NEW APPROACHES TO BASE CALLING

Ayush S., Chaitanya S., Nitish G.,
Paul M., Srikant A.

5th December, 2014

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abstract

Oxford Nanopore's MinION is a small and portable device which uses the third generation Nanopore sensing technology to enable long reads in a relatively short period of time. Neucleotides are detected based on the signature electrical discrepancies that they cause when they are passing through a Nanopore. Unfortunately, the algorithms for predicting the neucleotides given the signals(base-calling mechanism) are not very accurate and therefore not very useful in a large number of cases. In this project, we try to use other machine learning techniques to solve this problem. We model the base-calling problems in two different ways. We first use Support Vector Machines to predict 5-mers that corresponds to each signal and its neighbourhood. Secondly, we use a Naive Bayes model that tries to predict a single neucleotide using the corresponing signals and its neighbourhood(with the underlying assumption that the effects of the neucleotides are independent of each other). We show that both SVM and Naive Bayes approach gives better accuracy than random base calling even though they fail to beat Oxford Nanopore.

1 INTRODUCTION

DNA sequencing is the process of determining the precise order of nucleotides within a DNA molecule. Even though the field of DNA sequencing has improved significantly since the development of second-generation sequencing technologies, it still suffers from significant limitations. One such limitation is the small length of a read.

Third generation technologies like Oxford Nanopore's MinION solve this problem by enabling much longer reads. This is done by pulling a strand (analyte) of genome through a membrane that has an ionic current flowing through it. When the strand passes through the membrane, it creates a set of electrical disturbances known as signals(events). Nanopore then uses these signals to train a Hidden Markov Model, and uses Vitebi algorith to predicts the most probable set of bases given a signal set. The signals are the the observed states and the set of neucleotides(5-mers) are the hidden states.

The base calling technique used by Oxford Nanopore is suboptimal with a low accuracy rate($<$ 32 percent) and is not significantly useful in a number of cases. Recently Oxford Nanopore publicly released their events data for overnight runs on E Coli. This gives us enough data to train learning algorithms that can be used for base calling.

We endeavor to predict the sequence of nucleotides from the signal using two different machine learning algorithms that tends to model the task in two different ways. One is the SVM classifier and the other is Naive Bayes classifier.

2 LITERATURE SURVEY

Before starting with implementation of the project we went through some important papers. These papers describe some interesting algorithms for base calling using Nanopore signals. They also served as good introductions to Naive Bayes and SVM based learning algorithms applied in this context.

The Oxford Nanopore Base-Caller estimates the DNA sequence using a modified Viterbi Algorithm[5]. The algorithm takes a de novo approach of predicting the current base pairs based on the Hidden Markov Model. In their model, Nanopore signal at any instant is the observed data and the k-mer (which produces the signal) is the predicted state.

BayesCall[10] is a novel model-based basecalling algorithm which is flexible enough to incorporate various features of the sequencing process. In particular, it can easily incorporate time-dependent parameters and model residual effects. This new approach significantly improves the accuracy particularly in the later cycles of a sequencing run.

Naive Bayes[6], is again a model-based algorithm for high-throughput sequencing. It builds on the ideas of BayesCall to devise an efficient base-calling algorithm that is order of magnitude faster than BayesCall. It uses the same generative model and parameter estimation as BayesCall but improves upon the speed by utilizing approximation and optimization methods to achieve scalability.

Alta Cycle[8], is a base calling algorithm for next-generation sequencing. The algorithm uses SVM to compensate for noise factors. It works in two stages, training and base-calling. It learns run-specific noise patterns according to the model and finds an optimized solution that reduces the effect of the noise sources. The optimization is mainly achieved by supervised learning using a rich DNA library with a known reference genome. Alta-Cyclic then enters the base-calling stage and reports all the sequences from the run with the optimized parameters. Alta-Cyclic uses the fluorescence intensity values that are generated by Firecrest, the Illumina image-processing software. The training process is transparent, although computationally intensive.

3 IMPLEMENTATION DETAILS

Implementation of the project can be divided into following steps:

3.1 Data Extraction and processing

The genome read dataset was acquired from the whole-genome shotgun sequencing of the model organism Escherichia coli K-12 substr. MG1655 generated on a MinION device with R7 chemistry during the early-access MinION Access Program[9]. The data is available online and is about 47 GB in size. The downloaded dataset is in the fast5 format.

Initially, we randomly divide the entire E Coli data set(having close to 31500 reads/files) into three parts. The training set consists of 60 % (18900) of the data, the validation set contains 15 %(4725 files) of the data and finally the testing set contains the remaining 25 %(7875 files) of the data. Our initial plan was to use the entire data set for training, testing and validation. But due to time constraints we decided to use around 8000 reads from the training set, 2000 reads from the testing set and around 1500 reads for validation. Out of these 8000 reads in the training set, only 689 reads actually aligned. Similarly only 155 reads from the testing set aligned and only 96 reads from the validation set aligned. This severely reduced the size of our data sets.

We then run the poretools software on the fast5 files in each folder. Poretools runs directly on the native fast5 files and has a set of utility functions that can be used to extract the events data (i.e. time series of Nanopore translocation) and the corresponding fasta file (base pair sequence). The output of this step are saved in files with ".fasta" and ".events" extensions.

3.2 Alignment

The base pair sequences in the fasta files obtained in previous step are then aligned with the original genome sequence. The alignment is essential in order to extract correct data set for training. This is because Oxford Nanopore reads are not accurate and we would not want our classifier to learn just using the nanoprore reads. Thus by aligning the Nanopore reads with the original E Coli genome sequence we get the set of "correct" regions in the subject genome, which can then be used to train our algorithms.

Alignment is done using the Basic Local Alignment Search Tool(BLAST)[1]. A shell script is run through each of the fasta file taking the original E Coli genome sequence as the subject. It aligns each of the files with the original genome sequence and stores the alignment in a file with a ".out" extension. While trying to align the fasta file reads with the original genome sequence, the BLAST tool inserts gaps in our query read. Such gaps are unfavorable as they make it difficult for us to extract the corresponding feature vectors from the event files. Hence, we remove these gaps in our final aligned read. The aligned reads along with the original genome reads are then stored in a file with ".alignedfasta" extension. The aligned fasta file along with the corresponding event file is then used by our feature vector extraction script to generate the feature vectors that would be used to train and test the multiclass SVM and Naive Bayes.

3.3 Classification

Based on the assumption that in a base-calling experiment using Nanopore, each signal depends on three to five nucleotides that are passing through the Nanopore at that particular instant, there are two different ways of modelling this relationship between signals (features) and nucleotides:
a. We can either assume that each k-mer (that is currently inside the pore) affects the signal captured at that instant. Also, a part of the k-mer (and hence the k-mer itself) affects the signals before and after the current signal.
b. An alternative way to model the same is to assume that each base affects the current signal as well as the signals at its neighbourhood and these effects are independent of each other(i.e. each base independently affects the signals in a 'mutually exclusive' manner).
The above models relating the neucleotides to signals assist in deciding the input feature vectors and the behavior of our two classifiers. We use the first type of modelling in SVM and the second type in Naive Bayes. This is because after the initial implementation of the first model type in SVM, we realized that the second model type would be a better in resolving the case where two consecutive predicted k-mers have no overlap between them.

3.3.1 *Support Vector Machines*

Support Vector Machines(SVM) [2] are supervised learning algorithms that are used for data classification. A classification task usually involves separating data into training and testing sets. Each instance in the training set contains one *target value* (i.e. the class labels) and several *attributes* (i.e. the features or observed variables). The goal of SVM is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes. [3]

Given a training set of instance label pairs (x_i, y_i), $i = 1, 2, ..., l$, where $x_i \in R^n$, $y_i \in \{-1, 1\}$ and l is the number of training examples, the support vector machines(as described by Cortes and Vapnik[2]) require the solution of the following optimization problem:

$$min\ (w^T w)/2 + C \sum_{i=1}^{l} \xi_i \text{ wrt } w, b, \xi$$

$$\text{subject to: } y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i$$

$$\text{and } \xi_i \geq 0.$$

Here w is the objective function such that x_i is classified as 1 iff $wx_i + b > 0$ and it is classified as -1 iff $wx_i + b < 0$. Here $C > 0$ is the penalty parameter for the error term.

In short SVM finds a linear separable hyperplane with the maximum margin. Unfortunately in most cases the data set is not inherently linearly seperable. In such a case the training vectors x_i are mapped to a higher dimensional space(with the hope that the data set would be linearly seperable in higher dimensions). This is known as the kernel trick and it uses the clever observation that to solve our optimization problem we only need the pariwise dot products of the vectors x. A kernel is therefore a function, which given two vectors x_i and x_j, computes their dot product without explicitly mapping these vectors to a higher dimensional space. Of course such a space must exist(otherwise the kernel function makes no sense).

A Support Vector Machine is inherently a binary classifier. However, it may be extended to multiple classes [4]. Such an SVM is called a multiclass SVM.

SVM details and implementation

The feature vectors and the class label are generated in a .svm file based on the corresponding .fast5 file and .alignedfasta file(which contains the aligned sequence) in the Alignment step.

To model our SVM classifier we represent the relationship between the neucleotides and the signals based on the first type of model (mentioned at the start of Classification). That is each k-mer (that is currently inside the pore) is assumed to affect the signal captured at that and the neighbouring events. In our implementation k is assumed to be 5. Also, we assume that the 5-mer will affect the signal just before and just after the current signal. Though one can take more signals in the neighbourhood into consideration, we planned to stick to three signals. This is because we assumed that there would be a high autocorrelation between the signals that come one after the other in the time series. We also thought that we would just increase the number of features later if required.

So, let the event vector, α_t be the mean, standard deviation and the length of the t^{th} event(that is the signal at instant t). Then the feature vector f_t is the concatenation of the vectors $\alpha_{t-1}\ \alpha_t\ \alpha_{t+1}$. Hence feature vector contains 9 features. We then normalize our feature vector using the following linear transformation:

$$N(f_{ti}) = (f_{ti} - min_i)/(max_i - min_i)$$

Here f_{ti} represents the i^th feature of the vector f_t. Also, as we consider 5-mers with maximum of one gap(for simplicity) as the output. Total number of such classes of 5-mers is:

$$4^5 + 5 * 4^4 = 2304$$

(since there are 4^5 kmers with no gap and $5 * 4^4$ with exactly one gap.) Here, we choose gap as a possible base because it is entirely possible that some of our bases(in the query) aligns with gaps(in the subject) when we use the BLAST tool. But we ignore all the examples that have more than one gap(to make sure that our data set is meaningful). The number of classes in the SVM is therefore 2304, that are labelled from 0 to 2303.
We tried to train our SVM using the following kernels:

- **Linear kernel**: $K(x_i, x_j) = x_i^T x_j$
- **Polynomial kernel**: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$
- **RBF kernel**: $K(x_i, x_j) = exp(-\gamma(x_i - x_j)^2)$

We have used $SVM^{multiclass}$ which is an open source implementation of multiclass SVM for our project. $SVM^{multiclass}$ expects class labels and the feature vectors written one after the other in a space seperated file, for training and testing. [7]

3.3.2 *Naive Bayes Classifier*

In machine learning Naive Bayes Classifiers are a family of simple probabilistic classifer based on applying the Bayes theorem with strong independence assumptions among feature.

Naive Bayes details and implementation
In Naive Bayes, we find the relationship between neucleotides and signals using the second type of modelling(as described above). We assume that each base affects a signal and its neighbourhood independently and in a mutually exclusive fashion.

Our basic assumption is that each base pair that comes at time t affects the raw signals at time $t - 2$,$t - 1$,t,$t + 1$ and time $t + 2$. We also assume that a particular base pair that comes at time t in the time series, affects the signal at time t more than it affects the signals at time $t - 1$ and time $t + 1$. Similarly, we assume that the affects are lesser for time $t - 2$ and time $t + 2$. Since Naive Bayes maps inverse relations into predictions, each base pair is predicted using the current five signals. Hence there are 15 features and 4 classes (corresponding to each base A, C, T and G) that we used to train Naive Bayes.

To design a model that incorporates this basic assumption we run our algorithm in three phases. First we try to predict the base pair at time t using the signal at that time. Hence, the feature vector (f_t) is the mean, standard deviation and length of the signal at time t. Since, $f_t \in R^3$, we normalize our feature vector such that the new feature vector is $f_t \in Z_{20}^3$. This is done in order to make sure that enough training examples hit a particular vector and classification is meaningful.

Next we compute

$$Pr[f_t = (f_{t1}, f_{t2}, f_{t3}) \mid bp = i, \forall i \in B]$$

where $B = \{A, C, G, T\}$. This is done by calculating the number of occurrences of a particular base pair and the number of occurrences of a particular feature vector, given a base pair. Similarly we compute the corresponding probabilities for the other base pairs (and the features we observe). Now $Pr[bp = i, \forall i \in B \mid f_t]$ is calculated using Bayes theorem.

$$Pr[bp = i, \forall i \in B \mid f_t] = (Pr[f_t \mid bp = i, \forall i \in B] \times Pr[bp = i, \forall i \in B])/Pr[ft]$$

For the second phase we predict (the base pair at time t) using the feature vector that is the concatenation of the feature vectors f_{t-1}, f_t and f_{t+1}. And then, in our last phase we train against the concatenation of the feature vectors f_{t-2}, f_{t-1}, f_t, f_{t+1} and f_{t+1}. Now we estimate the probability that a base pair comes at time t as the average of the estimated probabilities in the three aforementioned phases. That is:

$$Pr'[bp = i, \forall i \in B \mid f_{t-2}f_{t-1}f_tf_{t+1}f_{t+2}] = 1/3 \times (Pr[bp = i, \forall i \in B \mid f_t] + Pr[bp = i, \forall i \in B \mid f_{t-1}f_tf_{t+1}] + Pr[bp = i, \forall i \in B \mid f_{t-2}f_{t-1}f_tf_{t+1}f_{t+2}])$$

The base pair with the maximum posterior probability is chosen by our algorithm. By taking the average, our technique cleverly encapsulates the assumption of giving more weightage to the event at time t, lesser weightage to the events at time $t-1$ and $t+1$ and the least weightage to the events at time $t-2$ and $t+2$.

3.4 Testing

The '.nb' and '.svm' files in the test folder created after the alignment step are used as input to the two classifiers. The SVM and the Naive Bayes classifiers are run on the input files to give the predicted output sequences. We measure the accuracy of our results in following two ways:
a. Edit Distance: To measure the accuracy of our results, we initially measure the Edit Distance between the the predicted sequence and the original sequence(Edit Distance is minimum cost of operations required to convert one string to another). Figure illustrates the working on the DP based algorithm to calculate Edit Distance.

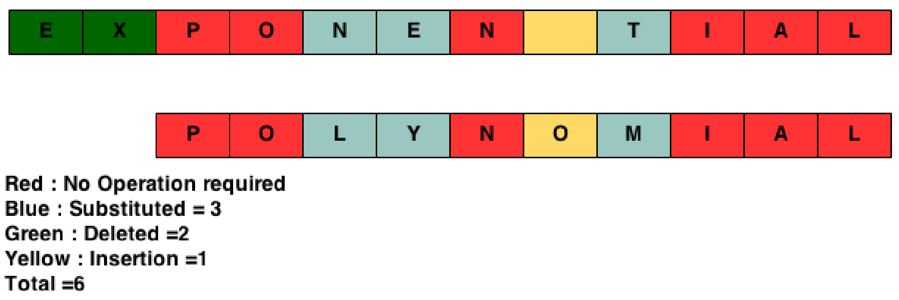

Figure 1: Edit distance illustration

For our purpose we take both the gap cost (cost of insertion, deletion) and the substitution cost as 1.Also, in our algorithm a gap(represented by an underscore) is treated as a separate character.

We choose this measure as Edit Distance is a standard measure of dissimilairty between two sequences. For same length of sequences, higher the dissimilarity between two sequence, higher is the Edit Distance between them.

After calculating the Edit Distance for each set of predicted and aligned sequence, we calculte the following parameters

- Mean Edit Distance(μ) : An average of Edit Distances calculated over all pairs of files.

$$\mu = \sum_{i=1}^{n} \frac{ED_i}{n}$$

- Root Mean Square Edit Distance (RMS) : Root of the quadratic mean of Edit Distances.

$$RMS = \frac{\sqrt{\sum_{i=1}^{n} ED_i^2}}{n}$$

b. F1 score: F1 score is a standard to measure a test's accuracy. It considers precision p, and recall, r, of the test to compute a score which is their harmonic mean. The score reaches its best value at 1 and worst value at 0.

$$F_1 = 2.\frac{precision.recall}{precision+recall}$$

where precision and recall are defined as:

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

where TP denotes true positives, FP denotes false positives, and FN denotes False Negatives.

Precision measures the probability that an item predicted to be in a particular class, indeed belongs to the same class. Recall on the other hand measures the fraction of items that belongs to a particular class and were labelled correctly.

For multiple classes, the definitions of precision and recall can be generalized, using a "confusion matrix"

Let M be the confusion matrix, whose rows correspond to the actual classes, and columns correspond to the predicted classes. Let m be the number of classes. The F1 score, precision and recall can be generalized for this matrix as follows:

$$Precision_i = \frac{M_{ii}}{\sum_{j=1}^{m} M_{ji}}$$

$$Recall_i = \frac{M_{ii}}{\sum_{j=1}^{m} M_{ij}}$$

$$F1_i = \frac{Precision_i}{Recall_i}$$

We also define average F1, precision, and recall

$$Precision_{avg} = \frac{\sum i \in MPrecision_i}{M}$$

$$Recall_{avg} = \frac{\sum i \in MRecall_i}{M}$$

$$F1_{avg} = \frac{\sum i \in MF_i}{M}$$

For a small number of classes (e.g for Naive Bayes Classificaion), we report the scores for each class. When the number of classes is large , we report the average scores.

We also use F1 scoring because it is widely used in machine learning. It has been widely used in the natural language processing literature. Also it has been used widely in computational biology field to measure the accuracy of machine learning algorithms.

c. Accuracy Percentage: We calculate a fraction, accuracy percentage, which signifies the percentage of correct results as compared to total results count. Let count of correct predictions be 'correct' and 'total' be the total predictions. Then :

$$AccuracyPercentage = \frac{100*correct}{total}$$

Clearly, accuracy percentage gives a measure of accuracy of a test as it compares the number of success scenarios relative to all possible scenarios.

4 RESULTS, OBSERVATIONS AND ANALYSIS

The genome read dataset used was from the whole-genome shotgun sequencing of the model organism Escherichia coli K-12 substr. MG1655 generated on a MinION device with R7 chemistry during the early-access MinION Access Program[9]. The data is available online.

To obtain the results, from the whole dataset, 8000 reads for training, 2000 reads for testing and 1500 reads for validation were used. Out of the 8000 reads in the training set, only around 689 reads aligned with the EColi original genome using BLAST. Similarly only 155 reads from the testing set and 96 reads from the validation set aligned.

4.1 Results and Analysis of SVM method

Our final training set for SVM contained 3,36,257 training samples. We used $SVM^{multiclass}$ as the tool for generating our models and predictions. There were 2304 classes and each feature vector contained 9 features. When we tried to validate, the results we got from different values of the parameter C were almost equivalent. Thus, we decided to run the testing set with different parameters as well because there were no real 'winning' parameter(as shown in Table1). The testing set had 67,746 testing examples.

Table 1 shows the error rate,correct and incorrect k-mer prediction using multiclass SVM. We have done it using varous values of C for linear kernel.

Table 1: Results of $SVM^{multiclass}$ for different parameters

Model	C	Correct	Incorrect
1	4	139	67607
2	0.5	234	67512
3	32	401	67345
4	0.01	234	67512
5	0.125	234	67512
6	64	274	67472

For the same testing data, Oxford Nanopore predicted 35,753 classes correctly and 31,993 classes incorrectly. Thus, at first glance it seems that Nanopore HMM model has a k-mer prediction accuracy of more than 50 percent. But these evaluations are done only for the aligned reads of Oxford Nanopore(i.e the reads that actually aligned with the E. Coli genome). Hence, the sample set is biased. Since only around $1/10th$(and sometimes even less) of the reads actually align, we can claim that the real accuracy ratio for k-mer prediction of Oxford Nanopore would be $< 5\%$. Our algorithm has the accuracy rate of $\sim 0.6\%$.

We also tried using RBF kernel and polynomial kernel for training the data set. Unfortunately, both of them failed to converge on our data set even though we ran it overnight.

We also calculated the $F1_{avg}$ scores for SVM and Oxford Nanopore, over 2304 classes,as shown in Table 2 and Table 3.

Table 2 and Table 3 shows the F1 score for Oxford Nanopore and SVM using 2304 classes.

Table 2: $F1_{avg}$ score for Oxford Nanopore

$Precision_{avg}$	$Recall_{avg}$	$F1_{avg}$
0.0784	0.0069	0.0069

Table 3: $F1_{avg}$ score for SVM

$Precision_{avg}$	$Recall_{avg}$	$F1_{avg}$
0.0005	0.00046	0.00047

As the F1 score value for Oxford Nanopore is greater than the F1 score value for the SVM, Oxford Nanopore's base-calling algorithm offers a higher accuracy. We did not find the edit distance for SVM, because our classifier performed poorly and the edit distance(if there were indeed any actual alignment) would have been too large and would not have been meaningful.

We claim that our training data set has a uniform distribution for the 5-mers that have no gaps. It also has a uniform distribution for 5-mers with exactly one gap. We just ignore 5-mers with more than one gap. Figure 2 and Figure 3 shows that distribution of labels with and without gaps is uniform. Figure 4 shows that the distibution of count vs number of 5-mers with that count is a Gaussian, further strenthening our claim.

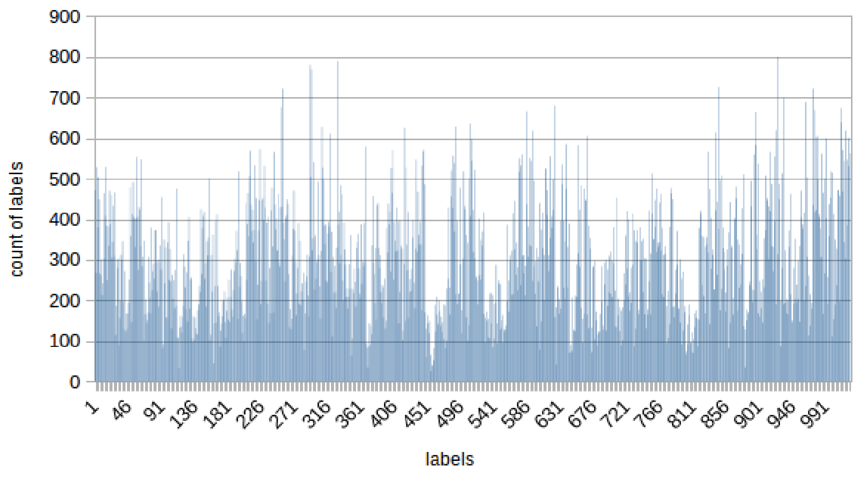

Figure 2: Label distribution for 5 mers without gaps in the training data

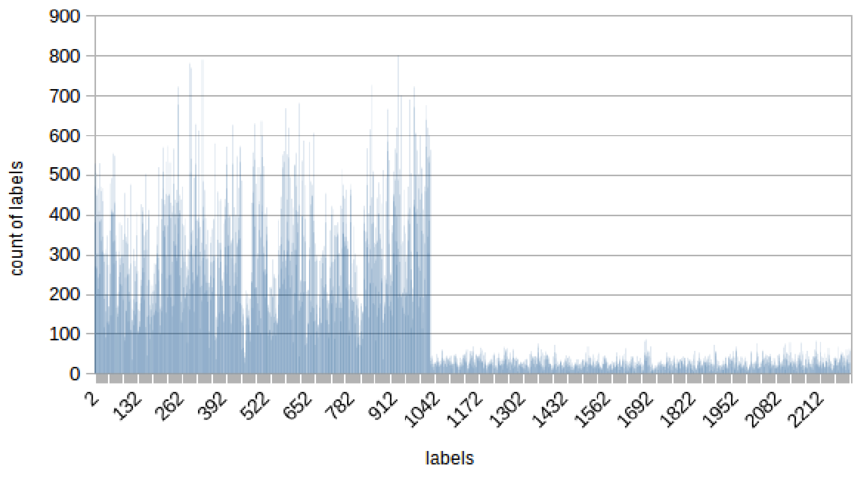

Figure 3: Label distribution for 5 mers with gaps in the training data

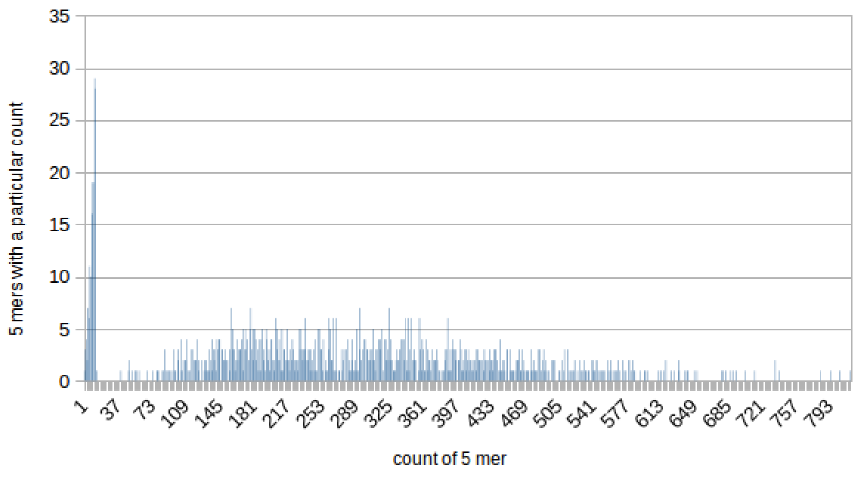

Figure 4: Counts vs no. of 5 mers with the particular count (without gaps)

Experimentally, we found out that the probability that a 5-mer has a gap is nearly equal to 0.1. So if we had a model that made a k-mer selection randomly, the accuracy rate would be nearly equal to 0.1%.

$$Pr[correctPrediction] = (0.1/1280) + (0.9/1024) \sim 0.001$$

From our results we conclude that our model performs slightly better than random selection. Still it performs significantly worse than the HMM model. The possible reasons are:

- The number of training examples is small for training a model with 2304 classes and 9 features. We used 3,36,257 samples for training this with an average number of hits of 146 per class. In reality the average number of hits is much greater for the classes without any gaps(which does not seem too bad). Nevertheless we suspect that a small number of training examples might be problem because our SVM converged(with $\epsilon = 0.5$) with only 347 support vectors and it is certain that a significant number of labels had no support vectors. We tried running the same number of samples with lower ϵ values but those runs did not converge. Our data set was sparser than we had anticipated since our initial assumption that a high fraction of the Oxford Nanopore reads align turned out to be false.
- Value of ϵ is large. We should have run the SVM with smaller ϵ values. We could not do this mainly because of the enormous running time that it required. We ran a test with a much smaller sample and $\epsilon = 0.2$ and that model had an accuracy of $\sim$ 0.8%. Thus, we were stuck in a trade-off between sample size and the value of ϵ.
- The training data is not inherently linearly seperable. We could not train our sample for higher dimensional kernels due to the time constraints(none of the overnight runs converged). This seems like the most probable explanation, primarily because the SVM predictions contained only a small fraction of the labels(i.e most of the labels were never really predicted). This hypothesis would also explain the low number of support vectors.
- It is also possible that our feature vector is too small and we should have considered more events in the neighbourhood. We assumed that events have strong auto correlation among themselves and taking too many events will give no useful information. It is possible that this assumption is incorrect.
- Our basic assumption that the current signal and its neighbourhood depends on the current 5-mer is erroneous. In this case our model has no real physical significance and it fails to accurately describe the physical event. This is the most unlikely explanation since if it were the case, Oxford Nanopore reads would have been much worse.

4.2 Results and Analysis of Naive Bayes method

Naive Bayes classifier has 4 classes and each feature vector contains 15 features. We ignore the gaps for our purposes.

Our Naive Bayes model predicts the correct class with 33.8% accuracy. We come up with this accuracy by alligning our predicted and the aligned fasta and calculating the number of correct alignments as a percentage of total possible allignments. The Naive Bayes approach does better than random picking the class. This is because the distribution of the neucleotides are random and uniform (as shown in figure 5 where the 4 bases have almost equal count) and any algorithm that randomly picks up a class would have an accuracy rate of 25%.

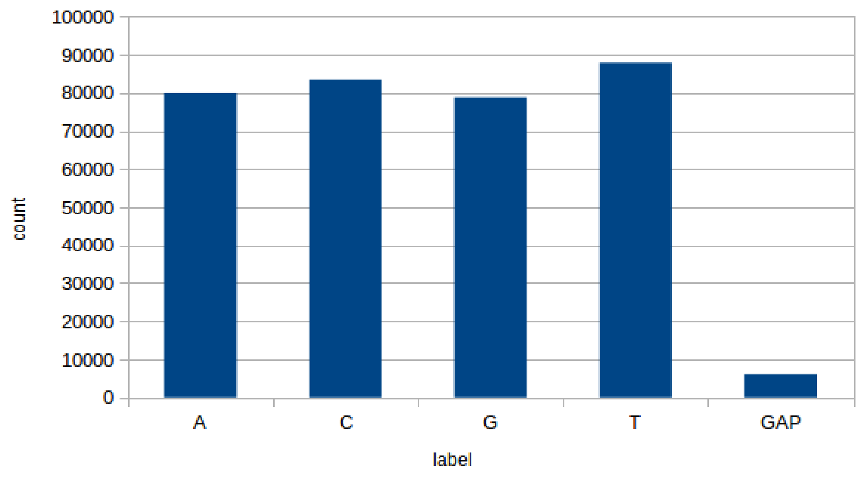

Figure 5: Distribution for 4 labels (bases) and gap in the training data

On other hand the HMM model predicts the correct class with an accuracy of 82.5% and it seems to do better than our model. But again we must note that the sample set for Oxford Nanopore is biased since we are only taking the examples that aligned. Only around 10% of the reads align and those are the best cases. Thus, if we assume that the HMM does no better than random guessing in the remaining 90% of the sample set, we would get an accuracy measure of $\sim$ 31%, which is "in theory" worse than our Naive Bayes model.

Table 4 and Table 5 shows the F1 score for Oxford Nanopore and Naive Bayes using the 4 base classes.

Table 4: F1 score for Oxford Nanopore

Class	Precision	Recall	F1-Score
A	0.822	0.799	0.811
C	0.861	0.828	0.844
G	0.827	0.787	0.806
T	0.919	0.885	0.885

Table 5: F1 scores using Naive Bayes Model

Class	Precision	Recall	F1-Score
A	0.249	0.510	0.335
C	0.262	0.387	0.340
G	0.381	0.294	0.332
T	0.453	0.277	0.344

Consequently, F1 score was calculated for each of the 4 classes (bases) both in case of Oxford Nanopore and Naive Bayes.

When the Edit Distance results from Oxford Nanopore and Naive Bayes is compared, it clearly shows that the HMM model (which had a Mean Edit Distance of 81.74 and a RMS Edit Distance of 146.58) performs better than the Naive Bayes Classifier (which had a Mean Edit Distance of 259.94 and a RMS Edit Distance of 425.96).

The results (both F1 and Edit Distance) point to the fact that the Naive Bayes performs worse than the HMM model (though it does not perform very bad). But there is no definite way to know that for sure because of inherent bias in the sample.

These are the ways in which our Naive Bayes model can be further improved:

- We can increase the size of the feature vector. In our analysis we never really calculated the auto-correlation of the events time series. It is entirely possible that increasing the size of the feature vector and learning against more time series features improves our results significantly.
- Our assumption that each base pair independently affects the current signal and its neighbourhood might be false. This seems like the greatest flaw in our Naive Bayes model. Of course we can rectify this problem by using some other learning algorithm like CNN, which does not assume that the feature vectors and the observations are independent and mutually exclusive.
- We think that our training set is large enough for learning. Looking at the results of our model using a larger data set might provide some interesting insights into the problem.

5 CONCLUSION

SVM does not seem to be a great approach for base calling, atleast when we use linear kernels and are trying to predict k-mers in general. When we tried using polynomial and RBF kernel, the $SVM^{multiclass}$ program failed to converge even on an overnight runs. However, using $SVM^{multiclass}$ on a larger dataset and with smaller number of classes may give better results.

Our implementation of Naive Bayes has given relatively better results, as shown in analysis above. Thus, we can conclude that our assumption that a base pair independently affects the signals in its neighbourhood, is not a very bad assumption and has merits. We can improve the Naive Bayes model by taking larger number of features(if the autocorrelation in the time series is not very high) or by using a larger training set. Other approaches like CNN etc can also be used to improve our model.

ACKNOWLEDGEMENTS

We are thankful to Prof. Rob Patro and Vivek Kulkarni for their help and guidance.

REFERENCES

1 "Altschul, Stephen F., et al. "Basic local alignment search tool." Journal of molecular biology 215.3 (1990): 403-410."

2 "Support-vector networks." by Cortes, Corinna, and Vladimir Vapnik. Machine learning 20.3 (1995): 273-297.

3 "A practical guide to support vector classification." by Hsu, Chih-Wei, Chih-Chung Chang, and Chih-Jen Lin. 2003

4 "On the algorithmic implementation of multiclass kernel-based vector machines." by Crammer, Koby, and Yoram Singer.The Journal of Machine Learning Research 2 (2002): 265-292.'

5 "DNA Base-Calling from a Nanopore Using a Viterbi Algorithm" by Winston Timp, Jeffrey Comer and Aleksei Aksimentiev. Biophysical Journal Volume 102, Issue 10, 16 May 2012, Pages L37–L39.

6 "naiveBayesCall: An efficient model-based base-calling algorithm for high-throughput sequencing" by Wei-Chun Kao and Yun S. Song. Journal of Computational Biology 2011.

7 " large scale SVM learning practical." by Joachims, Thorsten. 1999.

8 "Alta-Cyclic: a self-optimizing base caller for next -generation sequencing" by Yaniv Erlich, Partha P Mitra1, Melissa delaBastide1, W Richard McCombie1, Gregory J Hannon1. Nat Methods. 2008.

9 Bacterial whole-genome read data from the Oxford Nanopore Technologies MinION^{TM} nanopore sequencer (http://gigadb.org/dataset/100102).

10 "BayesCall: A model-based basecalling algorithm for high-throughput short-read sequencing" by Wei-Chun Kao1, Kristian Stevens and Yun S. Song. Genome Res 2009.