CS4011:Programming Contest

Vaibhav Nayel EE15B117, Anshuman Karthik ME15B150

November 12, 2017

1 Introduction

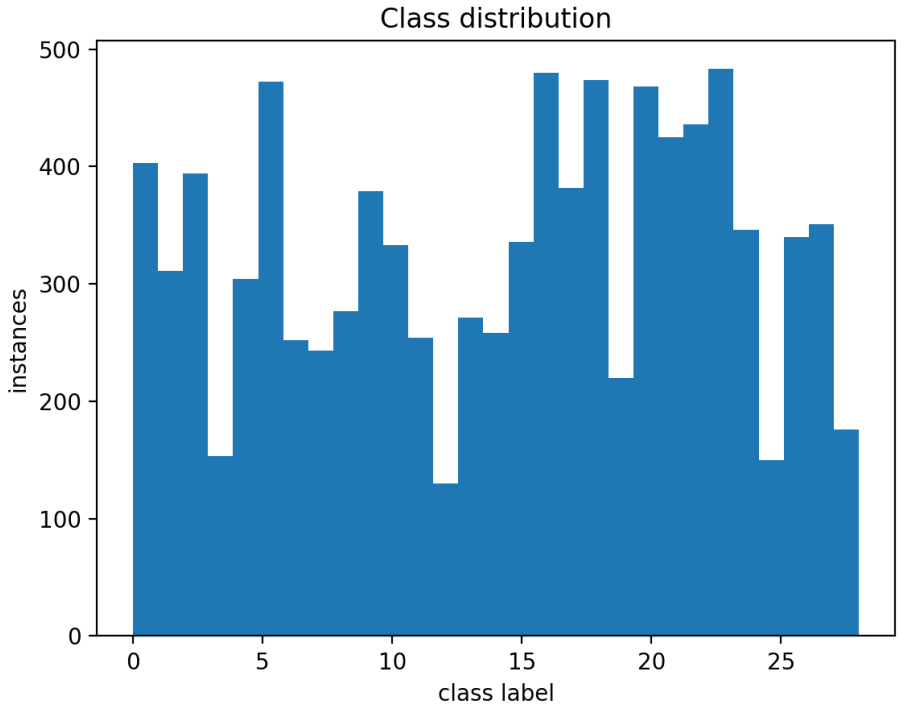
Given a large dataset with missing values, high dimensionality and a large number of classes, our task is to make a classifier to maximise the f1 score on a test dataset which also has missing values.

2 Dataset Analysis

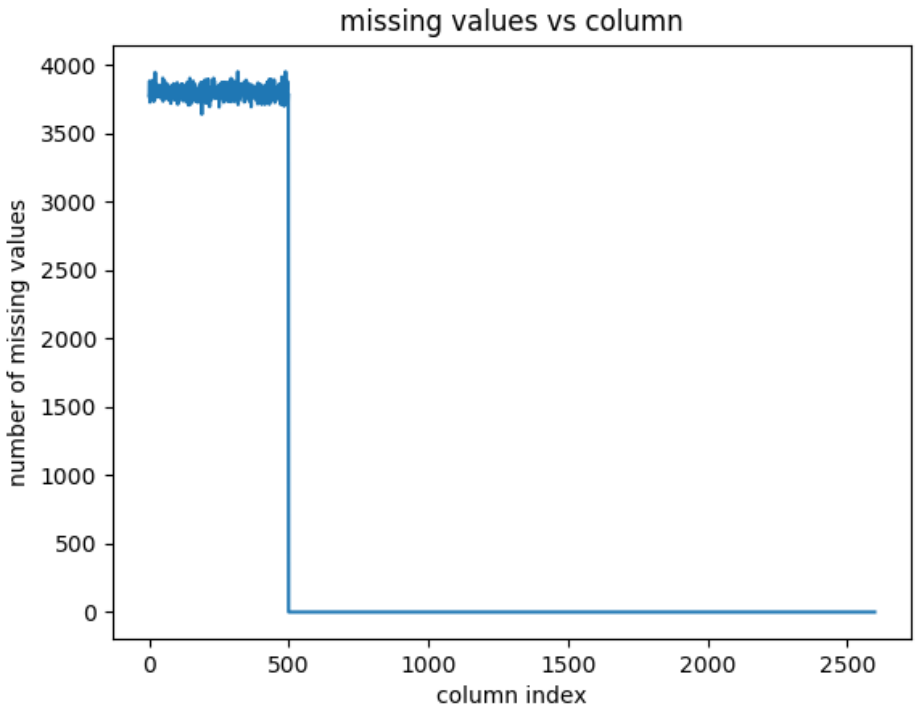
Before we begin training models, we would like to understand our data better. We do this because prior knowledge of the data may help us to better select or eliminate models before spending too much time on them.

```
Dataset summary:
cols=2600
rows=9501
normalised=True
no. classes=29
missing values=True
```

Since the data is already normalised, we need not perform any scaling. However, we should take a look at the class distribution of our data to avoid the pitfalls of class imbalance.



We can see that the classes are skewed. We should take care that minority classes do not get consistently misclassified. We should therefore keep f1 as our scoring criterion rather than accuracy.
Next, we examine the missing value distribution in our data.

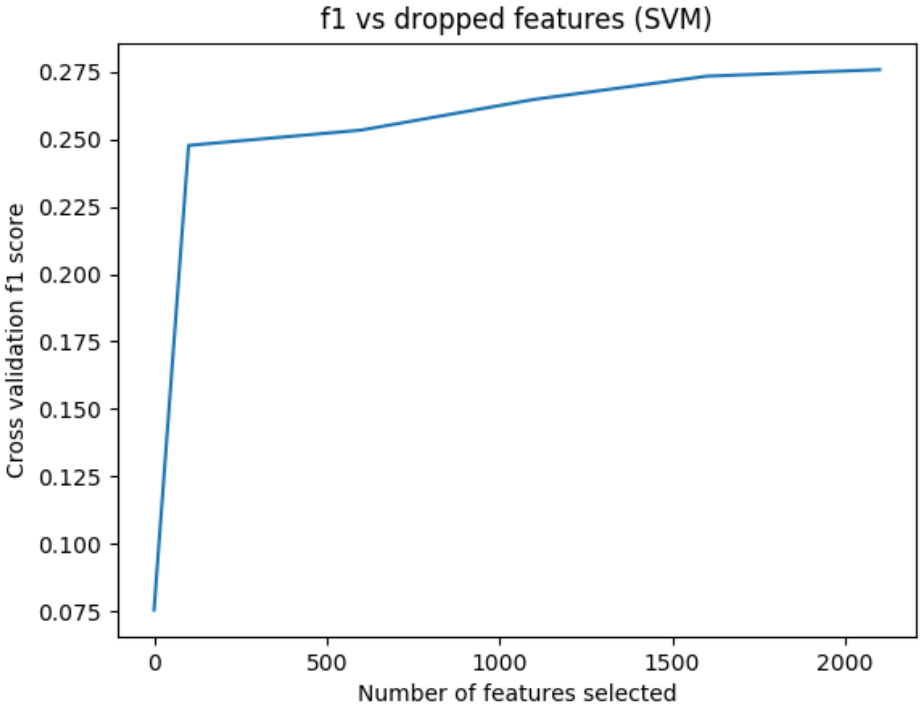


This is quite an interesting observation. Only the first 500 columns contain about 3700 missing values each. We can use this fact to impute the missing values using information from the other 2100 columns.
We should also check to see whether missing values are abundant in a particular class since this may make class imbalance even stronger.

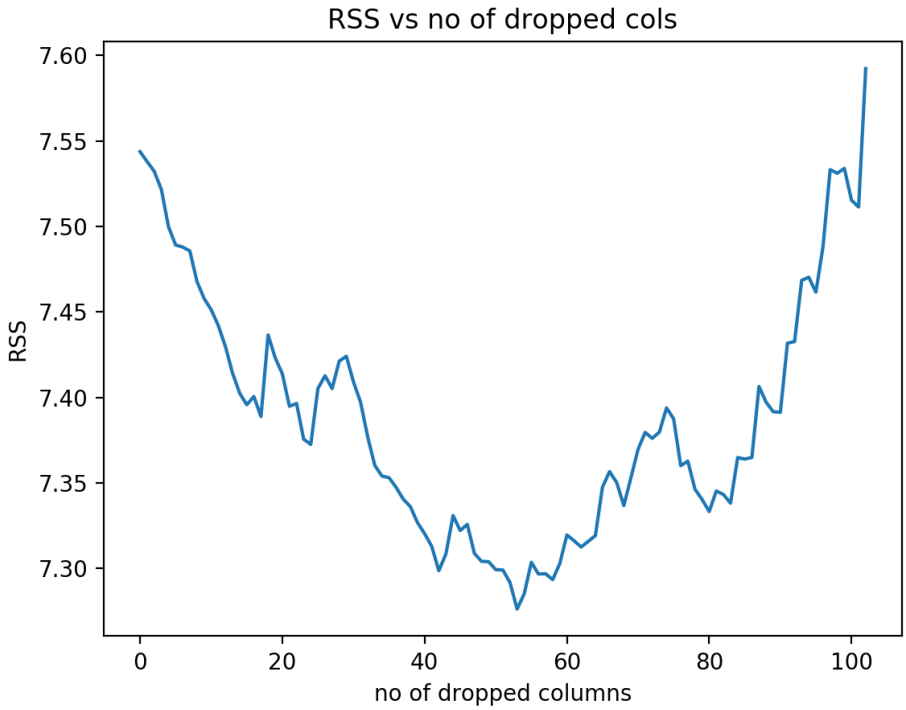
Luckily, it so happens that all classes have about 7% of their values missing, so no extra imbalance is induced.

3 Imputation

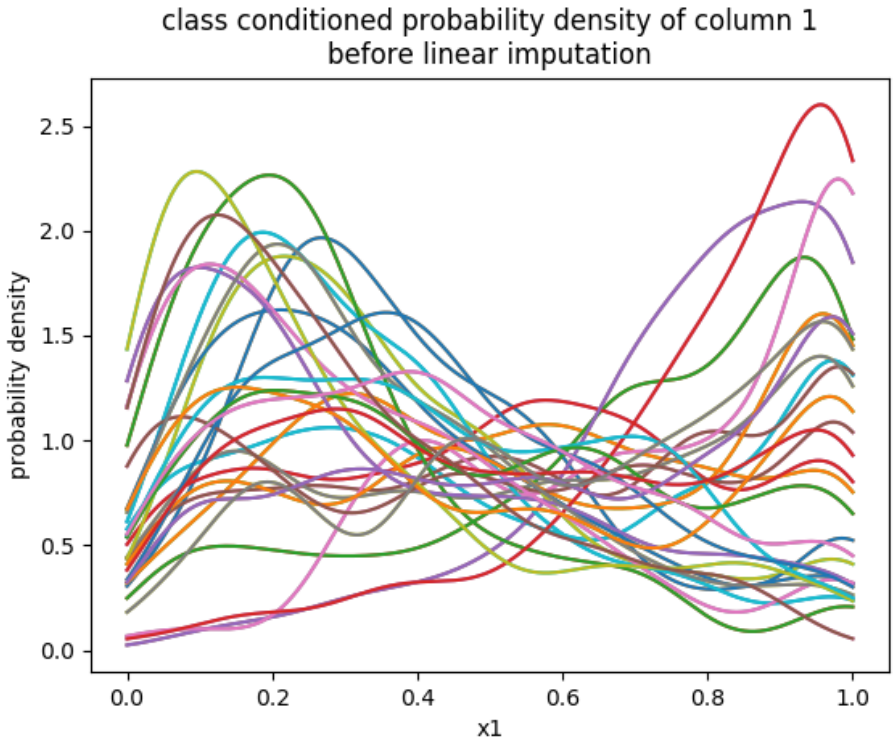
In this section we will attempt imputation by a few different methods. Before we do this however, we should take rough baseline estimates with simple models to ensure that methods like full information imputation aren't encoding too much information into the missing columns since our test data has no labels and full information retrieval is not possible in that case.
For baseline, we chose a linear svm and trained them only on the last 2100 columns in order to perform feature ranking/ feature selection. After doing this, we will train the same model on the imputed data to see if they assign high importance to the imputed columns. If this happens, our imputation was probably too informative for our model and this will not generalise well to unseen data due to bias induced in the training data.



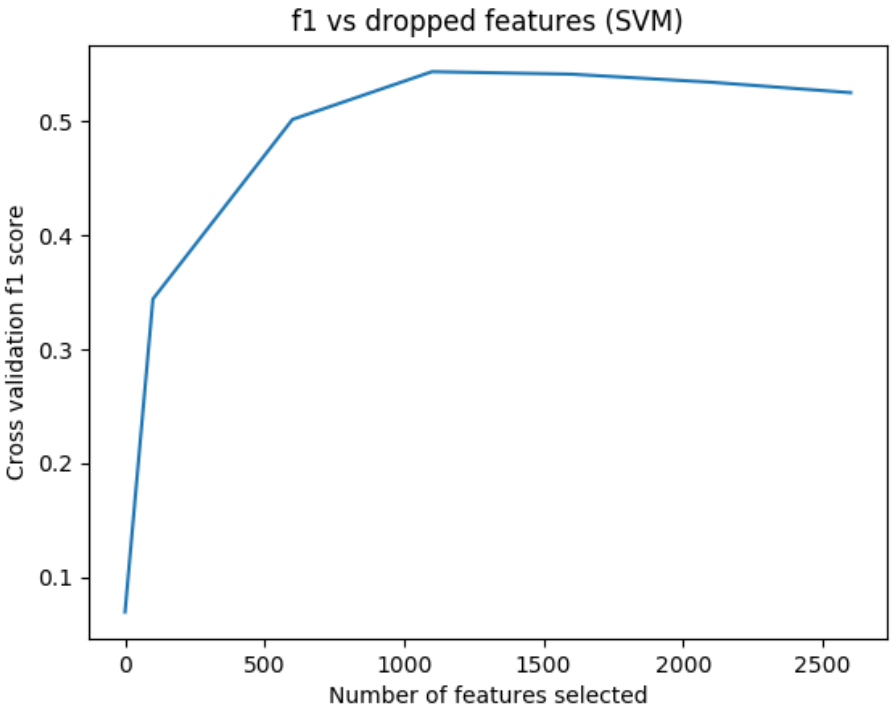
We see that the f1 score only varies by about 2% when more than 1000 of the non-missing features are being used and starts to fall off a little faster for less than 1000 features. This should tell us that all the features contain useful information and not noise. Compare this to a problem from PA1 where some of the columns contained information that was not predictive of crime rate in a city.



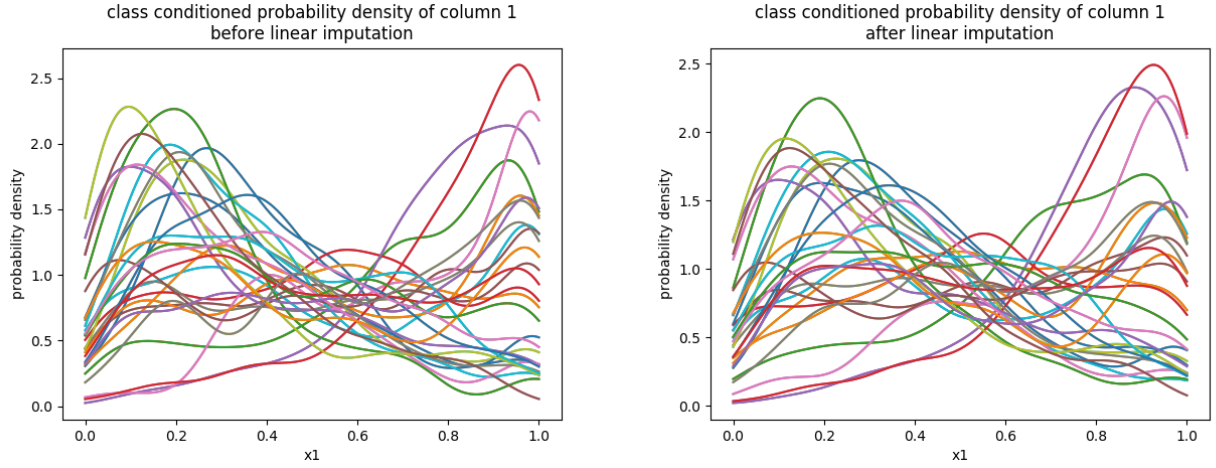
In this scenario, dropping columns actually reduced our error! This is not seen in the present dataset. Further, this should tell us that if a model trained on imputed data gives disproportionate weight to imputed values and very small weights to the present columns, we have given the imputed columns too much information about its class.
To begin with, we tried imputation using sampling from class conditioned densities. Below are the densities of one of the columns. It remains unchanged after imputation since samples are drawn from this distribution itself.



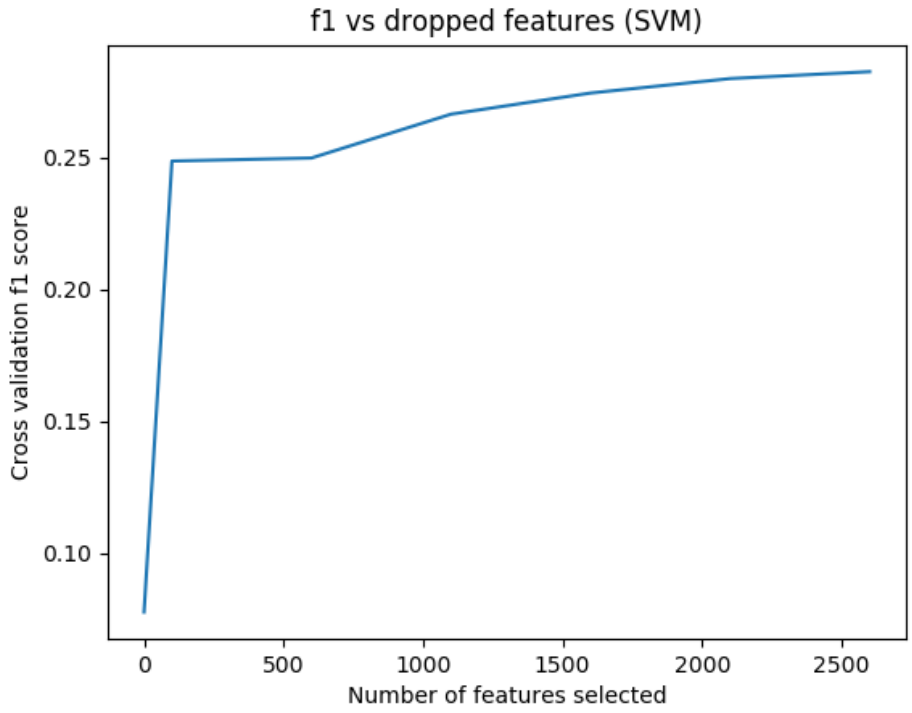
We can plot the f1-score vs columns dropped again.



Clearly, this indicates that too much information was passed to the first 500 columns while imputing them because an f1 score is too large a jump. We must therefore avoid this method.
We will next try linear imputation and plot similar graphs.



We see that the class conditioned density has changed slightly which is expected due to noise in the rest of the columns. The shapes are largely preserved, so this does not set off any red flags regarding extra or false information being embedded in the columns.



The graph of dropped features is flat like our initial graph, so we can be fairly certain that we haven't messed up our dataset with this technique. The data is then normalised to mean 0 and variance 1.

4 Classifier Performance

In this section we attempt to evaluate the performance of different classifiers on the data, along with various ensemble methods to improve their performance. First, we implement the basic versions of the classifiers without any ensembles. Their performances are listed below. It must be noted that the cross-validation scores provided are for 5-fold cross validation with the f1 score measure as the performance metric.

Linear Discriminant Analysis

LDA gave us a score of 15% which led us to believe that the classes are probably not gaussian clusters of similar shape. They could be multimodal and this throws off LDA. From this, we learned that it is probably best to use algorithms that do not make unimodal assumptions about the underlying distribution of classes.

Gaussian Mixture Models

We attempted to model the data as a Gaussian Mixture Model with 29 components to give easier classification. Here, we assumed a diagonal covariance matrix, which is probably the reason for the high failure of this model. This is due to the probable interdependence of the features, giving us a terrible precision in modeling the data.

Naive Bayes

We have that Naive Baye's for the training data with a Gaussian likelihood performs terribly, with a score of 20.725. This indicates that the data must have non-independent features. Also, this indicates that the data is probably not text data.

Support Vector Machines

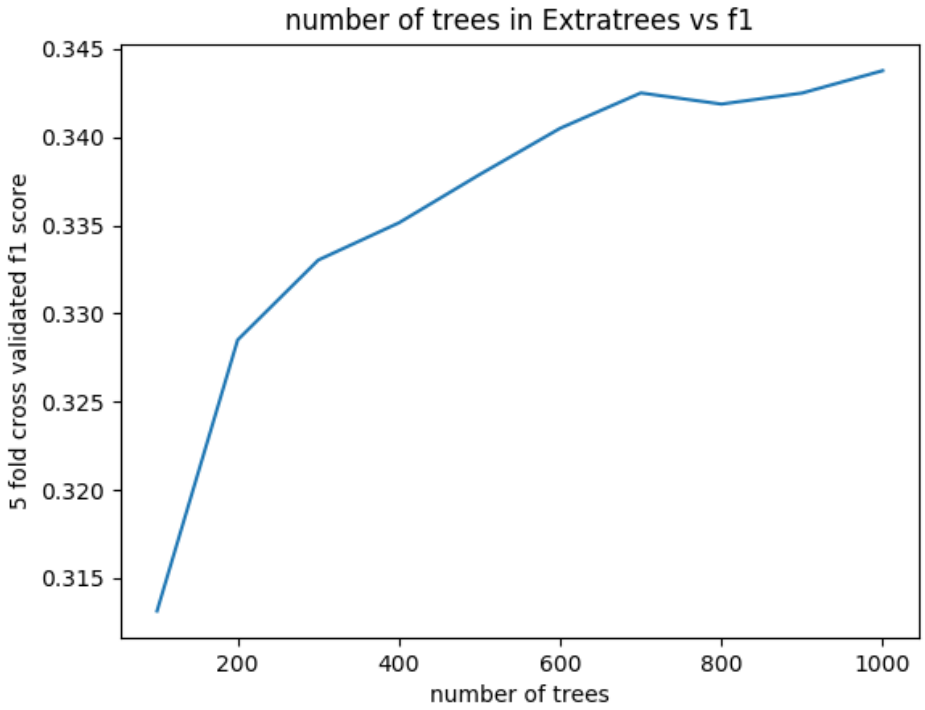
Support Vector Machines proved to be powerful base classifiers on the data, providing high f1 score with the right parameter. Now, we have, after an exponential search for the right hyper-parameter get $C = 100$. A simple, non-ensemble based classifier gave a score of 40.272 upon training. Now, with the linear kernel, we have that the radial basis function works best as a kernel.

- Linear Kernel: 28.6192

- Gaussian Kernel: 40.272

Decision Trees, Random Forests and Extra trees

Decision Trees performed poorly compared to the SVMs as well, giving a score of 14.380. This is a result of our high dimensionality data, seeing as the data will have many un-correlated features. (As a note, the high dimensionality of the data also poses inverse of the same problem to Naive Bayes.) The Random Forest Classifier performed better, with a score of 30.292, due to the increased stability of the Random Forest as compared to a Decision tree. sklearn contains a variant of random forests called extremely randomized forests which decreases stability slightly but can sometimes provide better results than RF. In our case we get about 34% f1 score with 1000 trees.



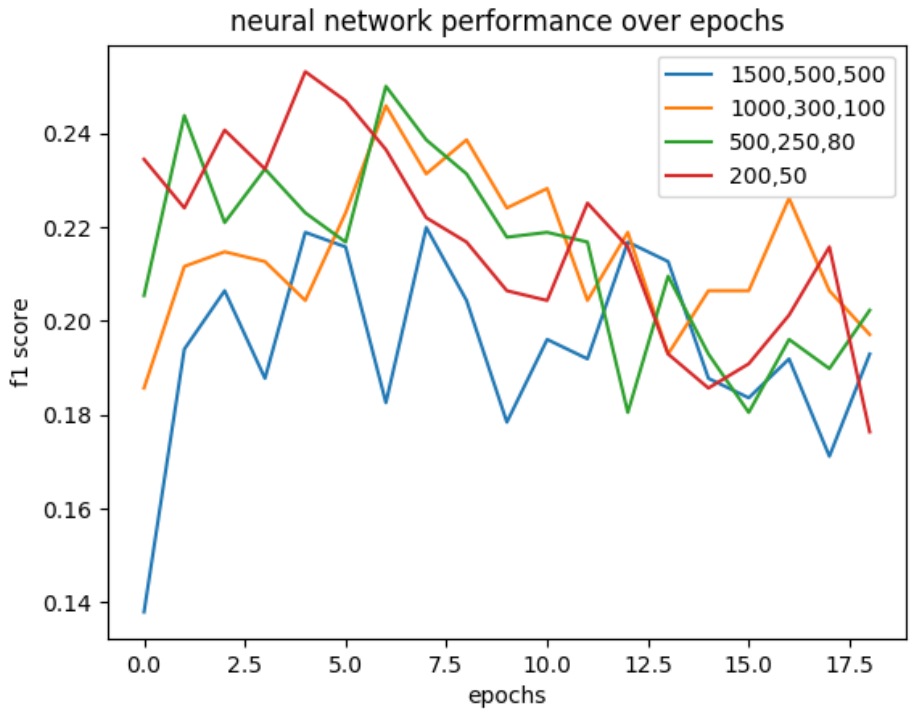
The score tends to plateau off around 35% but it takes far too long to train.

XGBoost

We used the XGBoost library to train our model, getting a score of 36.342 as per our 5-fold cross validation fit. Although not a base classifier, we include it here as it was tried along with our other bases. More has been explored in XGBoost. We will look at this in the ensemble methods section.

Artificial Neural Networks

Using Tensorflow, we constructed Artificial Neural Networks which gave us an f score around 20%



Clearly, these classifiers are weak and therefore lend themselves to ensemble methods like bagging. On bagging 40 networks we were able to achieve a score of 27%

5 Ensemble Approaches

In this section, we implement ensemble methods over the previously mentioned classifiers to improve their performance.

Support Vector Machine

We noticed that the Support Vector Machine classifier performed the best on our training data, given the right choice of hyper-parameters. We employed the following ensemble methods on the SVM, which yielded the following results.

NOTE: The SVM tuning is slightly ad-hoc as compared to some of the other models as the training time of the SVM over the data is extremely high. This is also why the hyperparameter tuning of the SVM isn't as granular as it could b. It remains at an exponential level of abstraction.

Bagging

We tried bagging the SVM while the data was both centered as well as non-centered. As predicted, the scaled version performed loads better, giving a cross-validation f1 score of roughly 0.38 as supposed to 0.36 with the non-centered, non-scaled data. We encountered the following issues with bagging.

- We noticed repeated sampling issues when we set the value of the number of estimators as something low (10, in this case). Thus the only possible alternative was to train the bag with a higher number of estimators, which we set as 30. Sadly, we didn't see any significant jump in the classifier performance, indicating the saturation of the effectiveness of the bag.

Boosting with Adaboost

Boosting with Adaboost yielded significantly poorer results as compared to even the simple base classifier. We operated with only 10 estimators with Adaboost, which still gave us a shocking cross-validation score of 0.049. Hence, we didn't pursue boosting for the SVM classifier.

XGBoost and Associated Tuning

The main concerns for us in XGBoost tuning were the regularization parameter and the maximum depth of the tree. As in Support Vector Machines, due to high training time, we have that an ad-hoc exponential search was the most appropriate option, given computational capacity constraints.

- For a regularization parameter of 10 and 0.1, we have that the algorithm gives us a 0.32 score, both plateauing in roughly the same time. Hence, we know that our optimal value of our regularization parameter is in this area. As usual, a granular search is infeasible and unnecessary, as similar orders of magnitudes for regularization would yield similar results.
- Coming to maximum depth of the tree, we need to ensure that the data isn't over fitted by increasing the maximum depth of the tree to a vary large extent. Hence, we tried a set of highly regularized models upon a high max depth. The aim was to get a complexity-controlled tree that could still pull off high granularity. However, we have that the data was still highly overfitted, giving us a poor cross validation score.

Thus, the final parameters for our XGBoost model are:

- maximum depth (to avoid overfitting)
- regularization rate = 1
- learning rate = 0.5

Voting among the Ensemble of Classifiers

As an end result to improve the score, we voted among some of the best classifiers that we got. These were predominantly the Support Vector Machine, the bagged Support Vector Machine, the Extra Tree classifier and XGBoost with the right parameters as given above. A simple majority vote was conducted among the classifiers for the label of a particular instance with a random label being chosen in the case of a tie. We have that this performed worse, showing that the SVM's altered predictions hindered its performance. The parameters chosen in this committee machine were:

- Support Vector Machine with $C = 100$ and $\gamma =$'auto'.
- Bagged Support Vector with estimators as 100 and similar parameters as above.
- Extra Tree classifier
- XG Boost with 3 as the maximum depth and the default parameters for the rest

6 Conclusion

Our best 4 models, as given for submission are the 4 we took for the vote, again listed below for convenience.

- Support Vector Machine with $C = 100$ and $\gamma =$'auto'.
- Bagged Support Vector with estimators as 100 and similar parameters as above.
- Extra Tree classifier
- XG Boost with 3 as the maximum depth and the default parameters for the rest