Predicting Percent Spliced In (PSI) in Alternative Splicing Using Deep Networks

July 6, 2016

Supervised by: Prof Bin Yu

Authors: Emin Arakelian, Fadi Kfoury

Abstract

Alternative splicing allows for gene products having various functions and increases protein diversity. The percent spliced in(PSI) indicates the efficiency of splicing a specific exon into the transcript population of a gene(1). In this project we attempt to predict the PSI of a set of exons using various gene regulators using a deep neural network and a shallow wide neural network. A better performance from neural networks is achieved thorough hyper parameter search which is capped by the computational power.

1 Introduction

The mRNA is spliced into protein synthesizing exons using regulators. In this project a dataset of regulator concentration and the corresponding PSI of the exons is presented. The dataset is imbalanced. Making a balanced dataset, random forest was trained as a benchmark with a validation AUC of 0.92. A deep and a shallow network were arbitrarily constructed and trained on the same data, performing poorly with AUCs close to 0.85. Tuning the networks for number of layers, number of neurons, the parameter for the L2 regularizer and the drop out rate increased the AUC to 0.91. Computational power proved to be a true bottleneck that potentially prevented a more thorough optimization of the networks' formation. A stability metric, the ES score, was computed that demonstrated the the relative instability of the networks as a model compared to random forest.

2 The Dataset

The DNA has all the information needed to produce various proteins or genes. To do that, firstly a copy of the DNA is transcripted that is the messenger

RNA or mRNA for short. The mRNA is then spliced to form a chain of exons, which are needed to build a certain protein. The exons code the recipe of the amino acids needed for a certain protein. The remaining parts, called introns, are discarded. The mRNA is spliced into exons by the help of certain regulator proteins that act as beacons on the intron, exon borders. Once the new mRNA is spliced, it is then translated into proteins. The process was first discovered by Francis Crick(2) . The percent spliced in index (PSI) indicates the efficiency of splicing a specific exon into the transcript population of a gene. A PSI of 1 indicates constitutive exons that are included in all transcripts and never removed. PSI values below 1 imply reduced inclusion of alternative exons and denote the percentage of proteins that contain the exon compared to the total transcript population. The PSI is calculated using:

$$PSI = \frac{IR}{IR + ER}$$

where IR is inclusion reads and ER is exclusion reads(1).

The dataset consists of observed exons with 299 regulator proteins as features having a value that represents their concentration, and the corresponding PSI of that exon. If the PSI is more than seventy percent we have a highly used exon, and if the PSI is less than thirty percent we have an exon that is not used as much. The task is, given the regulator protein concentrations, to determine whether a big percentage of the exon will be used in all the transcripts, which will be our positive class, or a small portion of the exons will be used which will be our negative class. The positive class is overpowering the negative class with a ration of about 88 to 12.

3 Analysis Framework

The main goal of this project is to train and tune a deep network and assess its performance against that the random forest, trying to attain an AUC close to random forest. All the analysis is done with in Python. Python libraries such as numpy(4) and scikit-learn(3) are used for handling the data and running machine learning algorithms. For the deep network the Keras(8) package is used which is a wrapper for the deep learning backend, Theano(7). In part 4 the data preparation processes are explained, exploratory and unsupervised methods are also investigated. Part 5 establishes a benchmark using random forest. Part 6 introduces deep neural networks and our approaches used in this report. The processes for tuning the deep networks, the choosing the optimal hyper-parameters and the comparison of the results with that of the random forest are presented in part 7. Further diagnostics are run on the misclassified samples, and the stability of the network is assessed against random forest which are reported in parts 8 and 9, respectively. Part 11 is the summary and part 13 enumerates the references used in this work.

4 Preprocessing and Exploratory Data Analysis

The data cleaning in this work involved dropping features that were all zeros, removing data points that were more than three standard deviations away as outliers, and dropping any non-numeric valued observation. The correlation matrix, displayed in figure 1, did not display too big of a correlated number of features. Hence all the features are used.

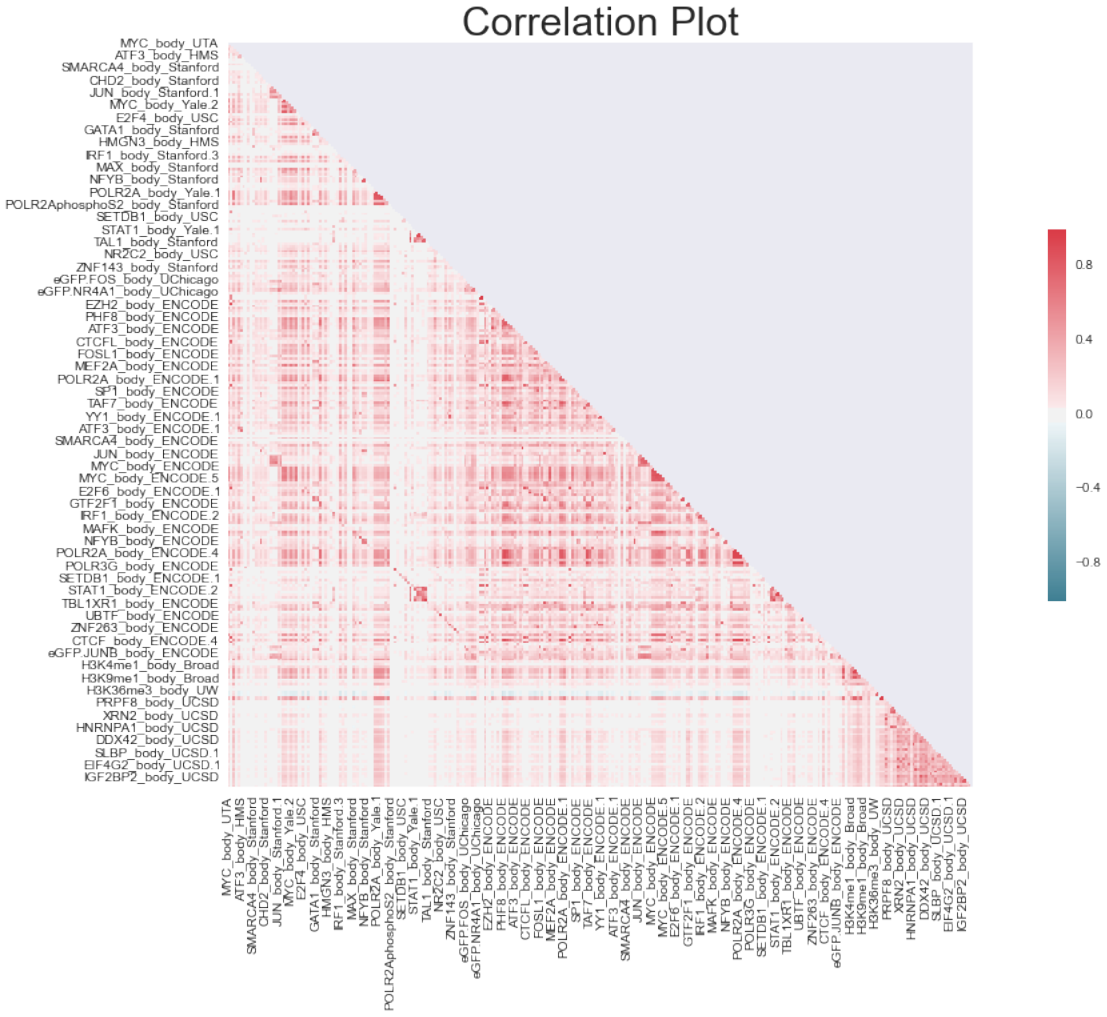

Figure 1: Correlation Matrix

To assess the structure in the data, kmeans with two centers is used and the predicted kmeans labels are compared with the observed labels in a confusion matrix displayed in table 1. The numbers are as a ratio of the whole dataset,

	y > 0.7	y < 0.7
K-Majority	0.82	0.14
K-Minority	0.03	0.01

Table 1: Kmeans Confusion Matrix

i.e. they add up to 1; of 12139 observed majority class, kmeans classifies 13639 as the majority class, and of the 2072 minority samples, kmeans predicts 572 as the minority class. This shows a weaker signal in the minority class. Since the task is to predict the high splicing rates versus the low splicing rates, samples with observed dependent value of at least 0.7 are transformed to label 1 and those with a value of less than 0.3 are transformed into label 0. The rest are deleted. The data has a slight imbalance of 88 to 12 percent with the majority class being class 1. To address this issue the training is done on a balanced dataset of 50 percent of each labels. Validation is done on 40% of the dataset and Testing on 20%.

5 Benchmarking with Random Forest

Random Forest is an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set(9). A random forest classifier is constructed by subsampling the features of a dataset for making a split at each node of a Bagged tree. A bagged(bootstrap aggregated) tree, is constructed by fitting a bootstrapped dataset on B decision trees and taking the majority vote of these trees as the prediction. The reason for subsampling of the feature space, is as follows: if a set of features are important in predicting the dependent variable, these features will keep showing up in all the trees, making the trees correlated. Ho et al(10) has shown the effect of subsampling the feature space on accuracy.

A random forest classifier of 50 trees had been fit to the balanced dataset. The result is shown in figure 11 in the appendix and a summary of the results i displayed in table 2. The out of bag error(OOB) for this classifier is 0.86. The out of bag error is the average error over all the 50 trees, when they predict the samples in the training set which they had not seen. Random forest can be used to determine feature importance and assist in feature selection. To do so, after computing the OOB, every feature is permuted across the dataset, and the new OOB for every feature is computed. The difference between the new and the base OOB is a metric for feature importance, as it means the permuted feature made a bigger difference. A plot of the feature importance is provided in figure 12 in the appendix. Using the important features only, the random forest performs almost the same, with slightly better accuracy but worse specificity, which means a worse job in the minority class.

Accuracy	Recall	Specificity
0.84	0.84	0.88

Table 2: Results for the Random Forest Classifier

6 Deep Networks

A one-hidden-layer deep network is a function $f : R^D \rightarrow R^L$, where D is the size of input vector x and L is the size of the output vector $f(x)$, such that, in matrix notation:

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x)))$$

with bias vectors $b^{(1)}, b^{(2)}$; weight matrices $W^{(1)}, W^{(2)}$ and activation functions G and s. Typical choices for the activation functions are the $ReLU = max(0, x)$, $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and $sigmoid(x) = \frac{1}{1+e^{-x}}$. A deep network is a sequence of activated layers. For training a deep network, the stochastic gradient descent optimizer is used:

$$\omega_{ij,k+1} = \omega_{ij,k} - \alpha \frac{\partial E}{\partial \omega_{ij}}$$

Where $\omega_{ij,k}$ is a vector of weights between neuron i and j, at iteration k and α is the learning rate. The partial derivative $\frac{\partial E}{\partial \omega_{ij}}$ is calculated using the chain rule as follows:

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial in_j} \frac{\partial in_j}{\partial \omega_{ij}}$$

where o_j is the output vector to layer j and in_j is the input vector to layer j. $\frac{\partial in_j}{\partial \omega_{ij}} = \frac{\partial}{\partial \omega_{ij}}(\sum_{k=1}^{n} \omega_{kj} o_k) = o_i$ and if the activation function is sigmoid as it is in our case then $\frac{\partial o_j}{\partial in_j} = sigmoid(in_j)(1 - sigmoid(in_j))$. The first term: $\frac{\partial E}{\partial o_j}$, can be evaluated with a recursive chain rule as follows:

$$\frac{\partial E}{\partial o_j} = \sum_{l \in L} (\frac{\partial E}{\partial in_l} \frac{\partial in_l}{\partial o_j}) = \sum_{l \in L} (\frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial in_l} \omega_{jl})$$

where L is the set of all the neurons. Putting it all togethere we will get:

$$\frac{\partial E}{\partial \omega_{ij}} = \delta_j o_j$$

where $\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial in_j}$.()

Theano's Python wrapper, Keras, has been used for implementing the deep networks in this project. For benchmarking purposes, three networks were run on the data:

1. A three layered base network, the number of neurons for which respectively were : 10, 20, 10.

2. A three layered wider network, the number of neurons for which respectively were : 20, 40, 20

3. A five layered deeper network, the number of neurons for which respectively were : 10, 15, 20, 15, 10

The choice of the number of neurons and layers were arbitrary, as deep networks need tuning to perform well. A comparison of their ROC curve against random forest and the AUC score can be seen in figure 2. It is evident that while they perform very similarly, their performance is inferior to that of the random forest. It must be noted that the hyper parameters to be tuned for deep networks are numerous, including but not restricted to: number of layers, number of neurons, learning rate, learning rate decay, momentum coefficient, parameter of the L1 or L2 regularizer, choice of the regularizer, choice of the activation functions, the drop out rate etc. Since empirically one layered wide networks have shown promising results for biological data, the remainder of the project will investigate a deep and a one layered wide network.

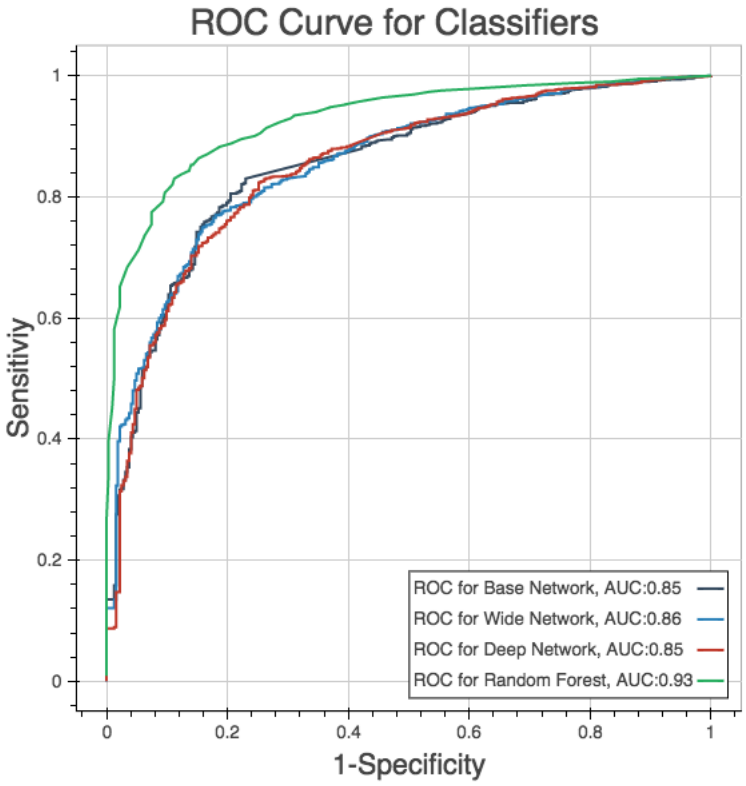

Figure 2: ROC curve comparison

7 Tuning Results and Comparison

Given the vast space of hyper parameters, the ones chosen for tuning are: number of neurons, number of layers, parameter for the L2 norm and the drop out rate. The learning rate has been chosen to be 0.001 over 200 epochs, which is sufficient to reach convergence, as the learning rate is small enough and the iterations are high enough. Due to the expensive computational cost, a simplifying assumption was made, where the it was assumed that the number of layers and number of neurons do not have an interaction with the rest of the hyper parameters. Moreover the grid search of the number of neurons and number of layers was done in a stepwise fashion: at layer one, the best number of neurons was chosen based on the accuracy score. Then keeping the previous best number of neurons, a new layer was added and another best number of neurons was chosen. A much more expensive alternative would allow for a best number of neurons choice for all of the layers, given a certain depth. For instance, assume for layer 1, 100 neurons resulted in the best accuracy. In the approach done in this project, the 100 will be kept constant while searching for the best number of neurons in the next layer, while in the expensive method, the previous 100 will be only valid for 1 layered networks, and for 2 layered networks two sets of neurons should be tuned, one for the previous one and one for the current one. This will reduce the time complexity from $O(n^l)$ to $O(nl)$. In this project we tuned over 9 neurons values and 12 layers; each tuning of one layer would take about an hour, with increasing complexity as new neurons were added. Thus with the current method the tuning of the neurons and layers took about 4 days. With the expensive and thorough method this would take about 32 thousand years. After the tuning of the neurons and layers was done, a tuning of the L2 parameter and dropout rate was done. The results are show in figures 3 and 4

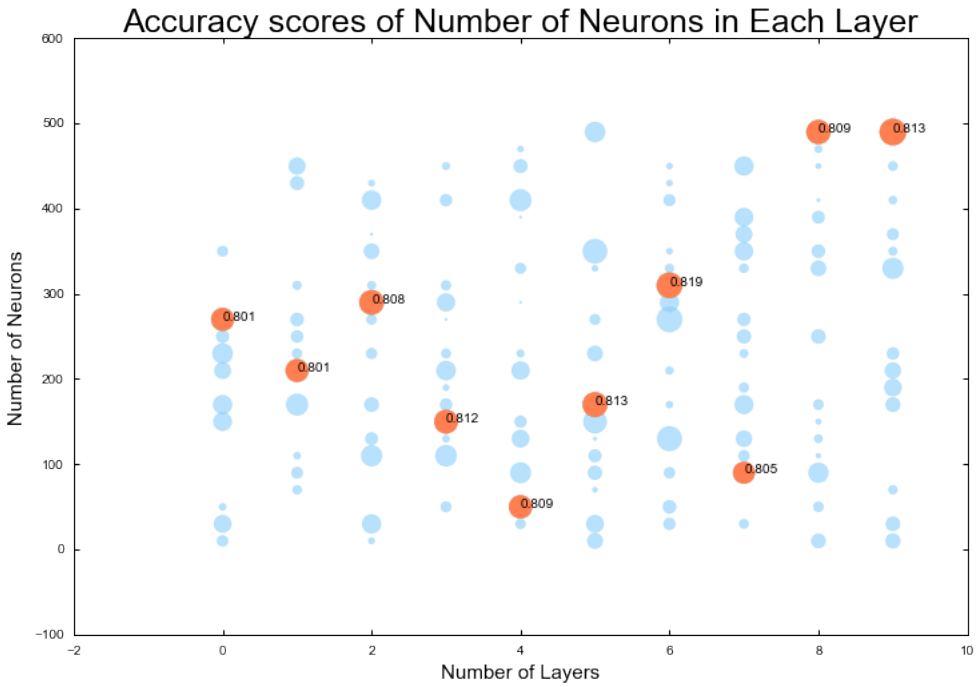

Figure 3: Accuracy score in number of neurons versus number of layers

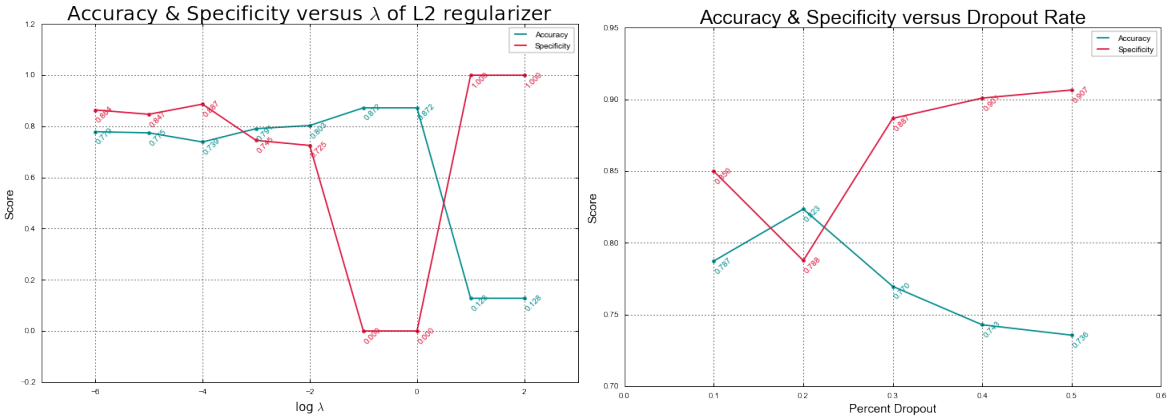

Figure 4: L2 parameter and Drop out tuning

Seven layers was chosen as the optimal number of layers for the deep network. The rationale is that while there is a small variance around the accuracy, the first big leap in the number of neurons to match the same accuracy happens on layer 6, which is the seventh layer. For λ, 10^{-3} was chosen as the parameter,

because that is were the accuracy and specificity were the closest, indicating a good true positive and true negative count. As the model becomes simpler specificity decreases because, although we would be classifying less false positives, we would also be classifying less true negatives. For a very big lambda the network arbitrarily predicts all 0's or 1', which is evident from the accuracy, always showing the ration of the population of the other class, e.g. when the network predicts all 0's, the accuracy is 12% which the portion of the zero labels in the population. The confusion matrices for all of the L2 parameter values are given in figure the appendix. For the drop out rate 0.22 was chosen.

For the one layered wide network, the neurons range was increased and for each neuron multiple runs were made to get a sense of the variability in accuracy. Although there is variance everywhere, neurons 800 and 900 display the highest mean. The results are displayed in figure 5. The L1 and L2 regularizer was tuned on the one layered network as well. Since the difference in accuracy was not significant and L2 is faster computationally, the L2 norm was chosen with a λ of $10^{-}3$. The results for this can be seen in figure 13 in the appendix.

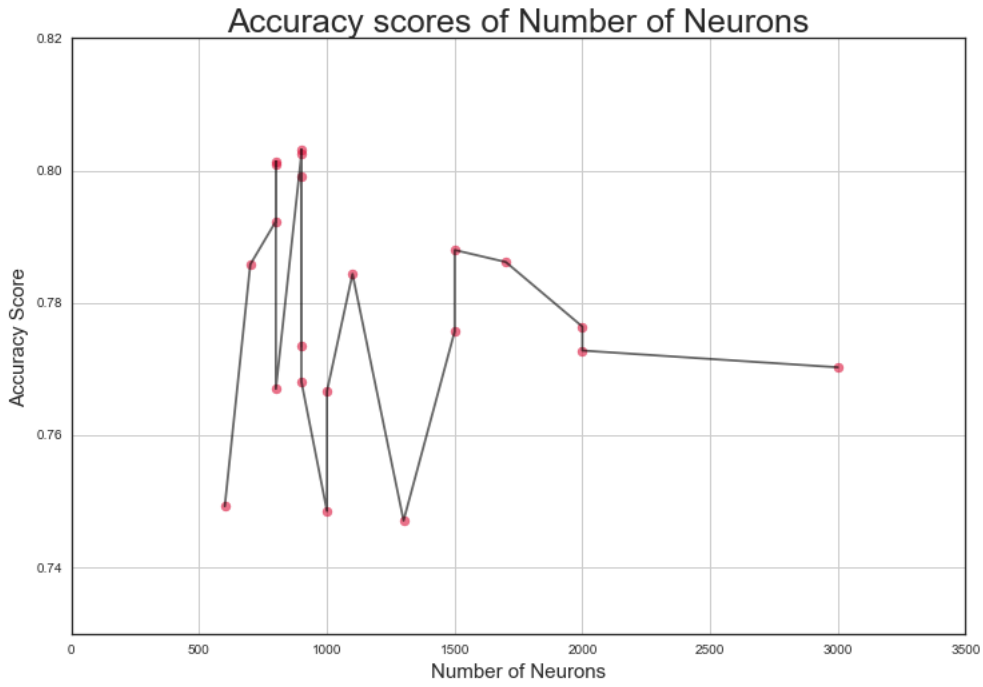

Figure 5: Tuning the number of neurons in the one layered network

8 Diagnostics

After the tuning, the networks showed significant improvement on the validation set, on par with that of the random forest as displayed in figure 6. To try and improve the results further, only the important features were fed to the networks and they were trained again. This did not provide much improvement, which signifies the networks' success in identifying the important features by itself.

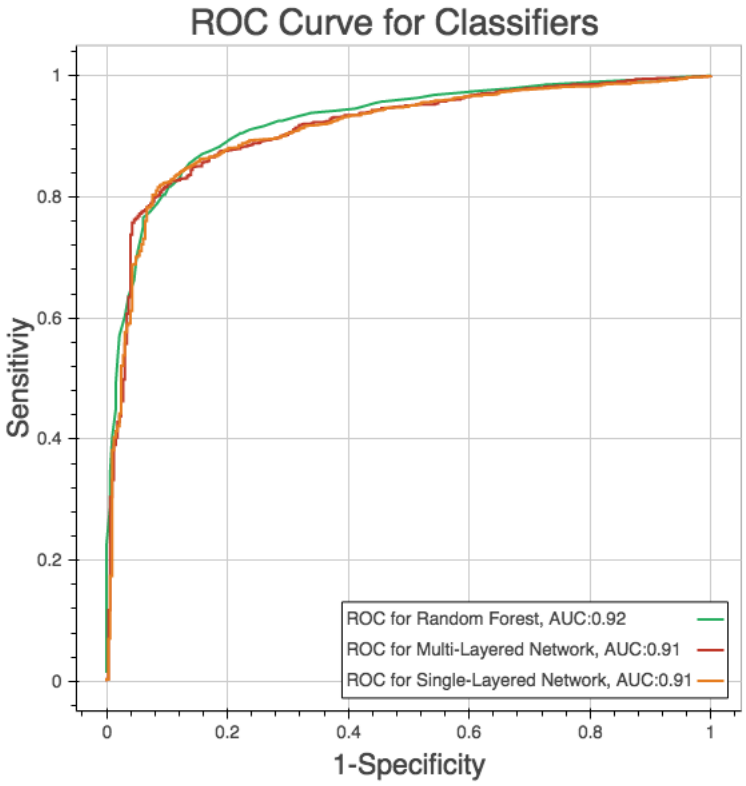

Figure 6: Optimized classifier comparison

The accuracy scores were as follows in table 3. The validation set has 2764 samples while the test set has 4767 samples.

Set \ Accuracy score	Random Forest	Single-Layered Network	Multi-Layered Network
Validation Set	0.84	0.80	0.78
Test Set	0.87	0.83	0.81

Table 3: Accuracy Scores

Another approach that was tried was getting the misclassified training sam-

ples and grouping them into false positives and false negatives. Then a correlation assessment was done on each of the groups, but the resulting correlations were very low. To amplify the signal in the misclassified training samples, the misclassified samples were re-fed to the deep network numerous times and the resulting network was tested on the training and the validation set. This procedure is essentially weighting up the misclassified samples. Figure 7shows the outcome. In both training and validation sets, the accuracy increases and the specificity decreases, which signifies a better predictive power in the majority class while sacrificing the minority class, which is not something that we are leaning towards. The volatility of the specificity may signify the big step size, but the result show here has a learning rate of $10^{-}5$. Since this approach did not show much promise it was not investigated further.

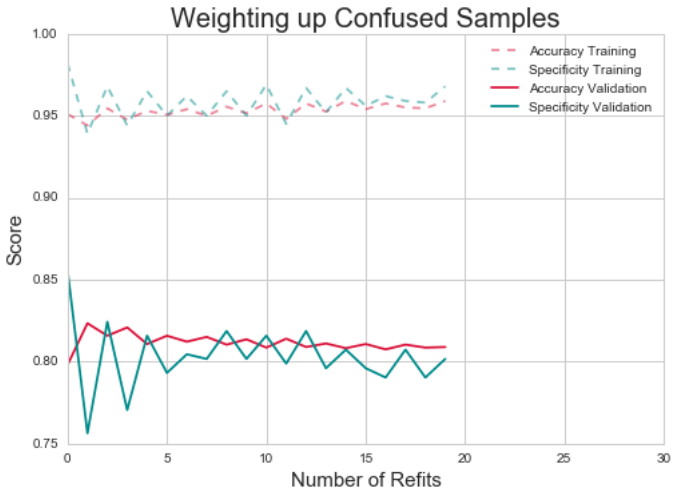

Figure 7: Accuracy and Specificity changes over many refits of the misclassified samples

9 Stability

Yu et al 2015(11), presented a Estimation Stability(ES) score, which is at its essence, the normalized variance of a set of vectors. Coupling ES with cross validation, ESCV, on K folds, one can arrive at a parameter estimate that has the lowest MSE or the highest accuracy, and at the same time is the most stable estimate, for instance in estimating the parameter of an L1 norm, one would have:

$$ES(\lambda) = \frac{\widehat{Var}(Y[\hat{\lambda}])}{\| \bar{\hat{Y}}[\lambda] \|_2^2}$$

Which would give us a score for the stability of our estimated model. Given the λ_{CV}, the ES score chooses a model that is smaller than the CV model, but

with more stability, i.e. a smaller ES score:

$$\lambda_{ESCV} = \arg\min_{\lambda \in \Lambda} ES(\lambda)$$

$$\Lambda = \{\lambda \geq \lambda_{CV} \mid ES(\lambda) = \min_{\omega \in (\lambda+\varepsilon, \lambda-\varepsilon)} ES(\omega), for\ some\ \varepsilon > 0\}$$

The variation in the predictions, come from the k-fold cross validation; as each fold will have its own labels to be predicted. In this section, the stability of the networks will be compared to that of the random forest with a method inspired by ESCV. Later the L2 parameter of the networks will be tuned using the ESCV.

To compare the stabilities, the ES score was obtained against different hold out sets. At each hold out set, 10 bootstrap samples were drawn from the remaining data. The samples were half the size of the remaining data each. Using these samples, 10 models were trained and their performance was assessed on a separate validation set. The result is shown in the left graph of figure 8. The idea is that the lower the ES score, the lower the normalized variance of the predictions should be. Also, the milder the slope, the less sensitive the model is to perturbations. If a model is less sensitive, it can signify that it learns faster, as it needs less data to stabilize. One might argue that in this case, the varying sample size might bias the results. Yu 2013* shows that:

$$ES(\tau) = \frac{\frac{1}{V}\sum_V \parallel X\hat{\beta}_v(\tau) - \hat{m}(\tau) \parallel^2}{\hat{m}^2(\tau)} = \frac{d}{n-d}\frac{1}{Z^2(\tau)}$$

where n is the sample size for fold V and $d = \lfloor \frac{n}{V} \rfloor$. Thus with a simple scaling we will arrive at the results in the right graph of figure 8. This will result in the $\frac{1}{Z^2(\tau)}$ as a score, and while it has increased range the of the score and also made the models more sensitive by having a slightly steeper slope, their relative stability is in tact. The previously computed ES scores are shown with dashed lines on the $\frac{1}{Z^2(\tau)}$ graph for scaling comparison.

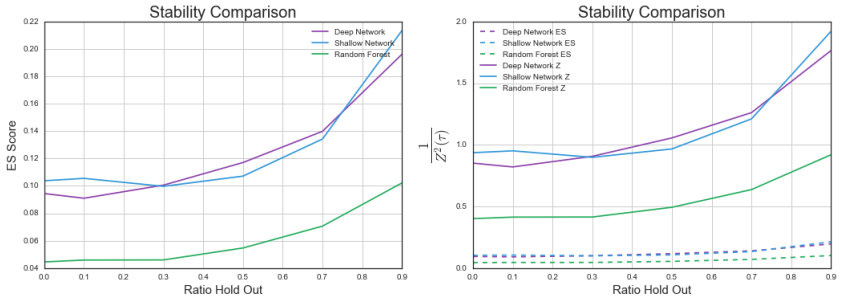

Figure 8: Left:Stability versus hold out set, Right:Controlled for sample size

To remain faithful to the original paper, the ES score is again calculated but this time, instead of bootstrapped samples, after taking out the hold out set,

the remaining samples are split into 10 folds, and 9 folds are provided to the models to train on and all of the training set will be used for prediction. The overlapping of the samples here will be more similar to the one proposed by Yu, but at the same time it will be less conservative as 90% of the data is used for training in this setting. The results for this procedure are presented in figure 9. As we can see, although the results are very similar, this method has a much lower stability score, indicating more stability. The sudden jump in the deep networks stability is because of chance.

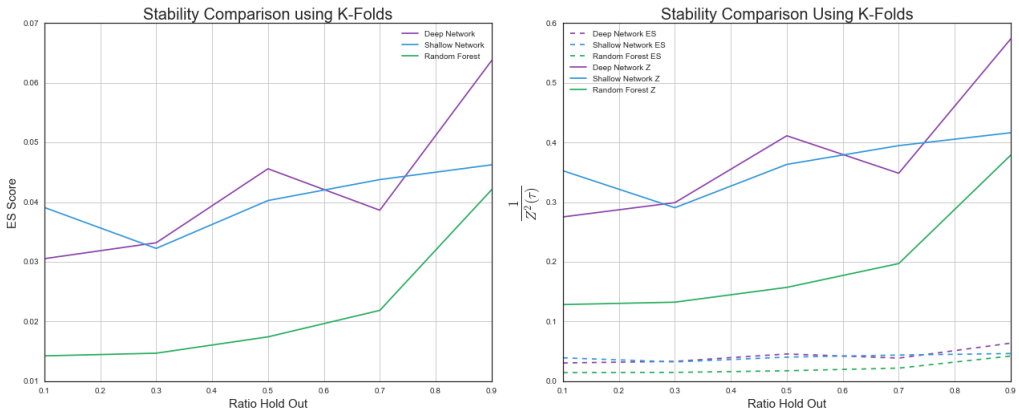

Figure 9: Left:Stability versus hold out set, Right:Controlled for sample size

A tuning of the L2 regularizer was done on the shallow network using the ESCV and the result is presented in figure 10 for demonstration. As seen the highest accuracy and lowest ES are arrived at around 10^{-4} to 10^{-3} with the latter allowing for the simplest model. At $\lambda = 1$ all the labels are predicted as 1 and thus we have 0 specificity and 0.5 accuracy and after that the classifier predicts all 0's which results in a specificity of 1 and accuracy of 0.5.

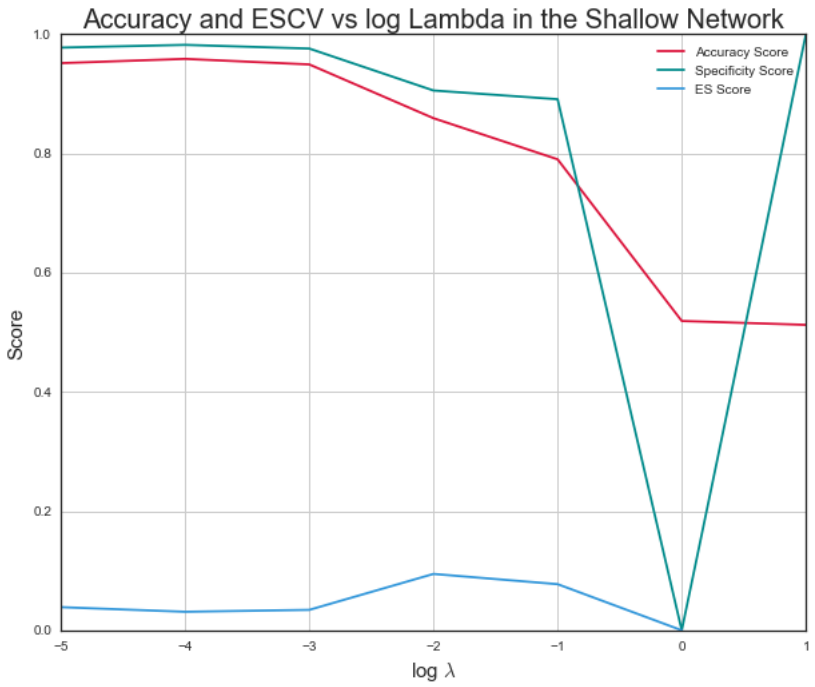

Figure 10: Tuning of the L2 parameter using ESCV

10 Decision Surfaces

Given the constraints of two dimensional vizualization, it seemed that all three classifiers were very consistent in choosing their decision surfaces. The top five most important features, per random forest, were chosen to display the decision surfaces. Although they seem very similar, it must be noted that this could be due to low dimensional projection and given more unfoldings of the dimensions we would expect to see more variable surfaces. The decision surface of the five most important features is given in figure 14 in the appendix.

11 Conclusion and Challenges

In this project the performance of deep networks was assessed against random forest. Random Forest is very robust in accuracy, AUC and stability. It also takes less than 10 seconds to run and requires no tuning. The Deep networks on the other hand needed quite a bit of tuning to be able to reach an AUC close to random forest. They also were less stable models. Deep networks outshine random forest in relatively large datasets, and require the infrastructure to run in reasonable time. Deep networks also need more experience to set up compared to an off the shelf random forest. None of the models need much feature

engineering. Given the extensive hyper parameter space of the deep networks, computational power and time became a real bottle neck in this project and compensations were made. A more stable model can be acquired by choosing all the hyper parameters using the ESCV score, but since it uses a k-fold system, computational power again is a bottle neck. Trying different permutations of the activation functions may also prove beneficial. One approach that may improve performance is introducing squared terms of the features. In spiral shaped data, squared terms increase the performance of the networks.

12 Acknowledgement

We would like to thank Sumanta Basu whose suggestions and feedbacks guided us and the project to fruition.

13 References

1. Schafer, S., Miao, K., Benson, C.C., Heinig, M., Cook, S.A., and Hubner, N. Alternative splicing signatures in RNA-seq data: percent spliced in (PSI), 2015.

2. Crick, Francis.On Protein Synthesis. The Symposia of the Society for Experimental Biology 12: 138-163, 1958.

3. Pedregosa et al. Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.

4. Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30, 2011.

5. Fernando Pérez and Brian E. Granger. IPython: A System for Interactive Scientific Computing, Computing in Science & Engineering, 9, 21-29, 2007.

6. John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95, 2007.

7. Theano Development Team. “Theano: A Python framework for fast computation of mathematical expressions”.

8. Chollet, Franois, Keras, https://github.com/fchollet/keras, 2015.

9. Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome. The Elements of Statistical Learning,(578-588) 2008.

10. Ho, Tin Kam, "A Data Complexity Analysis of Comparative Advantages of Decision Forest Constructors" Pattern Analysis and Applications: 102–112, 2002.

11. Lim, Chinghway and Bin Yu. "Estimation Stability With Cross Validation (ESCV)". Arxiv.org. N.p., 2013. Web. 14 June 2016.

14 Appendix

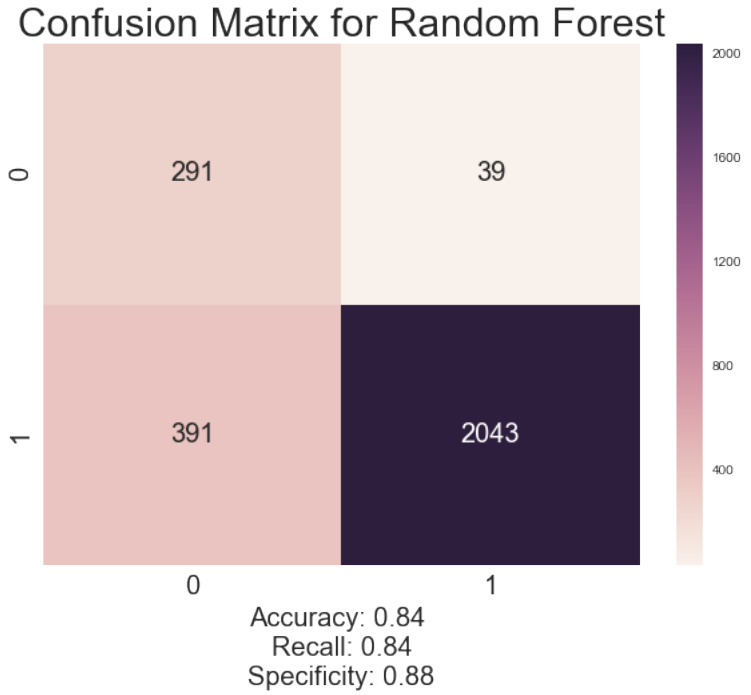

Figure 11: Confusion Matrix for Random Forest

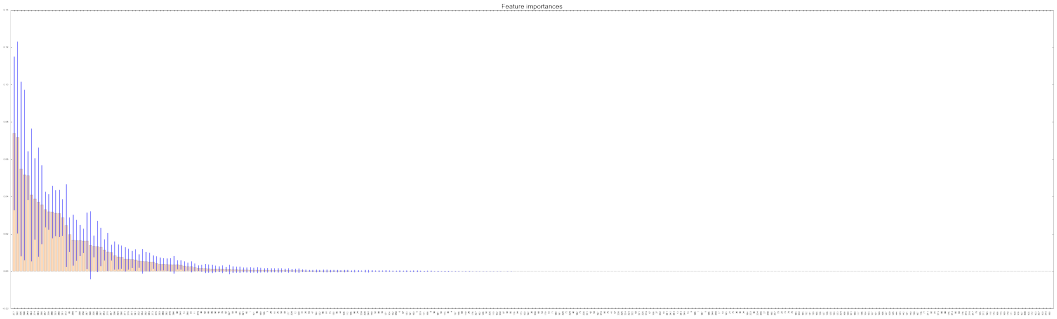

Figure 12: Feature importance(Please zoom in for better resolution)

Figure 13: Confusion matrices for λ of the L2 Norm in the deep network

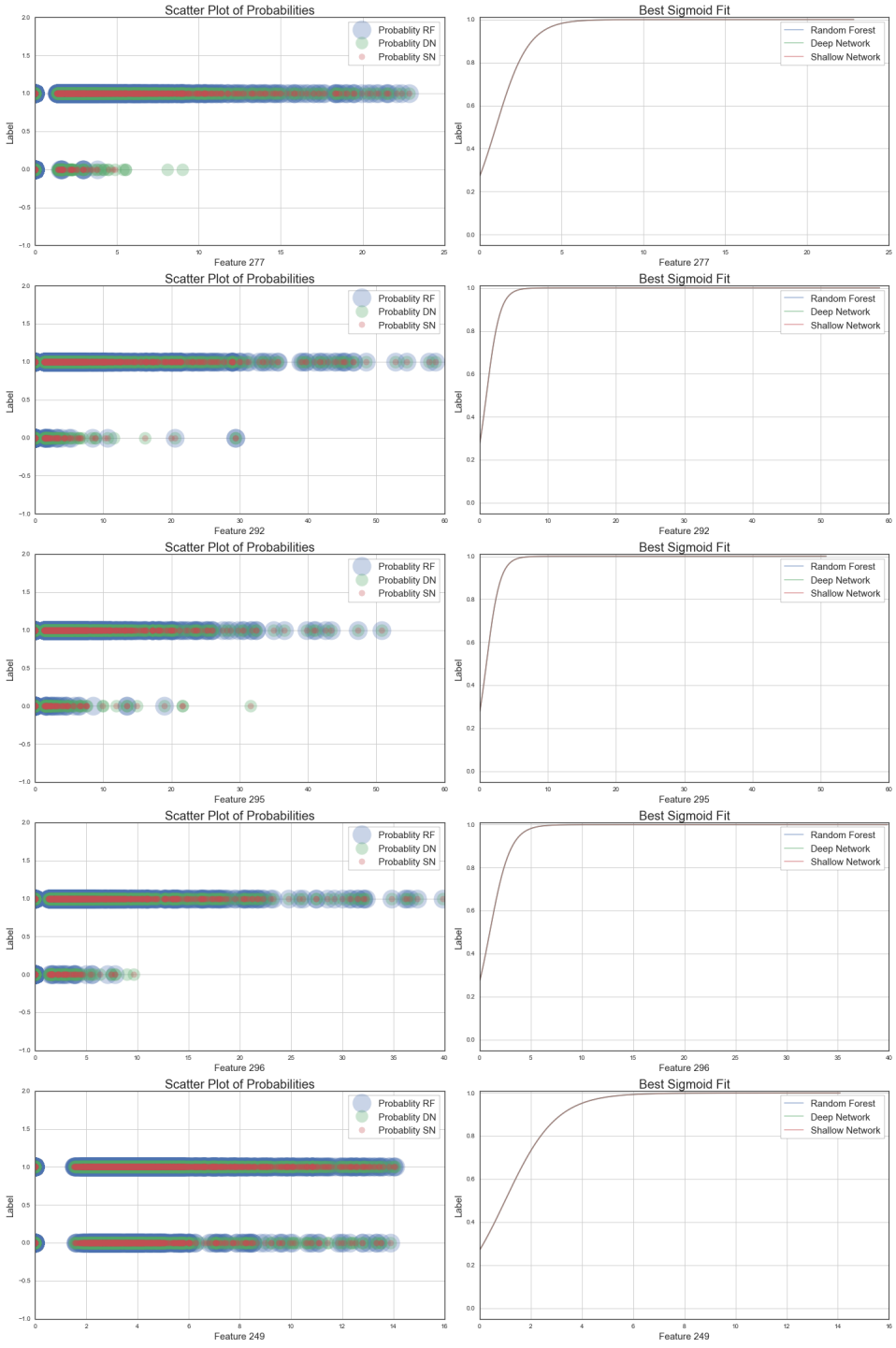

Figure 14: Decision surfaces for the top five important features