# Predicting US Graduate Program Admissions using Machine Learning

**CSE343 Machine Learning Project Final Report**
**Group No. 31**


Shloak Aggarwal
2017107
shloak17107@iiitd.ac.in

Syed Ali Abbas Rizvi
2017114
syed17114@iiitd.ac.in

Tanish Jain
2017115
tanish17115@iiitd.ac.in


## 1. Abstract and Introduction

Hundreds of thousands of undergraduate students in India apply to graduate programs in the United States every year. A single student's application requires many components such as a GRE score, TOEFL score, Statement of Purpose, Letters of Recommendation, CGPA, Research Experience, Conference Publications, Internship Experiences, Industry Experience, Research Experience, and others. Applying to a single university can cost hundreds of dollars in application fees, sending test scores, and more. This project aims to solve this problem by predicting whether an applicant will be admitted to a university or not. The prediction is done by analysing Edulix.com's [1] data of Indian students who previously applied to graduate programs in the US.

## 2. Related work

Several studies have been performed previously to model admissions data. One study from 2018 [2] used the same dataset from Edulix.com, but applied different methods of data cleaning and preprocessing related to missing data, inconsistent data, noise, unstructured data, and redundancy. The paper's baseline model gave an F-Score of 0.6743 while other models gave lower F-Scores. Modeling and predicting data for only one university gave a higher F-score of 0.6979. In our study, we do achieve a lower F-Score for the baseline model, however, more complex models end up improving the F-Score.

A similar study that also scraped data from edulix.com created a recommendation system that would suggest a list of 10 universities where the chance of a student getting an admit would be highest [4]. Their baseline model had an accuracy of only 22.2%, and their best model, SVM, had an accuracy of 53.4%.

Another study developed a statistical machine learning system called GRADE to support the work of a university graduate admissions [5]. Although GRADE achieved an accuracy of 87.1%, their baseline model already achieved an 84.4% accuracy just by rejecting all applicants. Hence, they had highly imbalanced data. However, they reported precision-recall characteristics as better metrics for their system.

## 3. Methodology

### 3.1. Dataset and preprocessing

The data for this project has been extracted from user joemanley01's dataset in his Github repository [3]. The dataset was constructed by web scraping user profiles on Edulix.com, a website where users make a profile to get to know whether they have high chances or not of getting admissions in colleges in the US. The total number of samples was 53645; after preprocessing, we were left with 47519 samples.

| | | |
|---|---|---|
| User Name | Major | Specialization |
| Department | User Profile Link | Term & Year |
| UG College | University Name | Program |
| Research Exp | Industry Exp | Intern Exp |
| Journal Pubs | ConfPubs | TOEFL Score |
| TOEFL Essay | greV | greQ |
| greA | gmatA | gmatQ |
| gmatV | Topper CGPA | CGPA |
| CGPA Scale | Admission | |

Table 1. List of all features

If we deleted all the samples that had null values in one or more columns, we were left with only 7 samples, of which 6 belonged to the same person. Due to this sparseness of the dataset, it was necessary to apply effective preprocessing techniques. The original dataset had 25 features, which can be seen in Table 1. Out of the 25, we removed 9 features - Username, Specialization, Major, Department, User Profile Link, Term & Year, gmatV, gmatQ, and gmatA. Username and User Profile Link were irrelevant for the purpose of this study. 'Specialization', 'Major', 'Department', and 'Term & Year' were too imbalanced with multiple vari-

ants of the same value. Over 85% of the samples had values missing for 'gmatA', 'gmatV', and 'gmatQ'.

Some CGPA and CGPA Scale values were 0, so we removed the samples corresponding to those. We had different CGPA scales throughout the dataset, so we standardized the CGPAs of all the samples to be out of 10. The same was done for the Topper CGPA column. At the end, we removed the CGPA Scale column as it was no longer needed.

The null or 0 values in some of the columns like greA, greQ, greV were replaced with the mean of the whole column. We also replaced the values of 0 in the Topper CGPA feature with 9, assuming that the best performing student in a class will have at least a CGPA of 9. Finally, using Sklearn's LabelEncoder, we encoded the features with string values into integers between 0 and number of classes minus 1.

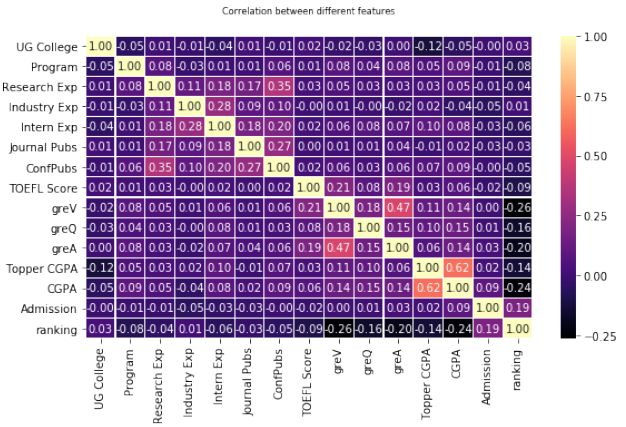


Figure 1. Heatmap for dataset

The heatmap in Figure 1 shows the correlation between different features of the dataset. Some interesting observations:

1. We are getting relatively high negative correlations between Ranking and greV, greQ, greA, and CGPA. This means that the above-average students with higher CGPA and GRE scores are more inclined towards applying to the lower-integer i.e. better ranked universities.

2. As expected, we also see a relatively high positive correlation between Ranking and Admission, meaning that the lower ranked universities are more lenient about who they accept.

3. Conference Publications has a high correlation with Research Experience, incontrovertibly.

4. greV has a high correlation with greQ, meaning that students who perform well in one section of the GRE are likely to perform well in the other.

### 3.2. Models

We trained several models of the Sklearn library on our preprocessed dataset. We tried to increase the achieved accuracy by changing the parameters of the models as best as we could. Table 2 shows all the models that we trained along with their parameters.

| Model Name | Parameters |
|---|---|
| SVM | Linear Kernel |
| SVM | Polynomial Kernel |
| SVM | RBF Kernel |
| Random Forest | 10 trees |
| Random Forest | 15 trees |
| Random Forest | 1000 trees |
| Logistic Regression | l1 loss |
| Logistic Regression | l2 loss |
| K-Nearest Neighbours | 2 neighbours |
| K-Nearest Neighbours | 3 neighbours |
| Multi Layer Perceptron | 1000 iterations |
| Multi Layer Perceptron | 2000 iterations |
| Bernoulli Naïve Bayes | None |
| Complement Naïve Bayes | None |
| Gaussian Naïve Bayes | None |
| Multinomial Naïve Bayes | None |
| Tensorflow Neural Net | Layers = [15, 30, 60, 2], Tanh & Softmax activation |

Table 2. Models and their parameters

All these models were fitted on five folds and observations were made on which parameters gave the optimum performance. Then observations were made on which models gave the best F1-score and accuracy on our dataset.

We then proceeded to train a custom Tensorflow model on our dataset. The model was trained for 1000 iterations with a learning rate of 0.0001 and the Adam optimizer. The input layer consisted of 15 neurons and the output layer consisted of 2 neurons (one for 'admitted' and one for 'not admitted'). The output used the Softmax activation function which is good for probabilities. The model was trained by varying the activation functions in each layer, number of layers, and number of neurons in each layer. First, only one hidden layer was chosen and the number of neurons was varied. Then, the number of hidden layers were increased one step at a time. The number of hidden layers were increased until the testing accuracy started decreasing instead of increasing. This model gave the best performance for Tensowflow.

## 4. Results

For our baseline model, we trained some models, using Support Vector Machines (with Poly, Linear and RBF kernels), Random Forest Classifier, Logistic Regression Classifier, Multilayer Perceptron, K Nearest Neighbours Classifier, and finally we made a custom Neural Network using the Tensorflow library in Python. Table 3 shows the obtained mean accuracies and mean F1-Scores.

| Model | Mean Accuracy | Mean F1-Score |
|---|---|---|
| SVM – Linear | 0.6064 | 0.6025 |
| SVM – Polynomial | 0.6076 | 0.5967 |
| SVM – RBF | 0.5860 | 0.6159 |
| Random Forest – 10 | 0.9117 | 0.9082 |
| Random Forest – 15 | 0.9232 | 0.9243 |
| Random Forest – 1000 | 0.9267 | 0.9308 |
| Logistic Regression l1 | 0.6076 | 0.6132 |
| Logistic Regression l2 | 0.5780 | 0.5899 |
| K-Nearest Neighbours - 2 | 0.7461 | 0.6870 |
| K-Nearest Neighbours - 3 | 0.9232 | 0.7559 |
| MLP – 1000 iterations | 0.5595 | 0.6047 |
| MLP – 2000 iterations | 0.5514 | 0.3469 |
| Bernoulli Naïve Bayes | 0.5290 | 0.6463 |
| Complement Naïve Bayes | 0.5661 | 0.5503 |
| Gaussian Naïve Bayes | 0.5362 | 0.6557 |
| Multinomial Naïve Bayes | 0.5660 | 0.5504 |
| Tensorflow Neural Network | 0.5610 | 0.5860 |

Table 3. Results for different models

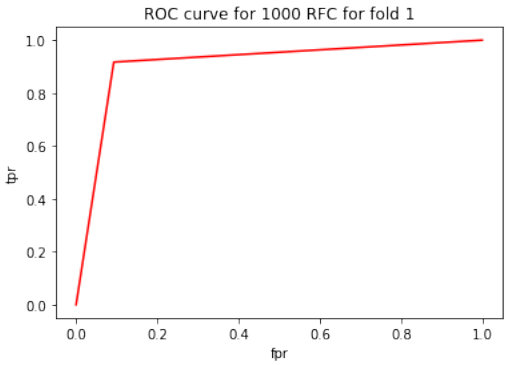


Figure 2. ROC curve for best model: Random Forest classifier with 1000 trees

The ROC curve and the Confusion matrix of the best model, Random Forest Classifier, are shown in Figure 2 and Figure 3, respectively. As you can see from these figures their diagonal entries are comparatively greater than those of the other trained models. It can also be observed that the area under the ROC curves for the RFC model is higher.

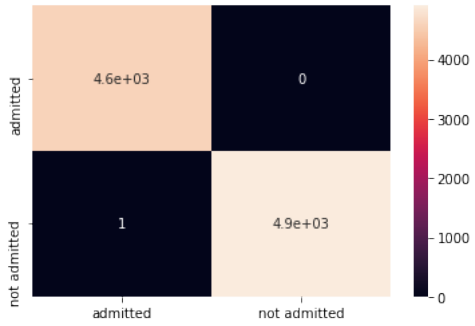


Figure 3. Confusion matrix for best model: Random Forest classifier with 1000 trees

(The ROC curves and the Confusion Matrices that we obtained for all the other models for the first fold are given after the References.)

For all of the models, there was hardly any difference in performance across the 5 folds. Our choice of choosing 5 folds was based on the fact that more folds would result in less testing data and less folds would result in longer training time as there would be more training samples. For the Neural Network made using the Tensorflow library, we tried to obtain as high of an accuracy as possible but were not able go beyond the testing accuracy of 56.10%. This accuracy is almost the same as the accuracy we got from the other models. This shows that the data is more suited to Random Forest classifiers, and thus ensemble of weak classifiers.

## 5. Conclusion

In most models we obtained accuracies of around 60%, but using K-Nearest Neighbours we obtained accuracies around 75%, and using Random Forest Classifier we obtained a very high accuracy of around 98%.

The state-of-the-art accuracy that we came across is around 87% for similar topics [5]. In our study's data, the users who created profiles on Edulix.com did not fully fill all the fields in their profiles. As a result, out data was very sparse. Likewise, other researchers who did similar studies got a different number of samples after they did their own preprocessing.

In most cases, researchers seem to drop a lot of samples as the data is highly sparse. We, on the other hand, only dropped around 6000 samples out of the total 53600. This is why we obtained such high testing and training accuracy using random forest classifier and these accuracies didn't change much even if we changed the number of trees from 10 or 15 to 1000. For the other models, the accuracies remained near 50 to 60% even after the parameters were varied greatly.

## References


[1] edulix.com.

[2] B. Ghai. Analysis & prediction of american graduate admissions process. Master's thesis, Department of Computer Science, Stony Brook University, Stony Brook, NY 11794, 2018.

[3] joemanley01. universityrecommendationsystem. Github.com.

[4] R. Swaminathan, J. M. Gnanasekaran, S. Krishnakumar, and A. S. Kumar. *University Recommender System for Graduate Studies in USA*. PhD thesis, University of California San Diego, 2015.

[5] A. Waters and R. Miikkulainen. Grade: Machine learning support for graduate admissions. In *Twenty-Fifth Innovative Applications of Artificial Intelligence Conference*, 2013.


## Other Figures

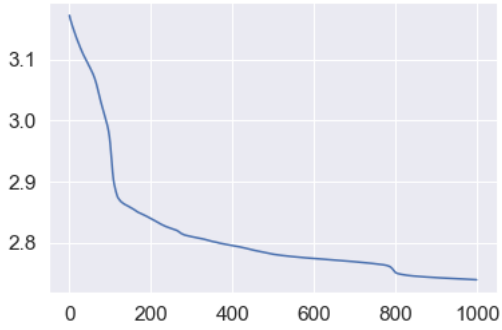


Figure 4. Cost vs Epochs for Tensorflow Neural Network

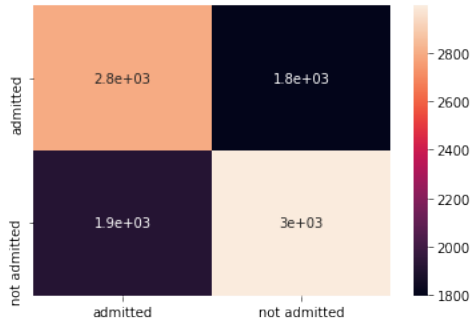


Figure 5. Confusion matrix for Logistic Regression with l1 penalty

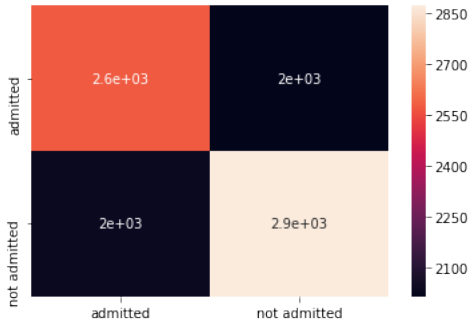


Figure 6. Confusion matrix for Logistic Regression with l2 penalty

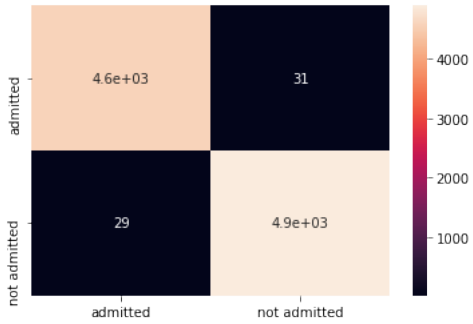


Figure 7. Confusion matrix for Random Forest Classifier with 15 trees

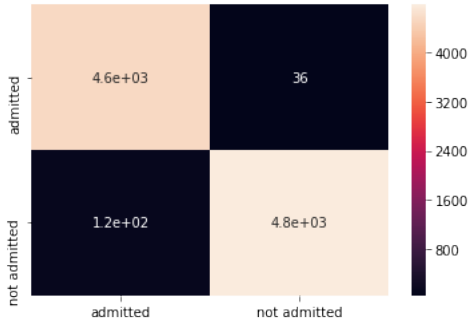


Figure 8. Confusion matrix for Random Forest Classifier with 10 trees


Figure 9. Confusion matrix for MLP with 1000 iterations

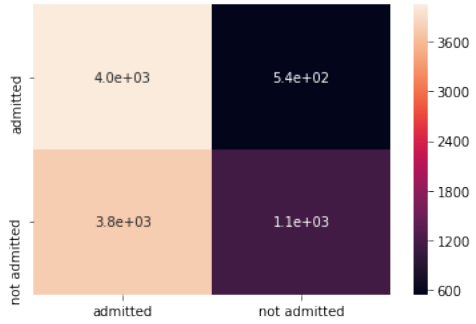


Figure 10. Confusion matrix for MLP with 2000 iterations

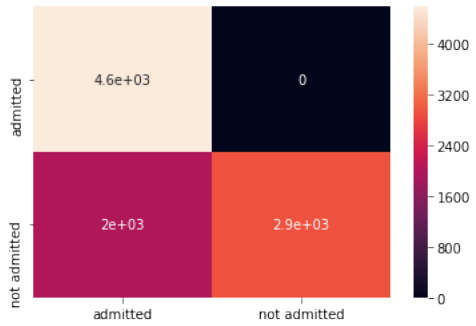


Figure 11. Confusion matrix for KNN with 2 neighbours

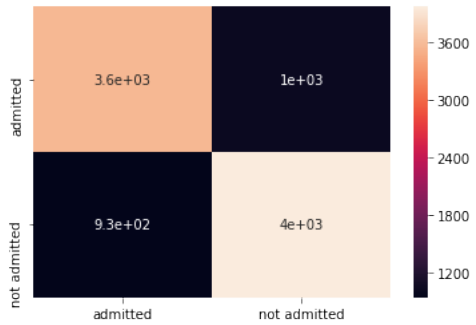


Figure 12. Confusion matrix for KNN with 3 neighbours


Figure 13. Confusion matrix for SVM with RBF kernel

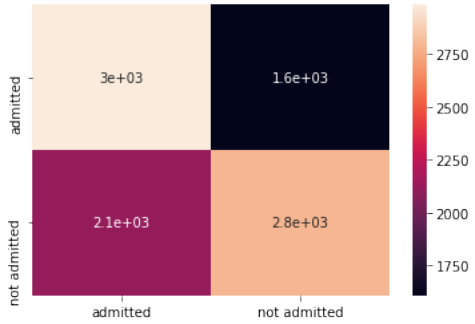


Figure 14. Confusion matrix for SVM with Polynomial kernel

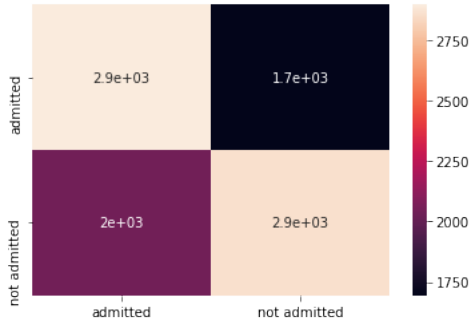


Figure 15. Confusion matrix for SVM with Linear kernel

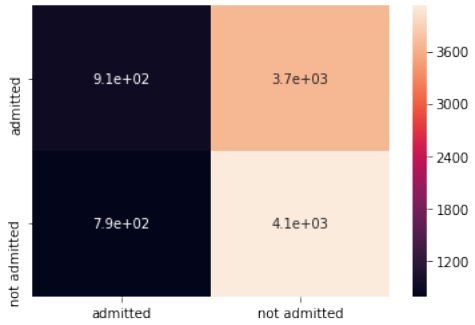


Figure 16. Confusion matrix for Bernoulli Naive Bayes

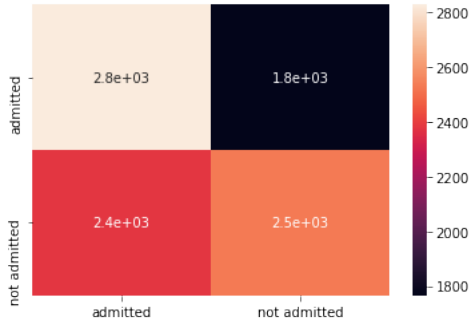


Figure 17. Confusion matrix for Complement Naive Bayes

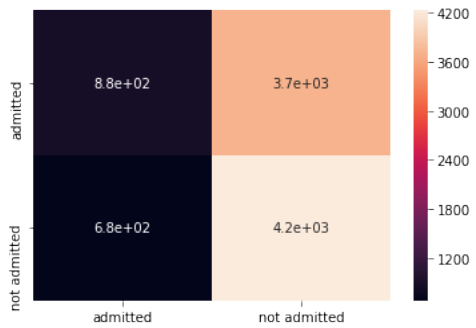


Figure 18. Confusion matrix for Gaussian Naive Bayes

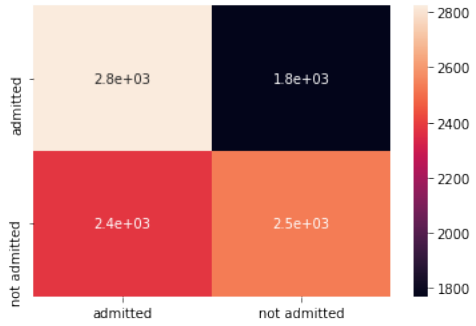


Figure 19. Confusion matrix for Multinomial Naive Bayes

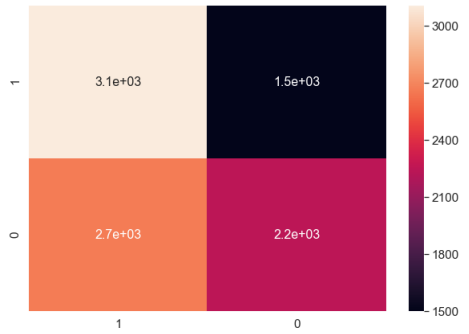


Figure 20. Confusion matrix for Tensorflow Neural Network

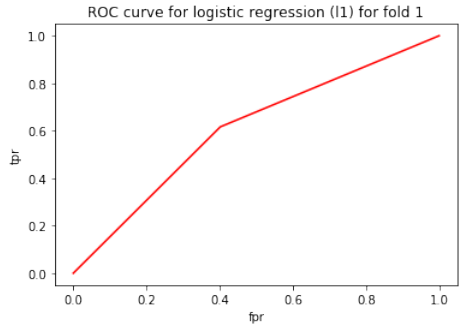


Figure 21. ROC curve for Logistic Regression with l1 penalty

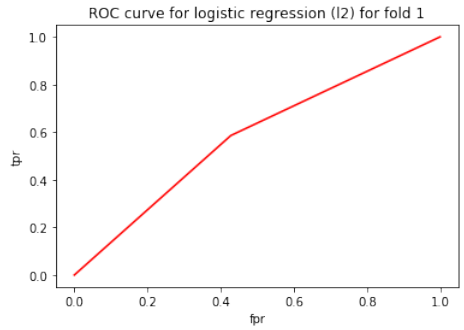


Figure 22. ROC curve for matrix for Logistic Regression with l2 penalty

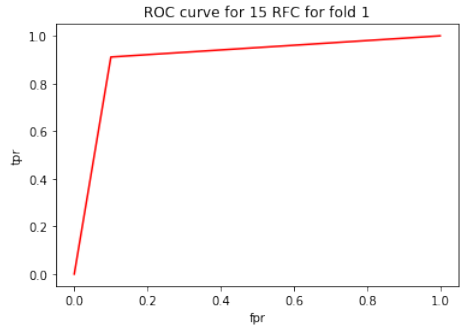


Figure 23. ROC curve for Random Forest Classifier with 15 trees

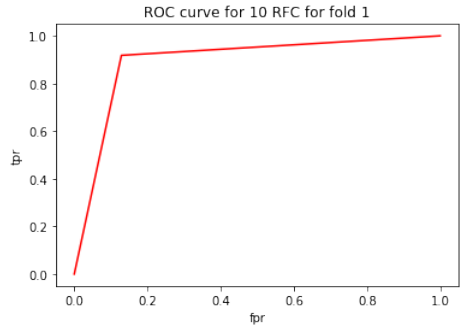


Figure 24. ROC curve for Random Forest Classifier with 10 trees

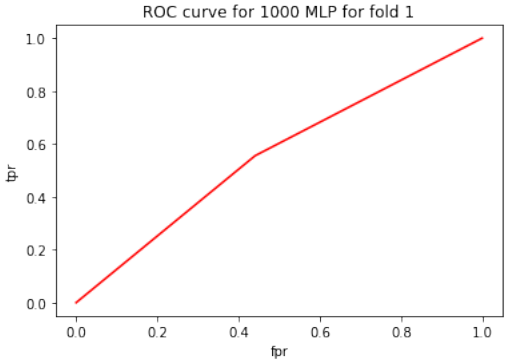


Figure 25. ROC curve for MLP with 1000 iterations

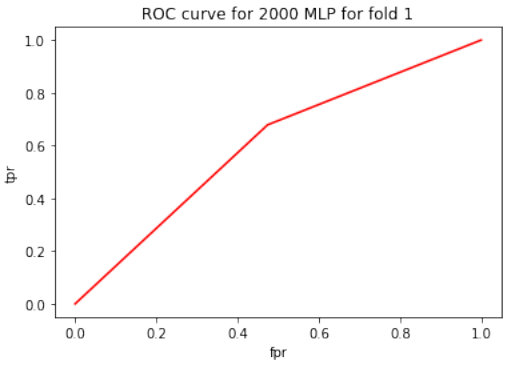


Figure 26. ROC curve for MLP with 2000 iterations

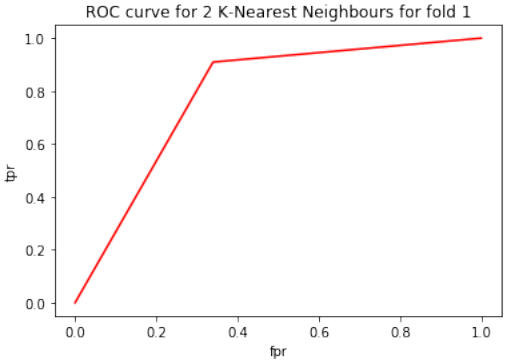


Figure 27. ROC curve for KNN with 2 neighbours

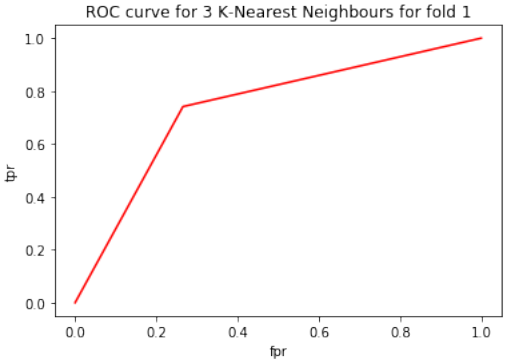


Figure 28. ROC curve for KNN with 3 neighbours

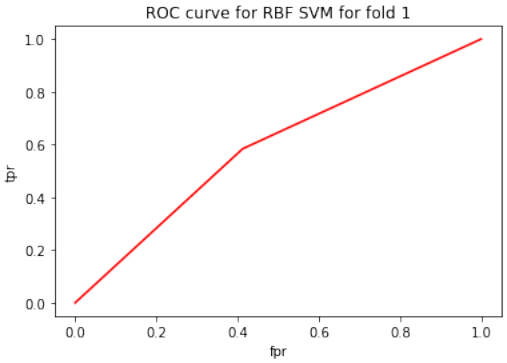


Figure 29. ROC curve for SVM with RBF kernel

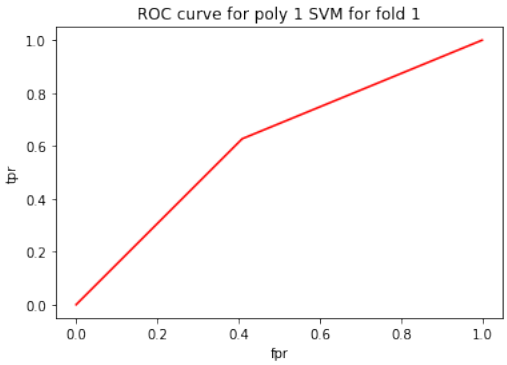


Figure 30. ROC curve for SVM with Polynomial kernel

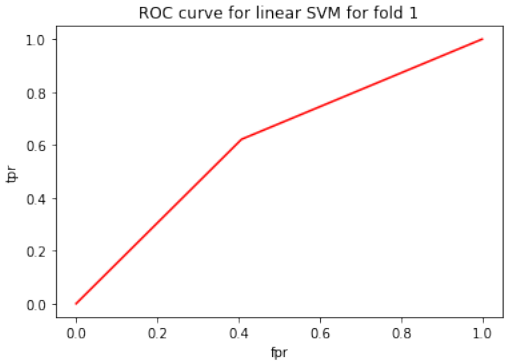


Figure 31. ROC curve for SVM with Linear kernel

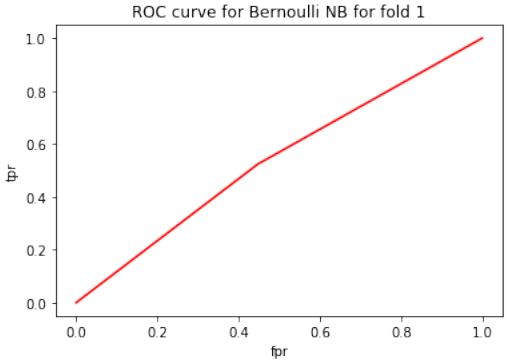


Figure 32. ROC curve for Bernoulli Naive Bayes

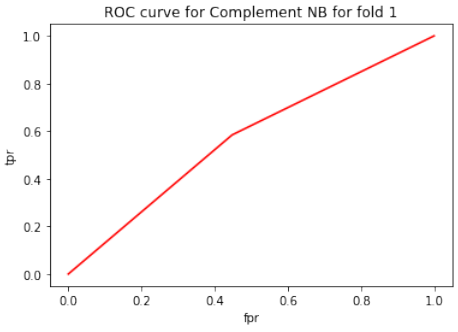


Figure 33. ROC curve for Complement Naive Bayes

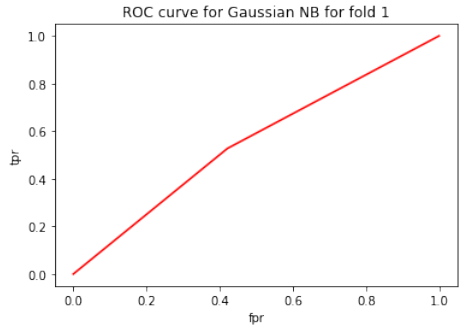


Figure 34. ROC curve for Gaussian Naive Bayes

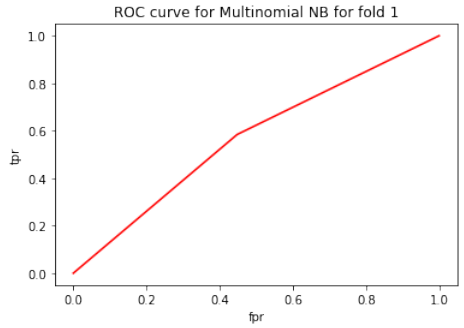


Figure 35. ROC curve for Multinomial Naive Bayes

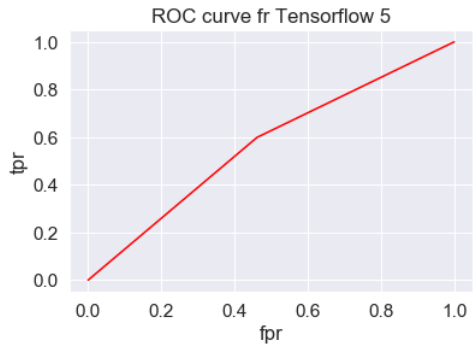


Figure 36. ROC curve for Tensorflow Neural Network