Neural Computing and Applications manuscript No.
(will be inserted by the editor)

Integrating Genomic Binding Site Predictions using Real-valued Meta Classifiers

Yi Sun[1], Mark Robinson[1], Rod Adams[1], Rene te Boekhorst[1], Alistair G. Rust[2], Neil Davey[1]

[1] Science and Technology Research Institute,
University of Hertfordshire,
College Lane, Hatfield,
Hertfordshire AL10 9AB

[2] Institute for Systems Biology,
1441 North 34^{th} Street,
Seattle, WA 981-3, USA

Received: date / Revised version: date

Abstract Currently the best algorithms for predicting transcription factor binding sites in DNA sequences are severely limited in accuracy. There is good reason to believe that predictions from different classes of algorithms could be used in conjunction to improve the quality of predictions. In this paper, we apply Single Layer Networks, Rules Sets, Support Vector Machines and the Adaboost algorithm to predictions from 12 key real valued algorithms. Furthermore, we use a 'window' of consecutive results as the input vector in order to contextualise the neighbouring results. We improve the classification result with the aid of under- and over- sampling techniques. We find that Support Vector Machines and the Adaboost algorithm outperform the original individual algorithms and the other classifiers employed in this work. In particular they give a better tradeoff between Recall and Precision.

1 Introduction

In this paper, we address the problem of identifying transcription factor binding sites in DNA sequences. There are many different algorithms for searching binding sites [36, 5, 8, 7] in current use. However, most of them produce a high rate of false positive predictions. This is problematic for practicing biologists who wish to validate these

results - testing a prediction is costly. We attempt to reduce these false positive predictions using classification techniques taken from the field of machine learning.

To do this we first integrate the results from 12 different base algorithms for identifying binding sites, using non-linear classification techniques. To further improve classification results, we employ windowed inputs, where a fixed number of consecutive results are used as an input vector in order to contextualise the neighbouring results. The data has two classes labeled at the nucleotide level as either part of binding sites or non-binding sites, with about 93% being non-binding sites. We make use of sampling techniques, working with a traditional neural network: single layer networks (SLN), rules sets (C4.5-Rules), a contemporary classification algorithm: support vector machines (SVM) and the Adaboost algorithm.

In previous work we have used binary valued base algorithms [26], here we extend this to use as much information as possible, as provided by the real valued base algorithms.

We expound the problem domain in the next section. In Section 3, we introduce the datasets used in this paper. We explain how we apply under- and over- sampling techniques in Section 4. Section 5 presents the classification algorithms used in this work. A set of common metrics and receiver operating characteristics graphs for assessing classifier performance are covered in Section 6. Section 7 briefly introduces our experiments and gives all the experimental results. The degree of matching between prediction and experimental verification is visualised in Section 8. The paper ends in Section 9 with conclusions.

2 Problem Domain

It is increasingly acknowledged that much of the variation in complexity of organisms is due to differences in the regulation of gene activity rather than to differences in the genetic specifications for protein coding per se. Gene activity is dynamic, being regulated by the complex interplay of gene regulatory networks and all their many components. Whereas the general principles underlying the translation of the coding regions of genes (exons) into their protein products are largely comprehended, the mapping between a gene's expression and the information contained in (non-coding) regulatory regions of the genome is not well understood. These regulatory regions are short stretches of DNA upstream or downstream of the position where gene transcription begins. They are generally composed of dense clusters of so-called transcription factor binding sites (TFBS). In turn, these binding sites are recognized by transcription factors, proteins that - upon binding to them - act as repressors or activators, thus controlling the rate of transcription.

Recent research has made clear that genetic regulatory mechanisms are much more intricate than was once assumed. For example, a single base substitution will commonly modify the intensity of the interaction be-

tween transcription factor and DNA rather than abolish it. This implies that such regions are fairly robust to mutations. It also allows a relatively small number of transcription factors to produce a wide range of patterns of gene expression. Furthermore, certain weakly binding transcription factors require the assistance of other, more vigorously binding proteins whereas others compete for access to a single regulatory site. The situation is further complicated by the fact that certain regulatory regions are more accessible to transcription factors than others [33]. In higher eukaryotes some of these regions may be located far upstream or downstream of the target gene. These are called enhancers or cis-regulatory modules, which make possible additional heuristics for their computational prediction.

One of the most exciting, but also challenging areas of current biological research is therefore devoted to the understanding of the regulation of gene expression. The identification of regulatory regions and transcription factor binding sites clearly forms an essential step in this endeavour. However, although as much as 50% of the human genome is estimated to be regulatory [21], most of this is not yet deciphered. The desire for large scale understanding has driven the development of high throughput methods. It favours computational approaches because these sidestep the ultimately more reliable but slow and expensive route of experimental verification.

Regulatory regions appear to have statistical properties that help to distinguish them from other parts of the genome, such as the over-representation of similar sequential motifs [10,34,2] and a sequential persistency and an informational entropy that is intermediate between those of exons and non-coding, non-regulatory DNA [30]. These and other statistical properties are exploited by various types of algorithms for predicting TFBS, or their motifs, from raw sequence data. *Enumerative algorithms* build or assume a background model of base pair distribution in the DNA non-coding regions that do not contain TFBS, and look for motifs in the given sequence that are statistically significant against this background. They are often applied to (putative) co-regulated genes found by expression (micro-) array analysis. Another enumerative approach is *phylogenetic footprinting*, which identifies motifs by comparing sequences from phylogenetically related species. *Iterative algorithms* use techniques such as Expectation Maximization to define weight matrices for the most over-represented motifs. These algorithms also require a collection of upstream sequences from possibly co-regulated genes and a model for background distribution. *Content based algorithms* segment the available sequence into a 'lexicon' of words and look for regularities in the way one would proceed to decipher a text consisting of a long string of letters written in an unknown language in which words are not delineated.

The downside of the developments sketched above is that we are currently burdened by a bewildering variety of algorithms. Nowadays it takes quite some computational and statistical expertise to make an educated choice about what methods to use. Even more worrying is the fact that many of the published algorithms are still severely limited in accuracy and of uncertain quality. Not only is picking regulatory regions out of the background of other non-coding DNA sequences a non-trivial enterprise, also the fierce competition in the prediction market hardly allows for a thorough evaluation. For example, in a large sample of annotated yeast promoter sequences, a selection of 12 key algorithms, as used in this study, were unable to reduce the error rate of positive predictions below 80%, with between 20% and 65% of annotated binding sites recovered. The choice of algorithms for use in this study was influenced by two factors: firstly, the necessity of ensuring representation of all major prediction strategies and secondly, we limited the choice to algorithms that were easily incorporated into the *Mogul Motif Prediction Pipeline* developed at the Institute for Systems Biology (ISB) (`http://labs.systemsbiology.net/bolouri/Mogul/`) which was used for this work. These algorithms represent a wide variety of approaches to the problem of transcription factor binding site prediction, such as the use of regular expression searches, *position weight matrix* (PWM) scanning, statistical analysis, co-regulation and evolutionary comparisons.

As already stated none of our 12 algorithms perform very well independently. One way to overcome this problem is to combine the outcomes of a large number of algorithms instead of relying on the result of just one. The importance of such meta-classifiers goes without question and their investigation will therefore be at the core of this paper. In the work described here we take the results from the 12 aforementioned algorithms and combine them into 2 different feature vectors, as shown in next section. We then investigate whether the integrated classification results of the algorithms can produce better classifications than any one algorithm alone.

3 Description of the Data

The data consists of a set of annotated promoters taken from the *S.cerevisiae promoter database* (`http://rulai.cshl.edu/SCPD/`), which is one of the largest and most reliable collections of experimentally verified annotated data available. The dataset has 68910 possible binding positions and a prediction result for each of the 12 algorithms. Table 1 shows the categorisation of these 12 algorithms, where two of them are the same algorithm (Dream) running in two different prediction modes: over and under represented motifs.

The label information contains the best information we have been able to gather for the location of known binding sites in the sequences. We use -1 to denote the prediction that there is no binding site at this location and $+1$ to denote the prediction that there is a binding

Table 1 Categorization of algorithms used in this study.

Strategy	**Algorithm**
Scanning	Fuzznuc [36] Motif Scanner [31] Ahab [23]
Statistical	PARS [38] Dream (over and under represented motifs) [1] Verbumculus [3]
Co-Regulatory	MEME [5] AlignACE [16] Sampler (Institute for Systems Biology) [37]
Evolutionary	SeqComp [8] Footprinter [7]

site at this location. For each of the 12 base algorithms, a prediction result can be either binary or real valued, see Figure 1. The data therefore consists of 68910 12-ary real vectors each with an associated binary label.

In this work, we divide our dataset into a training set and a test set: the first 2/3 (including 45943 nucleotides) is the training set and the last 1/3 (including 22967 nucleotides) is the test set. Amongst the data there are repeated vectors, some with the same label (repeated items) and some with different labels (inconsistent items). It is obviously unhelpful to have these repeated or inconsistent items in the training set, so they are removed. We call the resulting data the *consistent training set*. However in the case of the test set it is reasonable to consider both the full set of data and the subset consisting of only the consistent test items. The removal of repeated and inconsistent data dramatically reduces the number of data items: only 28% (about 12790 nucleotides) of data is left in the training set (see Table 2).

As the data is drawn from a sequence of DNA nucleotides the label of other near locations is relevant to the label of a particular location. We therefore contextualise the training and test data by windowing the vectors, as shown in Figure 2. We use the locations up to three either side, giving a window size of 7, and a consequent input vector size of 84. This has the considerable additional benefit of eliminating most of the repeated and inconsistent data: as can be seen in Table 2 now less than 7% of the training data is lost : 42919 of the original 45943 data items are retained.

The training set consists of either single vectors or windowed vectors. In both cases only consistent, non-repeating data is used. The test data consists of either single vectors or windowed vectors as appropriate. Either the full test set or the relevant consistent subset is used. There is however, a special case, namely when we want to compare the windowed model with the single input version. Here we want to evaluate the windowed model on the locations represented in the consistent test set of the single vector model. We therefore construct a test set for the windowed model consisting of only those vectors corresponding to the 7 locations around each of the data

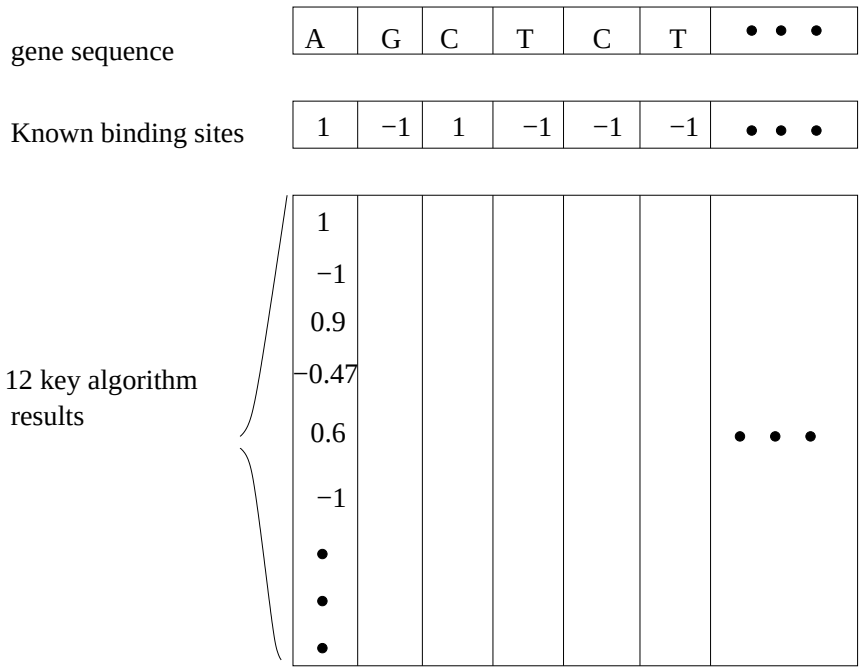

Fig. 1 The dataset has 68910 columns, each with a possible binding prediction (binary or real valued). The 12 algorithms give their own prediction for each sequence position and one such column is shown.

Table 2 Description of the datasets used in this work.

Type			Size
Training	consistent	single	12790
		windowed	42919
Test	consistent	single	5966
		restricted windowed	5966
	full	single	22967
		windowed	22967

points in the single consistent test set, this is referred to as the *restricted windowed set*.

4 Sampling Techniques for Imbalanced Dataset Learning

In our dataset, there are less than 10% binding positions amongst all the vectors, so this is an *imbalanced* dataset [19]. Since the dataset is imbalanced, the supervised classification algorithms will be expected to over predict the majority class, namely the non-binding site category. This is demonstrated in the very poor results shown in Section 7.2.2 for imbalanced data. There are various methods of dealing with imbalanced data [18]. In this work, we concentrate on the data-based method [11]: using under-sampling of the majority class (negative examples) and over-sampling of the minority class (positive

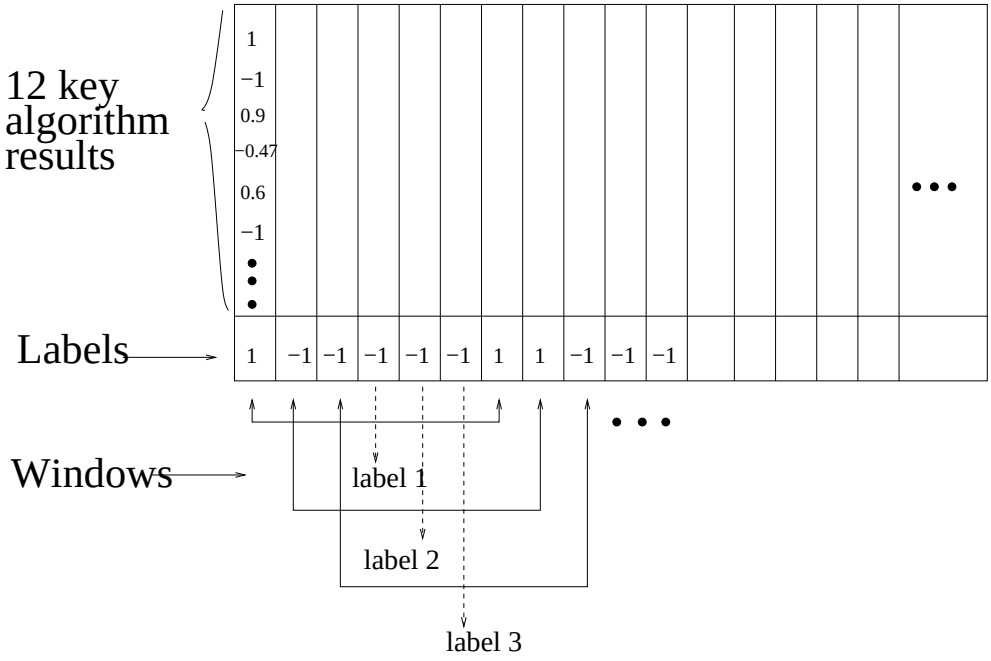

Fig. 2 The window size is set to 7 in this study. The middle label of 7 continuous prediction sites is the label for a new windowed inputs. The length of each windowed input now is 12×7.

examples). We combine both over-sampling and under-sampling methods in our experiments.

For under-sampling, we randomly selected a subset of data points from the majority class. The over-sampling case is more complex. In [19], the author addresses an important issue that the class imbalance problem is only a problem when the minority class contains very small subclusters. This indicates that simply over sampling with replacements may not significantly improve minority class recognition. To overcome this problem, we apply a synthetic minority over-sampling technique as proposed in [11]. For each pattern in the minority class, a new pattern belonging to the minority class can then be generated as follows:

- We search for its $K-$nearest neighbours in the minority class using a Euclidean distance like measure. (Since the dataset is a mixed one of continuous and binary features, we follow the suggestion in [11] to calculate this measure, see below.)
- for continuous features, a new feature value denoted by x_d^{new} is given by:

$$x_d^{new} = x_d^n + rand(0,1) \times (x_d^n - x_d^{NN})$$

where the difference of each feature between the pattern ($\mathbf{x}^n$) and its nearest neighbour ($\mathbf{x}^{NN}$) is taken, and then multiplied by a random number between 0 and 1, and added to the corresponding feature of the pattern.

– for binary features, the majority voting principle is applied to each element of the $K-$nearest neighbours in the feature space.

We take 5 nearest neighbours, and double the number of items in the minority class.

The calculation of the Euclidean distance like measure is given in [11] and is as follows: if the binary features do not differ between a pattern and its nearest neighbour, then a zero difference is used in the measure; if they do differ, then the median of standard deviations of all the real valued features for the minority class is used.

For example: assume two points, $\mathbf{x}'$ and $\mathbf{x}$, from the minority class in D dimensions, with the first two dimensions being binary, the Euclidean distance is given as follows:

$$Eulidean\_distance = \sqrt{Med^2 + Med^2 + \sum_{d=3}^{D}(x'_d - x_d)^2},$$

where $Med = median\big(std(x_3), std(x_4), \ldots, std(x_D)\big)$, and *std* denotes the standard deviation.

The actual ratio of minority to majority class is determined by the under-sampling rate of the majority class. According to our previous experience, we set the final ratio to a half, which works well in this work.

5 Supervised Classifiers

In this section, we briefly introduce the supervised classifiers used in our experiments. Readers who are interested in those classification techniques can follow the references to learn more.

Suppose we have a training dataset $\{\mathbf{x}^n, t^n\}_{n=1,\ldots,N}$, where $\mathbf{x}^n \in \mathcal{R}^D$ is the input, and t^n is the corresponding target. In classification the task is to assign each new input $\mathbf{x}$ to one of the classes, c_i, where $i = 1, \ldots, C$. We shall denote y as the corresponding predictor of each input.

5.1 Single Layer Networks (SLN)

First of all, we consider a simple type of classifier, using a non-linear function $g(\cdot)$ on the weighted sum a of the components of an input, and which can be written as follows [6]:

$$y = g(a) \,, \tag{1}$$

where

$$a = \mathbf{w}^T\mathbf{x} + w_0 \,, \tag{2}$$

and $g(\cdot)$ is an activation function, $\mathbf{w}$ is the weight vector determining the orientation of the separating hyperplane, and w_0 is the bias determining the position of the hyperplane in the data space.

For the two-class problem, one choice for the activation function is the logistic sigmoid activation function given by

$$g(a) = \frac{1}{1 + \exp(-a)} \,. \tag{3}$$

The logistic sigmoid function is monotonic, and its interval is $(0, 1)$. It allows the outputs of the discriminant to be interpreted as posterior probabilities.

The single layer neural network is equivalent to a logistic discrimination analysis. This is employed because the weights from the network can easily be interpreted as weighting factors for the algorithms, giving an easily understood combination rule.

5.2 Rules Sets

Rules sets applied in this work are derived from C4.5 decision trees [22], which induces classification models. In a decision tree, each node corresponds to each attribute of data, while each arc to a possible value of the attribute. A leaf of the tree specifies a classification result.

The information involved in each node is measured by *Entropy*, denoted by Info$(\cdot)$. Given a probability distribution $P = \{p_1, \ldots, p_J\}$, Entropy is defined as follows:

$$\text{Info}(P) = \sum_{j=1}^{J} p_j \cdot \log p_j. \tag{4}$$

To rank attributes and to build decision trees, a notion *Gain* is given by:

$$\text{Gain}(\mathbf{x}, N) = \text{Info}(\frac{n_{c_i}}{N}) - \sum_{l=1}^{L} \frac{N_l}{N} \text{Info}(\frac{n'_{c_i}}{N_l}); \tag{5}$$

where n_{c_i} is the number of data points belonging to class c_i in the dataset, L the number of partitions divided on the values of x, N_l the number of data points in partition l, and n'_{c_i} the number of data points belonging to class c_i in the partition N_l.

The attribute with greatest gain among the attributes and not yet considered in the path from the root is located at each node.

Each path in the decision tree, from the root to a leaf, determines a rule and all the rules constitute the rule set.

5.3 Support Vector Machine (SVM)

The SVM is a recently developed technique in the machine learning field. The basic idea of the SVM is to find the decision hyperplane that has maximum *margin*: the distance of the closest point to the hyperplane (as shown in Fig. 3). The general form of the decision function for SVM is given by [25]

$$y(\mathbf{x}) = \sum_{n=1}^{N} \alpha^n t^n k(\mathbf{x}, \mathbf{x}^n) + b \tag{6}$$

with constraints $\sum_{n=1}^{N} \alpha^n t^n = 0$ and $0 \leq \alpha^n \leq A$, where b is a threshold, the α^i are Lagrange multipliers introduced in a *constrained optimisation problem*, and A is a constant to determine the trade-off between minimising the training error and maximizing the margin. In equation (6), $k(\mathbf{x}, \mathbf{x}^n)$ is a kernel function, which defines a similarity measure for $\mathbf{x}$ and $\mathbf{x}^n$. The effect of using a kernel function is to implicitly map the data points into a higher-dimensional feature space, and to take the inner-product in that feature space. The potential benefit of using a kernel function is that the data is more likely to be linearly separable in the feature space, and also the actual mapping to the higher-dimensional space is never needed. During the training, only a few α^n are non-zero. Patterns with α^n having non-zero values are

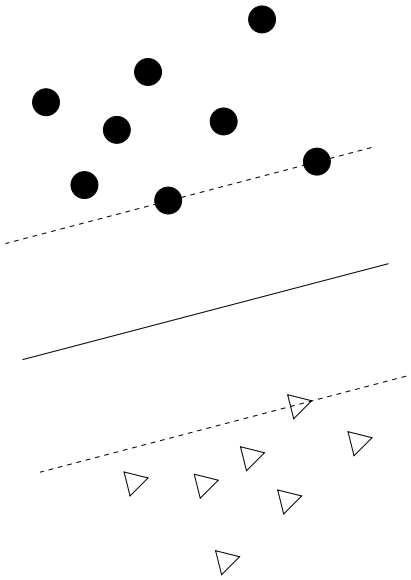

Fig. 3 A binary classification toy problem: separate dots from triangles. The solid line shows the optimal hyperplane. The dashed lines parallel to the solid one show how much one can move the decision hyperplane without misclassification of the data. The patterns on the dotted lines are the support vectors.

called *support vectors*. In our experiments, we used a Gaussian kernel function.

5.4 The Adaboost Algorithm

The Adaboost algorithm [17] produces a sequence of weak classifiers that collectively form a strong classifier. The algorithm begins with a weak classifier. The data points that are poorly classified, then have their frequency increased and the new dataset is used to train a second weak classifier. This process continuous for a specified number of iterations. The final strong classifier is a linear combination of the weak classifiers. Here the weak classifier used is a single layer neural network. A description of the Adaboost process [17] is as follows:

- Inputs: data points $\{\mathbf{x}^n, t^n\}_{n=1,\ldots,N}$, where $t^n \in \{-1, +1\}$, number of iterations M;
- Initialise weights: $d_n^1 = 1/N$;
- Do for $m = 1, \ldots, M$,
  - Train classifier on the training set with respect to $\{d_1^m, \ldots, d_N^m\}$, and obtain an hypothesis h_m: $\mathbf{x} \to \{-1, +1\}$;
  - Calculate the weighted training error ϵ_m of h_m;

$$\epsilon_m = \sum_{n:t^n \neq h_m(\mathbf{x}^n)} d_n^m,$$

  - Set:

$$\alpha_m = \frac{1}{2}\log\frac{1-\epsilon_m}{\epsilon_m}$$

– Update weights:

$$d_n^{(m+1)} = \frac{d_n^m \exp\{-\alpha_m y_n h_m(\mathbf{x}^n)\}}{Z_m}$$

where Z_m is a normalization constant such that $\sum_{n=1}^{N} d_n^{(m+1)} = 1$.

– Terminate if $\epsilon_m = 0$ or $\epsilon \geq \frac{1}{2}$ and set $M = m - 1$;

– The final strong classifier is then:

$$h(\mathbf{x}) = \sum_{m=1}^{M} \frac{\alpha_m}{\sum_{t=1}^{M}\alpha_t} h_m(\mathbf{x})$$

The Adaboost algorithm has a number of interesting properties. In [17], it was shown that the training error of the strong classifier approaches zero exponentially in the number of iterations. In [24], it was suggested that the generalisation performance of the Adaboost algorithm is related to the margin (separation) of the examples, and that it can rapidly achieve a large margin.

5.5 Majority Voting and Weighted Voting

In majority voting, each base algorithm contributes a single vote for its class. the majority class wins. While in weighted majority voting, each base algorithm votes with its *confidence*, which is measured by the probability that the given pattern ($\mathbf{I}$) is positive ($\mathbf{P}$), computed from the training set with single inputs as follows:

$$p(\mathbf{P}|\mathbf{I}) \approx \frac{\text{The number of the true positive examples}}{\text{The number of all the positive predictions}}. \quad (7)$$

Table 3 A confusion matrix: where TN is the number of true negative samples; FP is false positive samples; FN is false negative samples; TP is true positive samples.

TN	FP
FN	TP

Let W_+ denote the summed confidence of the base algorithms that give a positive prediction, W_- the negative. If $W_+ > W_-$, then the data is predicted as a part of a binding site; otherwise, as a non-binding site.

6 Classifier Performance

It is apparent that for a problem domain with an imbalanced dataset, classification accuracy rate is not sufficient as a standard performance measure. To evaluate classifiers used in this work, we apply *Receiver Operating Characteristics* (ROC) analysis [13] and several common performance metrics, such as *Recall*, *Precision* and *F-score* [9,20], which are calculated in order to quantify the performance of the classification algorithm on the minority class. The *Correlation Coefficient* as described in [32], is also defined. This gives a measure of correlation of predicted binding sites and known binding sites.

6.1 Performance metrics

Based on the confusion matrix (see Table 3) computed from the test results, several common performance metrics can be defined as follows:

$$\text{Recall} = \frac{\text{TP}}{(\text{TP} + \text{FN})}, \tag{8}$$

$$\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})}, \tag{9}$$

$$\text{F-Score} = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall+Precision}}, \tag{10}$$

$$\text{FP\_Rate} = \frac{\text{FP}}{\text{FP+TN}}. \tag{11}$$

Furthermore the Correlation Coefficient (CC), is given below:

$$\text{CC} = \frac{\text{TP} \cdot \text{TN} - \text{FN} \cdot \text{FP}}{\sqrt{(\text{TP} + \text{FN})(\text{TN+FP})(\text{TP+FP})(\text{TN+FN})}}, \tag{12}$$

In the context of identifying binding sites a high Precision and low FP_Rate are particularly important, as a higher cost is associated with a degradation of performance on these metrics. Where there is a trade-off between Precision and Recall integration of the metrics using the F-Score provides a single metric for evaluating overall performance.

6.2 ROC curves

ROC analysis has been used in the field of signal detection for a long time. Recently, it has also been employed in the machine learning and data mining domains. Here we follow [13] to give a basic idea of ROC curves.

In a ROC diagram, the *true positive rate* (also called Recall) is plotted on the Y axis and the *false positive rate* (FP_Rate) is plotted on the X axis. Points in the top left of the diagram therefore have a high TP rate and a low FP rate, and so represent good classifiers. The classifiers used here all produce a real valued output, that can be considered as a class membership probability. It is normal when using a ROC diagram to compare classifiers, to generate a set of points in ROC space by varying the threshold used to determine class membership. In this way a ROC curve corresponding to the performance of a single classifier, but with a varying threshold, is produced. One classifier is clearly better only when it dominates another over the entire performance space [13]. One attractive property of ROC curves is that they are insensitive to changes in class distribution, which makes them useful for analysing performance of classifiers using imbalanced datasets.

As noted for a ROC curve to be generated a real valued classifier is needed. The original SVM is a binary classifier. However, as described in [35] it is possible for the SVM to generate real valued outputs . For majority voting and weighted majority voting, we adopt methods proposed in [14]. The score assigned to each pattern is the fraction of votes won by the majority in majority voting. In weighted majority voting the class with the highest summed confidence wins, and the score is the average confidence. For the neural network classifiers a real valued output is automatically generated.

Often to measure the performance of a classifier, it is convenient to use a single metric and the area under a ROC curve (AUC) can be used for this purpose. Its value ranges from 0 to 1. An effective classifier should have an AUC of more than 0.5.

7 Experiments and Results

7.1 Experiments

The SVM experiments were completed using LIBSVM, which is available from the URL `http://www.csie.ntu.edu.tw/~cjlin/libsvm`. The C4.5-Rules experiments were undertaken using C4.5 software from [22]. C4.5-Rules is a companion program to C4.5. It creates rules sets by post-processing decision trees generated using the C4.5 algorithm first. The SLN was implemented using the NETLAB toolbox, which is available from the URL `http://www.ncrg.aston.ac.uk/netlab/`.

7.1.1 Parameter Settings Since we do not have enough data to build up an independent validation set to evaluate the model, all the user-chosen parameters are obtained using cross-validation. There are two training sets (single or windowed), and for each of these sets, and each classifier, the following cross validation procedure is carried out. The training set is divided into 5 equal subsets, one of which is to be a validation set, and there are therefore 5 possible such sets. For each classifier a range of reasonable parameter settings are selected. Each parameter setting is validated on each of the five validation sets, having previously been trained on the other 4/5 of the training data. The mean performance, as measured by the AUC metric over these 5 validations, is taken as the overall performance of the classifier with this parameter setting. The parameter setting with best performance is then used with this classifier and the corresponding data set (single or windowed) in the subsequent experiments. For example the SVM has two parameters and six different combinations were evaluated.

There are several approaches to generate an averaging ROC curve from different test sets [13]. In this paper, average ROC curves of the cross-validation results are obtained by first generating a ROC curve for each of the validation sets, and then by calculating the average scores from them.

The standard deviation of the AUC can therefore be attained using the cross-validation method. When only a single curve is available, the standard error can be approximated as follows [15],

$$se = \sqrt{\frac{A(1-A) + (N_p - 1)(Q_1 - A^2) + (N_n - 1)(Q_2 - A^2)}{N_n N_p}}, \tag{13}$$

where A denotes AUC, N_n and N_p are the number of negative and positive examples respectively, and

$$Q_1 = \frac{A}{2 - A},$$
$$Q_2 = \frac{2A^2}{1 + A}.$$

7.2 Results

7.2.1 Cross validation In this experiment, we trained and tested the classifiers using 5-fold cross-validation as described above. The best set of parameters for each classifier were selected and the resulting AUC value (averaged over the 5-fold validation) is shown in Table 4. Table 4 also shows standard deviations computed using cross-validation. For both single and windowed inputs,

the C4.5-Rules have the best performance. In addition, due to the different size of the training sets (see Table 2), almost all classifiers have smaller standard deviations with windowed inputs than single inputs.

Table 4 Cross validation with different classifiers.

Input	Classifier	Mean of AUC	Std
Single	SLN	74.41	2.04
	SVM	78.36	1.8
	C4.5-Rules	86.55	1.21
Windowed	SLN	75.94	0.59
	SVM	75.14	0.31
	C4.5-Rules	87.01	1.24

7.2.2 SLN results for the original imbalanced data with no sampling The results of the SLN trained with consistent, but imbalanced, data is shown in Table 5. The SVM and C4.5 algorithms have similar performance. It can be seen that the F-Score is very poor (particularly when compared to the F-Score for the best algorithm shown in the first rows of Tables 6 and 7). It suggests that the classifiers over predict the majority class. Thus, it is necessary to apply sampling methods in this work. Results of the SLN, SVM, C4.5-Rules and Adaboost shown in the following sections are obtained with sampled inputs.

7.2.3 Classification results on the fixed consistent test set with single and restricted windowed inputs This test set has 5966 data points (see section 3) in both the single and restricted windowed versions.

The results of performance metrics are shown in Table 6, together with the best base algorithm (the one with the highest F-Score). Compared with the best base algorithm, all classifiers with both single and windowed inputs, except MV and WMV decrease the FP_Rate and increase the Precision. It can be seen that the Adaboost algorithm with single inputs gives the best Precision and F-Score and the lowest FP_Rate. It improves the Precision by 55.6%, the F-Score by 17.8% and decreases the FP_Rate by 57.4% when compared with the best base algorithm. However this is at a cost: in comparison to the best base algorithm the Recall has been decreased. The classifier has become more conservative, predicting binding sites less often but with greater accuracy. When comparing the single and windowed results the only major difference is that C4.5-Rules does a lot better with single input data.

Figs. 4 and 5 show the correlation coefficient (CC) and the F-Score of each classifier with single and windowed inputs respectively. Fig. 4 shows that the Adaboost algorithm and WMV have similar performance on the CC, though WMV gives a slightly better value. It also shows that the Adaboost algorithm has the highest F-Score. Fig. 5 shows that the Adaboost algorithm outperforms all the other classifiers on both measures.

Fig. 6 shows ROC curves obtained with single inputs. The curves show that the Adaboost algorithm, SVM and

Table 5 Performance (%) of an SLN on the consistent and full test sets for the original imbalanced data.

Test set	Recall	Precision	F-Score	FP_Rate	CC
Consistent	0.95	50.00	1.87	0.07	6.15
Full	0.70	41.67	1.37	0.07	4.74

Table 6 Performance (%) on the single and restricted windowed test sets.

Input	Classifier	Recall	Precision	F-Score	FP_Rate	CC
Single	best Alg.	40.95	17.46	24.48	14.66	18.12
	MV	43.10	13.14	20.14	21.57	17.21
	WMV	41.19	17.35	24.42	14.86	23.25
	SLN	28.81	22.16	25.05	7.66	18.76
	SVM	32.14	24.46	27.78	7.52	21.74
	C4.5-Rules	29.29	23.08	25.81	7.39	19.64
	Adaboost	30.71	27.16	**28.83**	6.24	23.13
Restricted windowed	SLN	34.29	18.87	24.34	11.16	17.71
	SVM	38.81	20.25	26.61	11.58	20.25
	C4.5-Rules	23.57	18.64	20.82	7.79	19.64
	Adaboost	37.38	22.66	**28.21**	9.66	22.13

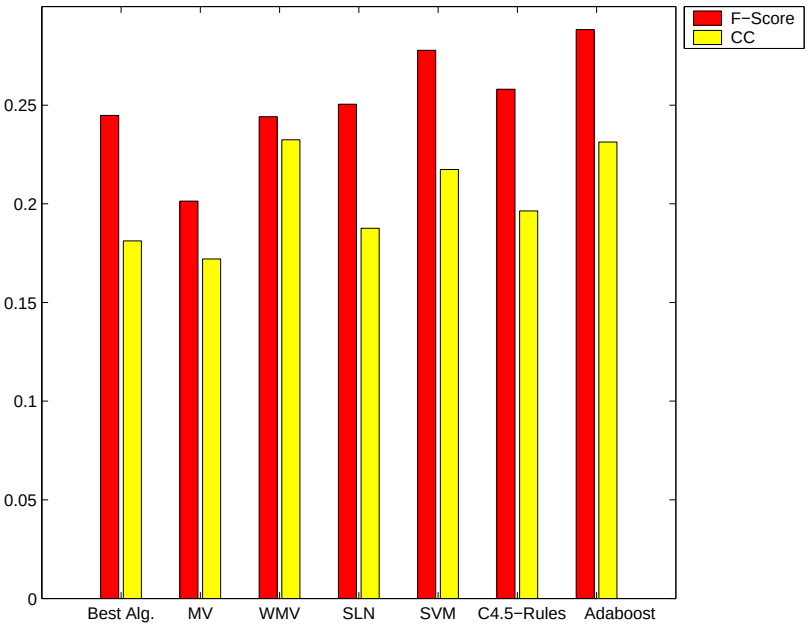

Fig. 4 Bar graph: statistics comparing the accuracy of different classifiers on consistent dataset with single inputs.

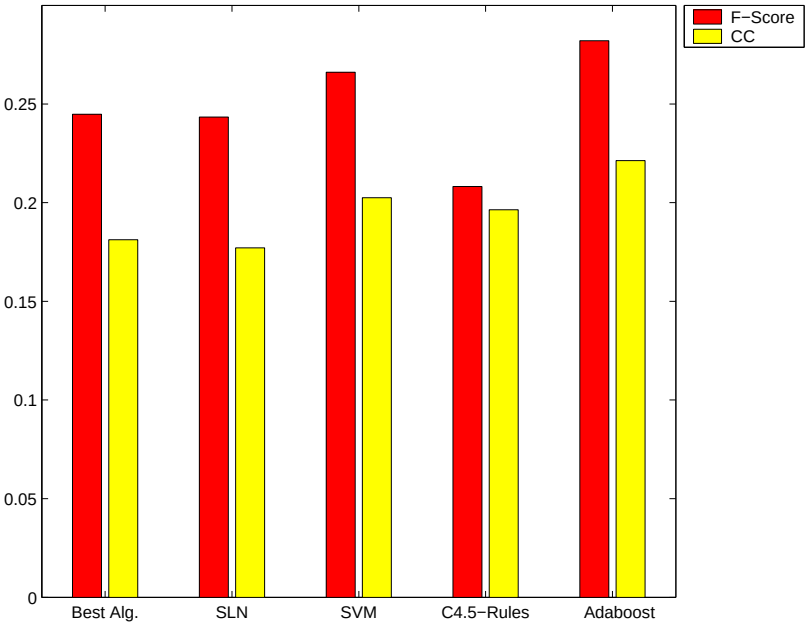

Fig. 5 Bar graph: statistics comparing the accuracy of different classifiers on consistent dataset with windowed inputs.

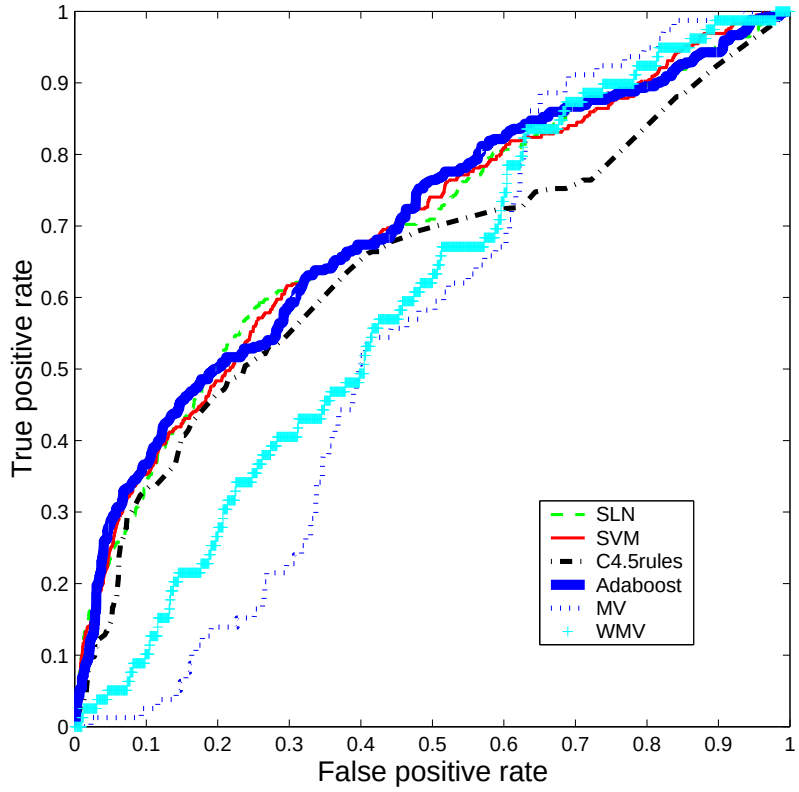

Fig. 6 ROC graph: five classifiers applied to the consistent test set with single inputs.

SLN have similar performance, outperforming the others over most of performance space. MV and WMV are the weakest. When the FP_Rate is greater than about 0.65, MV and WMV have a little higher Recall than the others. Figs. 7 and 8 are the AUC values of each classifier with error bars specified by the *se* (Equation 13). On single inputs (Fig. 7) SLN, SVM and Adaboost are similar, but do better than the other algorithms. Fig. 8 shows the SVM is significantly better than all the others on the windowed inputs.

In summary, the Adaboost algorithm performs well on the AUC measure and outperforms all the other classifiers on the biologically important values of Precision, FP_Rate and F-Score.

7.2.4 Classification results on the full test set with single and windowed inputs In this experiment, we use the full test set. All the results are presented in Table 7, Figs. 9 and 10. One of the corresponding ROC curves and AUC with error bars are shown in Figs. 11, 12 and 13.

Looking at the results for single inputs, the SVM performs well on the F-Score. The Recall values of the SLN, SVM, C4.5-Rules and the Adaboost algorithm are lower than the best base algorithm. This is explained by their far lower FP_Rate and higher Precision, in particular the Adaboost algorithm reduces the FP_Rate from the best base algorithm by 61.5% and increases Precision by 52.4%. With windowed inputs the story is very much the same. In fact the windowed SVM is the overall best performer on F-Score across single and windowed classifiers, even improving on the Recall of the best base algorithm. The C4.5-Rules perform particularly poorly, as is shown in Fig. 11, where over some of the range it is predicting below random. Comparing the SVM and the Adaboost algorithm with the best base algorithm, the SVM improves F-score by 17.3%, improves the Precision by 27.7% and decreases FP _Rate by 25.7%, while the Adaboost algorithm improves F-Score by 13.8%, improves the Precision by 35.9% and decreases by FP_Rate 41.7%, both with windowed inputs.

For the other measures, Figs. 9 and 10 show that the Adaboost algorithm has a similar performance to the SVM, though the SVM gives slightly better values. When considering the AUC metric, the SLN performs well with single inputs, while the SVM does well with windowed inputs.

8 Visualisation of Results

To aid in the biological interpretation of our results a well known visualisation tool is used. This tool is the Apollo Genome Annotation Browser [39]. It allows us to see the predictions made by the algorithms in relation to the "known" binding sites, in situ, on the genome fragments under consideration.

Figs. 14 and 15 show an alignment of various features along the DNA promoter sequences for the genes SIS1 and PHO84. In each figure, a) represents results obtained with single inputs and b) represents results ob-

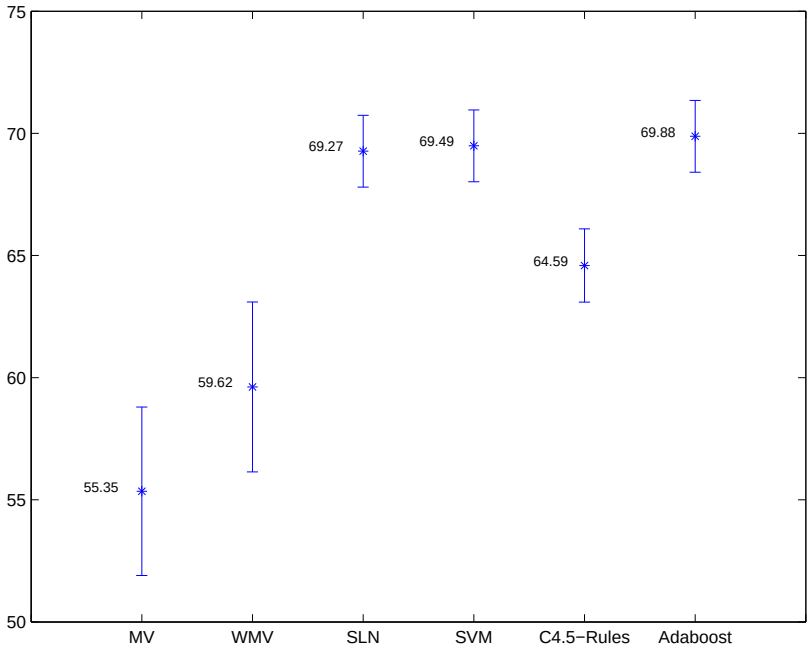

Fig. 7 The graph of AUC (%) of each classifier on the consistent possible binding sites with single inputs, with error bars specified by the standard error (*se*).

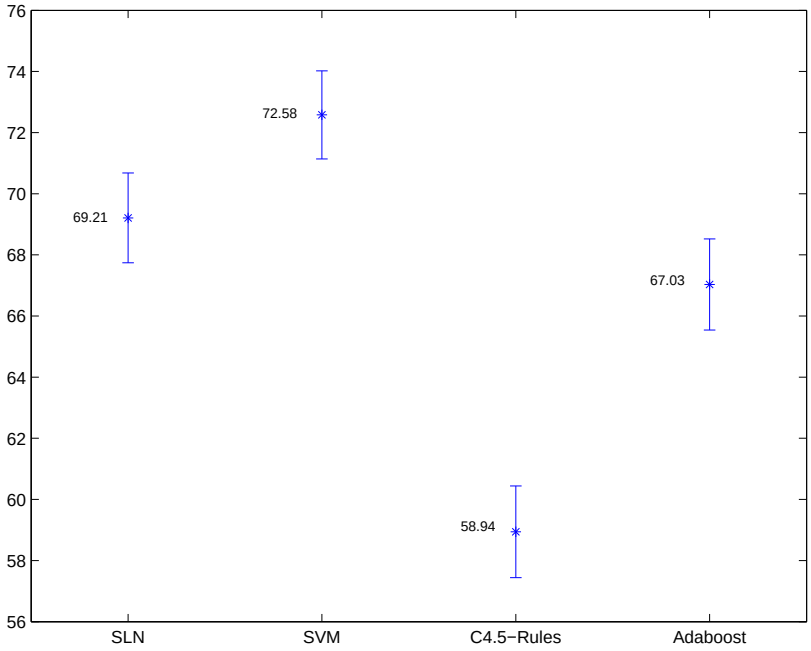

Fig. 8 The graph of AUC (%) of each classifier on the consistent possible binding sites with windowed inputs, with error bars specified by the standard error (*se*).

Table 7 Performance (%) on the full test set with single and windowed inputs.

Input	Classifier	Recall	Precision	F-Score	FP_Rate	CC
Single	best Alg.	36.36	18.40	24.44	10.73	18.86
	MV	35.73	15.12	21.25	13.35	15.27
	WMV	34.75	20.04	25.42	9.23	19.87
	SLN	25.19	25.09	25.14	5.01	20.15
	SVM	27.91	26.97	**27.43**	5.03	22.52
	C4.5-Rules	23.03	23.14	23.08	5.09	17.98
	Adaboost	24.21	28.05	25.99	4.13	21.52
Windowed	SLN	31.82	22.66	26.47	7.23	21.04
	SVM	36.78	23.50	**28.67**	7.97	23.67
	C4.5-Rules	22.26	19.70	20.90	6.04	17.98
	Adaboost	31.33	25.00	27.81	6.26	22.59

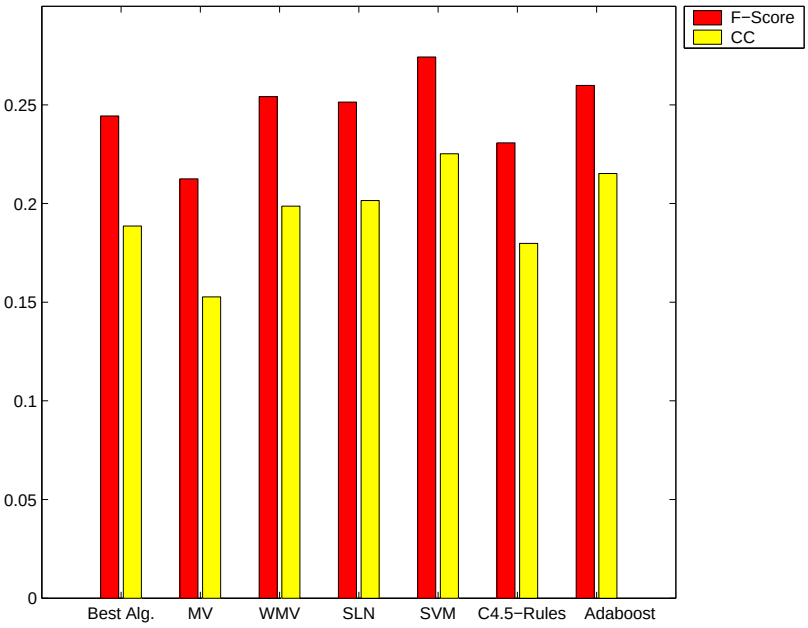

Fig. 9 Bar graph: statistics comparing the accuracy of different classifiers on the full test dataset with single inputs.

tained from the windowed inputs. The scale bar indicates size and distribution of elements in basepairs. There are two Fuzznuc predictions, one above and one below the scale, which represent predictions made on both the forward and reverse DNA strands.

These figures should not be considered representative for the performance on all genes, which is typi-

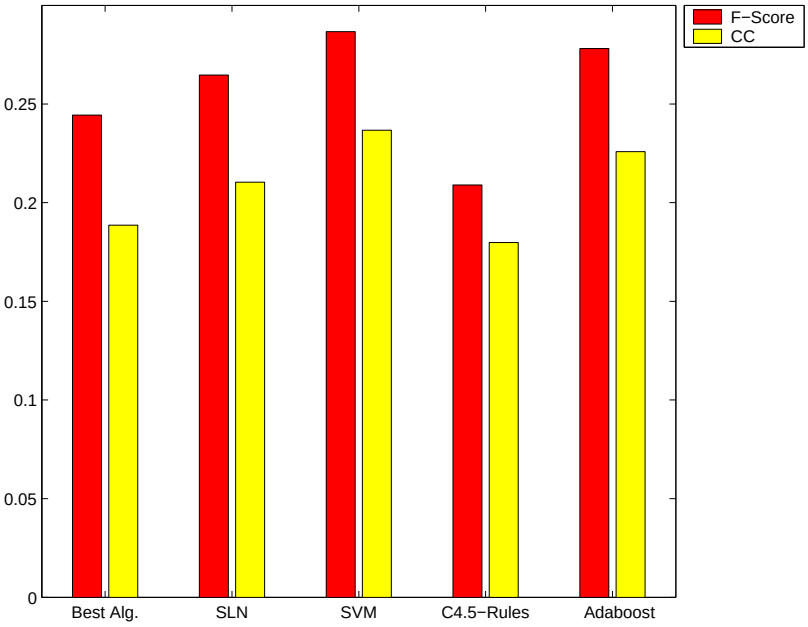

Fig. 10 Bar graph: statistics comparing the accuracy of different classifiers on the full test dataset with windowed inputs.

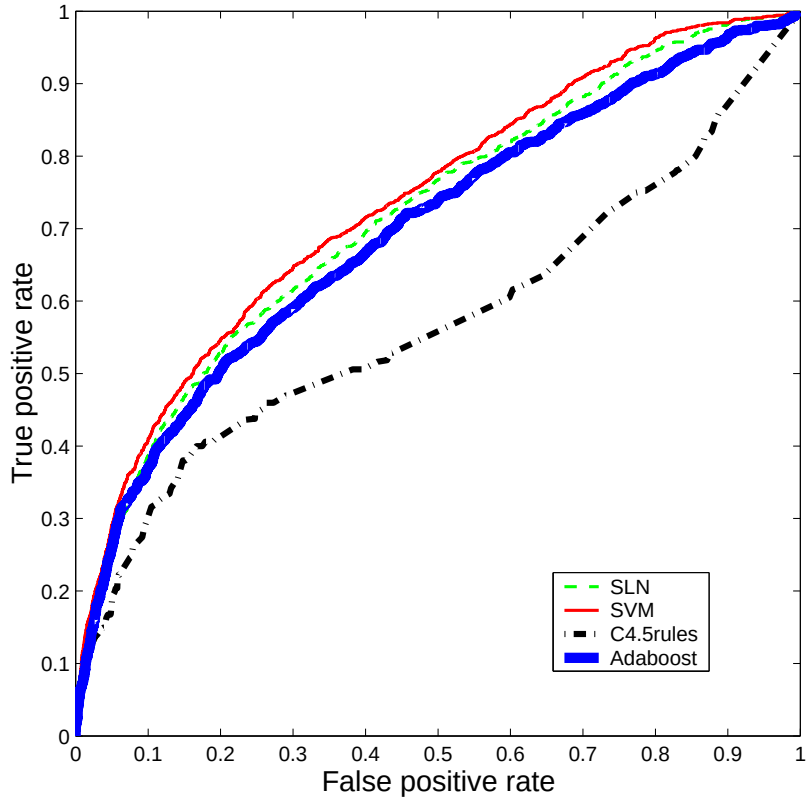

Fig. 11 ROC graph: three classifiers applied to the full test set using windowed inputs.

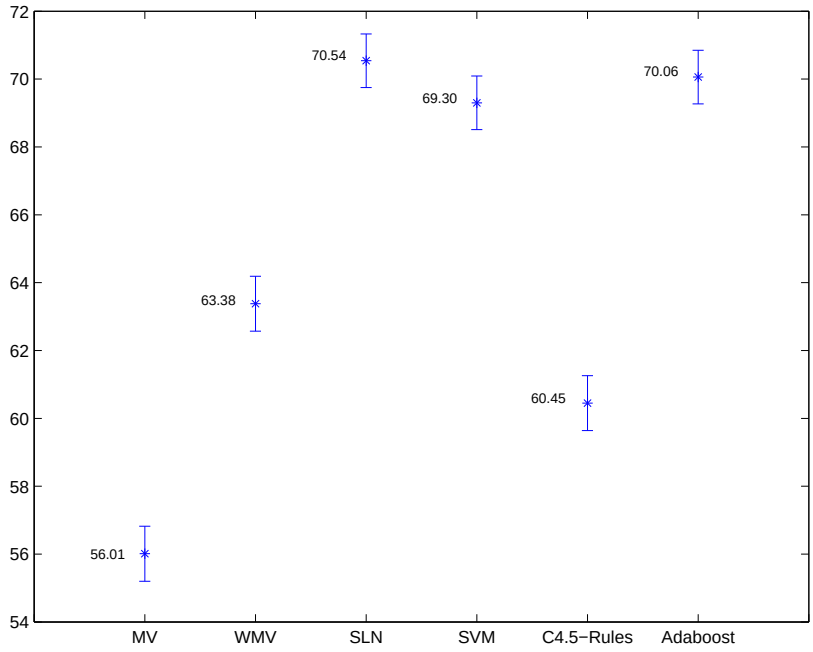

Fig. 12 The graph of AUC (%) of each classifier on the full possible binding sites with single inputs, with error bars specified by the standard error (se).

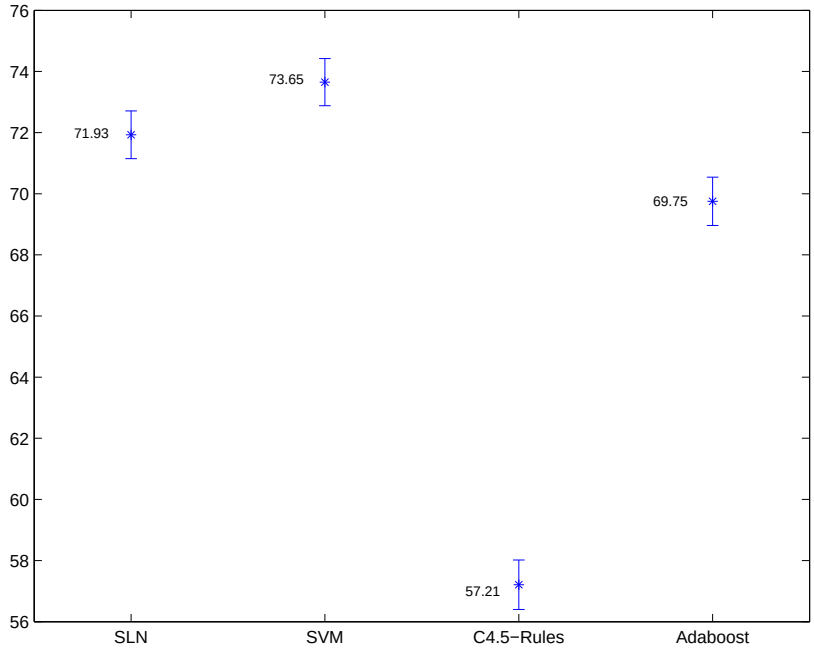

Fig. 13 The graph of AUC (%) of each classifier on full possible binding sites with windowed inputs, with error bars specified by the standard error (se).

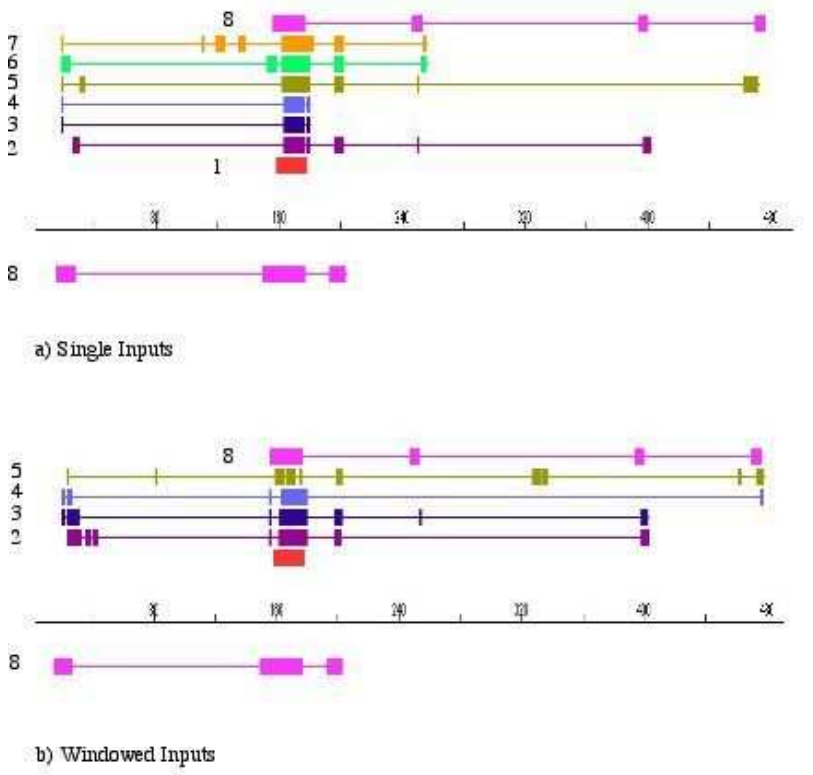

Fig. 14 Visualisation for the gene SIS1:(1) - Known Sites;(2) - Adaboost; (3) - SVM;(4) - SLN; (5) - C4.5; (6) - WMV; (7) - MV; (8) - Best Algorithm (fuzznuc). Visualisations were created using the Apollo Genome Annotation Browser.

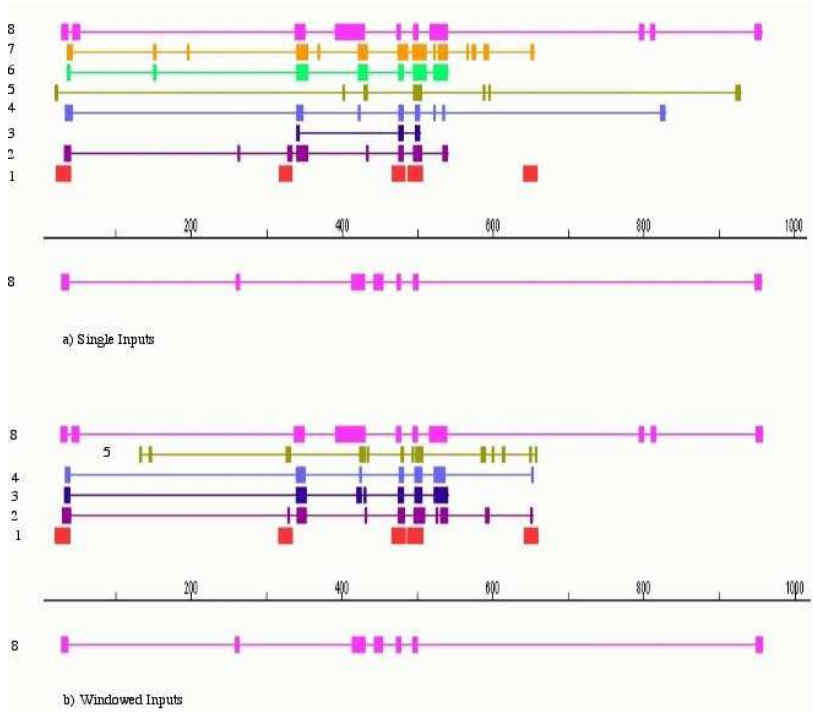

Fig. 15 Visualisation for the gene PHO84:(1) - Known Sites;(2) - Adaboost; (3) - SVM;(4) - SLN; (5) - C4.5; (6) - WMV; (7) - MV; (8) - Best Algorithm (fuzznuc). Visualisations were created using the Apollo Genome Annotation Browser.

cally highly variable from one promoter sequence to another, however they illustrate some simple examples. The "known" binding sites are shown in red (1) and represent the best available information for the location of *in vivo* functional cis-regulatory elements. The remaining features represent alignments of the respective predictions from the best original algorithm and the classification algorithms used in this study, one per line.

It can be seen from the visualisations that in all examples the SVM is the most conservative predictor, closely followed by the SLN and the Adaboost algorithm. This may be an explanation for the fact that for SIS1, which has only one annotated binding site, the SVM is the best performing algorithm, while for PHO84, which has five annotated binding sites, the Adaboost algorithm outperforms the other algorithms. There also seems to be a clear advantage, predominantly in terms of increased recall, for windowing in the example of PHO84 which does not seem to be shown for SIS1. However, it is dangerous to rely too heavily on the completeness of the annotated data, there being no way, currently, to assure that this is the case. These kinds of questions can only be answered conclusively by experimental validation of algorithm predictions. Our choice of *S.cerevisiae* for our model being largely influenced by the higher confidence generally associated with its state of genomic annotation. It should also be noted that it is frequently the case that some post-processing should be done on the predictions, for instance, to eliminate predicted sites that do not have sufficient length to be considered as real binding sites.

9 Conclusions

The most significant result presented here is that by integrating the 12 algorithms and subsequently using an SVM or the Adaboost algorithm to classify the data, results in a considerable improvement in binding site prediction. In fact when considering the full test set with windowed inputs, our best result was using the Adaboost algorithm, which improves F-Score by 13.8%, improves the Precision by 35.9% and decreases FP_Rate by 41.7% when compared to the best base algorithm. As expected the SVM gave a better classification result than the SLN and the decision trees. Majority voting was actually worse than the best individual algorithm. However, weighted majority voting was a little better. C4.5 has a tendency to badly overfit the training data and produce very poor predictions, sometimes worse than random.

By windowing the data we were able to significantly reduce the number of inconsistent data points. In fact only 7% of the data is lost when windowing as against 72% otherwise. In recent work [27], we investigate the classification results on the windowed dataset when using different feature selection filtering methods. Interestingly, it is shown that the worst performing algorithms were not detrimental to the overall performance of the meta-classifier.

Recently, we have applied our method to the much more complicated multicellular eukaryotic genome of the mouse [29]. Again, the results showed that integrating individual evidences/predictions by a meta-classifier can improve classification performance. It should be noted that these results rely critically on the use of sampling for the imbalanced data. Without sampling, as shown in Table 5, we obtained very poor results (see [28] for a further exposition).

Future work will investigate i) searching for a method to find out a suitable ratio of minority to majority classes, which could give better results; ii) building up a robust classifier based on our observation to Fig. 6, where the SVM and the Adaboost algorithm can be combined with MV and generate a ROC curve dominating the whole ROC space; iii) further investigate the biological significance of our results.

References

1. I. Abnizova, A. Rust, M. Robinson, G. te Boekhorst, and W. Gilks, (2006) Using short memory Markov models to detect regulatory elements. *Journal of Bioinformatics and Computational Biology* (In press).
2. I. Abnizova, R. te Boekhorst, C. Walter and W. Gilks, "Some statistical properties of regulatory DNA sequences and their use in predicting regulatory regions in Drosophila genome: the 'Fluffy Tail Test"'. Accepted for publication in *BMC Bioinformatics*.
3. A. Apostolico, M. E, Bock, S. Lonardi and X. Xu, "Efficient Detection of Unusual Words," *Journal of Computational Biology*, Vol.7, No.1/2, 2000.
4. M. I. Arnone and E. H. Davidson, "The hardwiring of development: Organization and function of genomic regulatory systems", Development 124, 1851-1864, 1997.
5. T. L. Bailey and C. Elkan, "Fitting a mixture model by expectation maximization to discover motifs in biopolymers," *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, 28-36, AAAI Press, 1994.
6. C. M.Bishop. (1995) *Neural Networks for Pattern Recognition*. Oxford University Press, New York.
7. M. Blanchette and M. Tompa, " FootPrinter: a program designed for phylogenetic footprinting," *Nucleic Acids Research*, Vol. 31, No. 13, 3840-3842, 2003.
8. C.T. Brown, "New computational approaches for analysisof cis-regulatory networks," *Dev Biol*, 246(1), 86-102, 2002.
9. M. Buckland and F. Gey, "The relationship between Recall and Precision," *Journal of the American Society for Information Science*, Vol. 45, No. 1, pp. 12–19, 1994.
10. P. Bucher, "Weight matrix descriptions of four eukaryotic RNA polymerase II promotor elements derived from 502 unrelated promotor sequences." *J. Mol, Biol.* 212: 563-578.
11. N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling Technique," *Journal of Artificial Intelligence Research*. Vol. 16, pp. 321-357, 2002.
12. F. H. Crick, "The genetic code", Sci Am 207, pp. 66-74.

13. R. Fawcett, "ROC graphs: notes and practical considerations for researchers," Kluwer Academic publishers, 2004.

14. T. Fawcett, "Using rule sets to maximize ROC performance," *Proceedings of the IEEE International Conference on Data Mining (ICDM-2001)*, Los Alamitos, CA, pp 131-138, IEEE Computer Society, 2001.

15. J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve," *Radiology*, 143, 29-36, 1982.

16. J. D. Hughes, P. W. Estep, S. Tavazoie and G. M. Church, "Computational identification of cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae," *Journal of Molecular Biology*, Mar 10;**296**(5):1205-1214, 2000.

17. Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, 55(**1**):119-139, 1997.

18. Wu, G and Chang, E. Y.: Class-boundary alignment for imbalanced dataset learning. *Workshop on learning from imbalanced datasets, II, ICML*, Washington DC, 2003.

19. N. Japkowicz, "Class imbalances: Are we focusing on the right issure?" *Workshop on learning from imbalanced datasets, II, ICML*, Washington DC, 2003.

20. M. Joshi, V. Kumar and R. Agarwal, "Evaluating Boosting algorithms to classify rare classes: Comparison and improvements," *First IEEE International Conference on Data Mining*, San Jose, CA, 2001.

21. M. Markstein, A. Stathopoulos, V. Markstein, P. Markstein, N. Harafuji, D. Keys, B. Lee, P. Richardson, D. Rokshar and M. Levine, "Decoding Noncoding Regulatory DNAs in Metazoan Genomes", *proceeding of 1st IEEE Computer Society Bioinformatics Conference (CSB 2002)*, Stanford, CA, USA, 14-16, August, 2002.

22. J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kauffman, 1993.

23. N. Rajewsky, M. Vergassola, U. Gaul and E. D. Siggia, "Computational detection of genomic cis regulatory modules, applied to body patterning in the early Drosophila embryo," BMC *Bioinformatics*, **3**:30, 2002.

24. R. E. Schapire, Y. Freund, P. L. Bartlett, and W. S. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods". *the Annals of Statistics,* 26(5):1651-1686, October 1998.

25. B. Scholköpf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, The MIT Press, 2002.

26. Y. Sun, M. Robinson, R. Adams, P. Kayes, A. G. Rust and N.Davey, "Integrating binding site predictions using meta classification methods", *Proceedings ICANNGA05*, 2005.

27. Y. Sun, M. Robinson, R. te Boekhorst, R. Adams, , A. G. Rust and N.Davey, "Using feature selection filtering metohds for binding site predictions", *submitted to The 5th IEEE International Conference on Cognitive Informatics, ICCI05*, Beijing, 2006.

28. Y. Sun, M. Robinson, R. te Boekhorst, R. Adams, , A. G. Rust and N.Davey, "Using sampling methods to improve binding site predictions", *accepted by the 14 th European Symposium on Artificial Neural Networks, ESANN*, Bruges, 2006.

29. Y. Sun, M. Robinson, R. Adams, A. Rust and N. Davey,"Prediction of Binding Sites in the Mouse Genome using Support Vector Machine", Proceedings of

18th International Conference on Artificial Neural Networks (ICANN2008), Prague, September 2008, Editors V. Kurkova, R. Neruda, J. Koutnik, Springer Part 2 (LNCS 5164), pp 91-100.

30. R. te Boekhorst, I. Abnizova, and C. L. Nehaniv, "An adaptive sliding windopw algorithm for inferring DNA functionality from sequence information", submitted to *Applied Bioinformatics*, 2004.
31. G. Thijs, K. Marchal, M. Lescot, S. Rombauts, B. De Moor, RouzP and Y. Moreau, "A Gibbs Sampling method to detect over-represented motifs in upstream regions of co-expressed genes," *Proceedings Recomb'2001,* 305-312, 2001.
32. M. Tompa, *et al.*, "Assessing computational tools for the discovery of transcription factor binding sites," *Nature Biotechnology*, 23(1), January 2005.
33. R. J. White, *Gene Transcription: Mechanisms and Control*, Blackwell, 2001.
34. T. G. Wolfsberg, A. E. Gabrieliam, A. E. Campbell, M. J. Cho, R. J. Spouge, and D. Landsman, "Candidatge regulatory sequence elements for cell cycle - dependent transcription in Sacharomyces cerevisiae." *Genome Research*, 9, 775-792.
35. T. F. Wu, C. J. Lin and R. C. Weng, "Probability Estimates for multi-class classification by pairwise coupling," *Journal of Machine Learning Research*, 5 pp. 975-1005, 2004.
36. *http://emboss.sourceforge.net/.*
37. *http://sourceforge.net/projects/netmotsa*
38. *http://sourceforge.net/projects/pars*
39. *http://www.fruitfly.org/annot/apollo*