tr>
<tr>
<td>SVM (RBF)</td>
<td>60.12</td>
</tr>
<tr>
<td>RandomForest</td>
<td>80.67</td>
</tr>
<tr>
<td>RandomTree</td>
<td>81.5</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>90.77</td>
</tr>
<tr>
<td>PSBML (NaiveBayes)</td>
<td>93.5</td>
</tr>
</tbody>
</table>
Bibliography
Bibliography


152


Curriculum Vitae

Uday Kamath is a Ph.D. student in the information technology department at of the George Mason University Volgenau School of Engineering. He earned a B.E. in Electronics Engineering from Bombay University in 1996, an M.S. in Computer Science from University of North Carolina in 1999, and is completing his Ph.D. in the spring of 2014.

Uday is advised by Dr. Kenneth De Jong, whose lab studies the theory and application of evolutionary computation. Uday’s research involves building evolutionary algorithms for different machine learning applications ranging from feature construction, feature selection and supervised learning. This work has resulted in numerous publications in the fields of both computer science and computational biology—including a Honorable Mention award at Humies competition held at GECCO 2012.

Uday works as analytics architect at Detica, responsible for designing scalable and robust algorithms for detecting fraud in various financial, insurance, healthcare and government institutions.

Education

• Masters of Science, Computer Science, University of North Carolina at Charlotte, 1999

• Bachelor in Engineering, Bombay University, 1996

Awards

• Honorable Mention Award, GECCO HUMIES Competition 2012 for “Genetic Programming Based Feature Generation for Automated DNA Sequence Analysis ”

Journal Articles


Conference Papers


Books and Book Chapters