MOOC Dropout Predictions

Srivathsa S Rao* 4140852851 raosriva@usc.edu

Niroop R Sagar* 4897621292 ramdassa@usc.edu

Prashanth S Pujar* 5616456770 ppujar@usc.edu

Sushma M* 3939734806 mahadeva@usc.edu

Abstract

High dropout rate in MOOC is one of the major recurring issues. In this project, we address MOOC attrition using machine learning techniques for KDD Cup 2015 dataset. Feature engineering is performed on this dataset to generate categorical, time and course completion based features and avoid overfitting. We explored sampling methods to avoid the original class imbalance problem. KNN, Logistic Regression, Neural Network, Random Forest, Gradient Boosting and XGBoost algorithms are applied to obtain the predictions. We picked the best models out of these as estimators to the weighted voting classifier. Our model reports an accuracy of 87.74% and AUC of 87.97%.

1 Introduction

Massive open online courses (MOOCs) revolutionized education by delivering high-quality education to students located all over the world. However, one of the major issues is the high drop-out rate of students. Although many thousands of students enroll in these courses, the completion rate for most courses is very low [4]. The dropout rate can be reduced by studying MOOC dropouts and boosting the values of MOOCs. In this project, we are using machine learning techniques to predict MOOC dropouts using the 2015 KDD cup data. The task is to predict all enrollments who fail to complete the course based on the given data set. We do feature engineering to extract meaningful features for each model from the huge log data. Code implementation can be found at [1].

2 Data Description

The KDD Cup competition is associated with the annual Knowledge Discovery and Data Mining conference. The KDD Cup 2015 dataset contains multiple CSV files. Each file uses either 'enrollment_id' or 'course_id' as the key. 'enrollment_id' is an unique ID for each enrollment. An user can be enrolled in multiple courses and hence might be associated with many enrollment IDs. 'course_id' is an unique id associated with each course. The details of each CSV files are as follows:

- **log_train.csv** : This csv contains enrollment_id as they key and each entry in the column is an event performed by the student.

  event: The different types of event that might have occurred are:

  1. problem - problems or homework that are associated with the course
  2. video - watching course materials
  3. access - accessing other course material except for "video" or "problem"
  4. wiki - associated with course wiki
  5. discussion - access to discussion forum
  6. navigate - navigating within a course
  7. page_close - number of times the web page was closed while viewing the course
  8. chapter - chapters that are associated with each module

  time - timestamp of when the event occurred.

  source - This column tells whether the source of the event was from the browser or server.

  object - the module which students access or navigate to.

- **enrollment_train.csv**: This file also uses enrollment_id as key and gives general information about the users and their course enrollment.

* All authors have equal contribution

[1] https://github.com/srivathsa-rao/MOOC-Dropout-Prediction/tree/master/submissions

username: ID of the user, who is enrolled in a course

course_id: Course in which the user has enrolled

- **truth_train.csv**: This csv contains the label associated with each enrollment and enrollment_id is the key.

  label: This represents the "Label", indicating whether the user dropped the course or not. It can take values of either 0 or 1. 0 indicates the user did not drop the course and 1 indicates the user dropped out. Here, drop-out means the user does not stay in the course for more than 10 days.

- **date.csv**: Provides information on start and end date of a course with course_id as a primary key.

  from: Start date of the course

  to: End date of the course.

- **object.csv**: Each entry represents a module consisting of information about module release date, module category, associated course, and its children. Its a tree-like data structure with course divided into chapters, each chapter divided into various sections and each section containing information about videos, assignment problems, etc., course_id is the key representing the unique ID to which the module belongs to.

  module_id: An unique ID associated with each module. Each course can have multiple modules.

  category: Category of the course module.

  children: Containing information about children modules of the course module.

  start: Date when the module was released.

3 Feature Engineering

We have used log_data.csv as one of the main CSV's to generate data as this involves data about student's access to course materials. For training, this has about 4,880,382 rows of data for enrollment of 72,395(for which truth_train.csv has labels). We have extracted about 132 features in the below categories. We followed the same feature extraction techniques for the 24,013 enrollments in the test data. Once the features have been generated, we have dropped a few columns to arrive at better generalization of the built models.

3.1 Feature Generation

3.1.1 Time-based features

We have extracted the time spent on each browser or server event by the user in the course. This helps us to analyze the amount of time a user spends in accessing each course. If a user regularly spends more time on a course, it helps us to infer that the user will complete the course. So, we have also extracted the number of logs that are less than 60 seconds in any event.

This accounted for a total of 29 features for the given dataset.

3.1.2 Count-based features

Count Based features comprises of 40 columns, this is inclusive of :
Week based sessions count : The number of 30 minute user sessions doing different activities on the course modules.
Week based activity count : number of times a user shows an activity on a particular enrollment in a week.
Day based activity count : number of times a user shows an activity on a particular enrollment in a day.

These features help us to discriminate between users who access the course regularly and those students who access the course sporadically. Furthermore, these features account into classifying if a user will complete the course.

3.1.3 User course interactions

This set of features extracts information on the total number of enrolled courses(including the parallel enrollments) by a student and the number of students enrolled in each course. This helps in feeding the models with the appropriate data on the student to course relation. Higher enrollments in a course might convey that it is a popular course or an easy course and might have a high completion rate. Similarly, higher the parallel enrollments for a student might result in drop out of some courses from his/her enrolled courses.

To understand the complexity of the course and the interest of the student towards the course, we calculated the number of modules accessed by the student in a course and to what percentage does

his access compounds to when all the modules in the course are considered.

This accounted for a total of 5 features for the given dataset.

3.1.4 Category Based Features

Extracted various features based on the category of modules. Features like how many times a video was accessed, how many times a user participated in the discussion, chapter count and many other features like sequential, course_info, combinedopenended, static tab duration, problem module access count, etc were extracted. So, using these features can find a relation to how static tabs influence the drop rate? How video accessing count will affect the drop rate of an enrollment?

This accounted for a total of 15 features for the given dataset.

3.1.5 Completion based features

Two sets of completion features are extracted related to the overall completion of courses and the individual student completion rate based on the label of the training data. This helps in deducing the courses which have high completion rates and as well as student's behaviors with respect to enrolled course completions. We impute this data collected from the training to the test data to infer these behaviors.

This accounted for a total of 3 features for the given dataset.

3.2 Dropped Features

There were certain features dropped because of overfitting issues or due to close correlation with other features.

3.2.1 Overfitting Issue

We extracted the 'completion_rate/drop_rate' feature using the label, which is the course completion history of a user. Based on the history of courses taken, the course completion rate will be calculated for each user. If completion_rate = x, drop_rate will be 1-x. For example, if the user has taken 5 courses previously and has completed only 2 courses, then his completion_rate will be (2/5)*100, which equals 40%. This feature is added in the test data, by checking the previous completion in train data(if the user has taken any course at all), however, if the user hasn't taken any courses before, we imputed the column the value of 50% or 0.5.

```
auc: 0.5916055306689147
accuracy: 0.49323283221588304
logloss: 17.50315741772447
              precision    recall  f1-score   support

           0       0.25      0.76      0.38      4902
           1       0.87      0.43      0.57     19111

    accuracy                           0.49     24013
   macro avg       0.56      0.59      0.48     24013
weighted avg       0.75      0.49      0.53     24013

Confusion matrix, without normalization
[[ 3715  1187]
 [10982  8129]]
```

Figure 1: Overfitting Issue

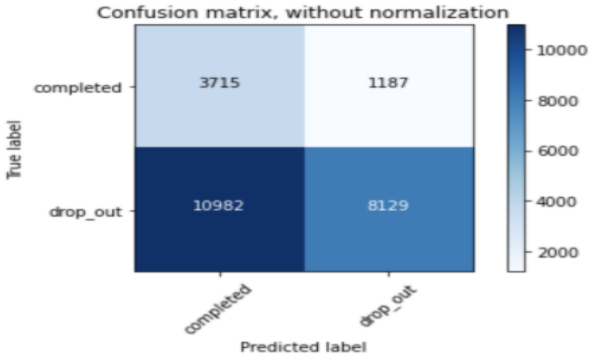


Figure 2: Overfitting Issue

We plotted the heat map and feature importance graph to understand how this feature correlates to the label and also the importance of this feature for the model.

Observations:

1. We were able to achieve a pretty good F1 score, the Area Under the Curve(AUC) of Receiver Operating Characteristic (ROC) Curve and accuracy for the training data as seen in Fig.3 compared to other models that didn't include this column.

2. In-sample error was very less.

3. This feature had a really good importance as seen on the feature importance graph.

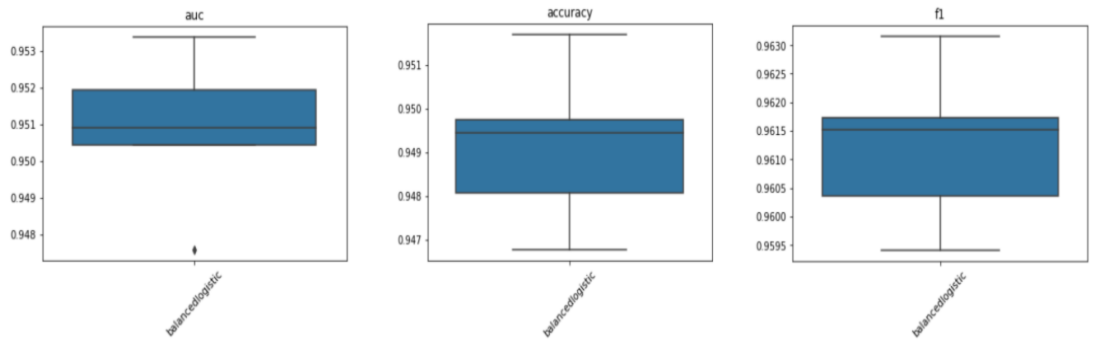


Figure 3: AUC, Accuracy & F1 Score with Completion Rate

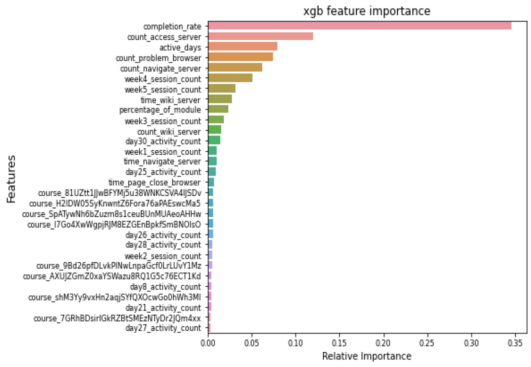


Figure 4: Feature Importance

However, when we used this model on the test data, it did not generalize well. The area under the curve, F1 score, and accuracy for the test data can be seen in **Figure 1** and **Figure 2**. Even though the in-sample error was less and the feature had a good importance score, out of sample error was very high. The 'relative' importance of the feature compared with other features was very high, making the model over-dependent on this feature and causing the model to overfit. So, in test data for a username which was not present in train data, there was a close to 50% chance of predicting either way.

3.2.2 Other Issues

1. **Correlated Features**

   Based of correlation between certain similar features we have removed few columns:

   In **Figure 5A**, the number of modules in a course and the percentage of modules are highly correlated. we removed the number of module feature. We see better results after the removal of the number of module feature.

   In **Figure 5B**, parallel enrollments of user and number of courses taken by the user are highly correlated. Based on the correlation of these to labels, we deleted parallel enrollments from the data.

   In **Figure 6**, the individual week activity count is closely correlated to week session count. So, we have removed the week activity count from the data (as week session count is closely related to label than week activity count).

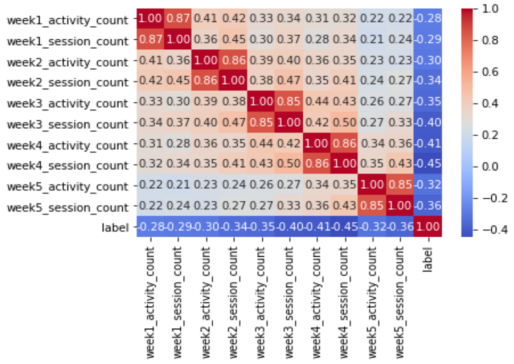


Figure 6: Correlation map

2. **Biased Features:** The 'enrollment_id' feature is used as an identification feature in the data to merge different csv's to extract data. We have de-identified the data to remove the bias that might be due to this enrollment id.
3. **Features with NULL data:** Deletion of categories of modules which has no events performed on them. We have 15 different categories of modules for each course. Out of these 15, we have student logs for only 5 categories of modules in log_data.csv. So, we have removed the 10 columns for which the values are zero in both train and test data.

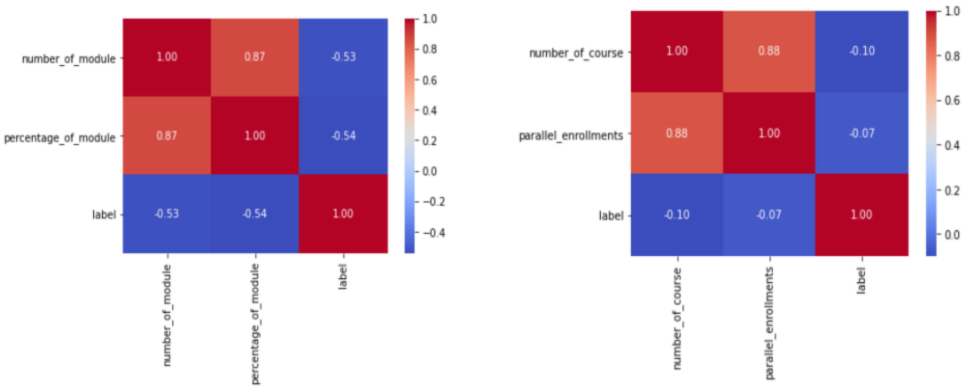


Figure 5: Correlated Features Figure A, Figure B

4 Sampling

We used a dummy classifier provided by sklearn on train data and we observed the results shown in **Figure 7** and **Figure 8**.

```
auc: 0.5
accuracy: 0.7924027902479454
logloss: 7.170319602029022
Confusion matrix, without normalization
[[    0 15029]
 [    0 57366]]
```

Figure 7: Dummy Classifier

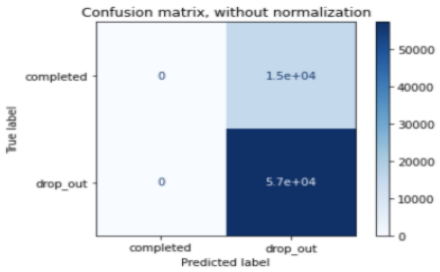


Figure 8: Dummy Classifier

In the training dataset, which consists of 72,395 entries, 57,366 were 1's (dropped out of the course). So, the majority of the label consisted of 1's. This is not the actual representation of the real-world data as it is skewed. If test data consists more of 1's then we will end with very high accuracy, but if the test data contains 0's since our model hasn't seen much of this class we will end with a bad accuracy, and our model will produce a really bad out-of-sample error in general. So, to handle this class imbalance we evaluated two **sampling approaches**.

Here, we sample the data of different classes, either by increasing the minority class - **Upsampling** or by decreasing the majority class - **Downsampling**. However, there are problems with both approaches.

- The problem with upsampling is that the data sampled will be random with replacement, as a result, there will be chances of duplicate entries.
- And the issue with downsampling, where we reduce the majority class (here 1's) to minority class is we will end with less data for training.

So, in order to handle these two issues we resorted to the Synthetic Minority Oversampling Technique (SMOTE) sampling approach[2].

We have analyzed two specific techniques using SMOTE for the MOOC dataset. This helps mitigate the problem of duplicate samples when up-sampling the minority class in random sampling. Here, it synthesizes the examples of minority class from the given data set and thus augmenting new data, helping the model have enough data points of both the classes. The new data is generated by choosing one of the K nearest neighbors of a given data point, subtracting the 2 points(the point and one of K neighbors) in that feature space and multiplying this value with a random value between 0 and 1. This would be a point on the straight line between the 2 points.

We evaluated the performance of models with up-sampling the minority class using SMOTE and as well as a combination of up-sampling the minority class and down-sampling the majority class, which achieved better classifier performance. This involved generating the minority class to a certain percentage of the existing majority class(sampling_strategy = 0.4) and then generating(reducing) the majority class by randomly removing certain data points till it converges to a certain percentage of the up-sampled minority class(sampling_strategy = 0.5).

5 Evaluation Metrics

Since we had a class imbalance, we considered other metrics like precision, recall, f1 score, and area under the curve(AUC) and not just accuracy. Even for grid search, an exhaustive search used for tuning the hyper-parameters of an estimator, we passed the metric as 'AUC' and not accuracy which is the default value passed.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

Precision talks about how accurate your model is. That is, out of those predicted positive values, how many of them are positive values.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

Recall calculates how many of the Actual Positives our model captures through labeling it as Positive. Let's say, if a fraudulent transaction is considered non-fraudulent, then the cost of this misclassification is extremely high.

$$F1score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Area Under the Receiver Operating Characteristics (AUCROC): This tells how our model is distinguishing 0's as 0's and 1's as 1's. So, the higher the value, the higher is the rate our model is predicting properly.

So, when we did not use any sampling, all of our classifiers produced a bad precision (between 0.5 and 0.6) for the completion rate(0's - minority class) compared to when we used SMOTE, our precision increased to a value between 0.7 and 0.8.

6 Modeling

6.1 K-Nearest Neighbors

KNN is a lazy learning algorithm. There is no assumption for underlying data distribution in this algorithm. K is the number of nearest neighbors and it is the core deciding factor. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its K nearest neighbors. For the given dataset we tuned the following: n_neighbors=5,weights='distance',algorithm='auto.

6.2 Logistic regression

Logistic regression is a binary classification algorithm. This computes regression coefficients corresponding to a decision boundary. We use the derived regression coefficients to compute the outcome for the new data and this regression value is passed through a sigmoid to predict a binary outcome.

We have used the class_weight as balanced for the algorithm to adjust weights inversely proportional to class frequencies in the data. This helps in getting better precision and recall for the minority class. After the grid search, we kept the regularization strength value at 0.01, tolerance for stopping criterion as 100.

6.3 Support Vector Machine

SVM partitions data into two classes using a separating hyperplane. The hyperplane maximizes the gap on either side, to minimize the misclassified data in the test set. On either side, the equidistant points closest to the hyperplane i.e support vectors hold the hyperplane. This model is useful in use cases like this where we have non-linear relationships.

We have analyzed different kernel functions for SVM like linear, RBF, and polynomial Kernels of higher degree and found out RBF performed better. We used the regularization strength as 1 and tolerance for stopping criterion as 0.01. These values were deduced out of the grid search done over a range of parameters.

6.4 Neural Network

Neural networks are multi-layer networks of neurons that are used to classify things, make predictions. Initially started with the basic settings of max iterations being set to 200, relu for activation, one hidden layer with 100 neurons, Adam

as a solver, and a regularization parameter set to 0.0001. With the above-mentioned settings, we were able to obtain the weights but weren't good as the model didn't converge for 200 iterations. So, we had to tune the hyperparameters using a grid search, an exhaustive search technique used to tune the hyperparameters to obtain the optional values: max_iter=1000, activation=relu, alpha=0.0001, hidden_layer_sizes=(100, 150, 100), solver=sgd, random_state=0.

So, using this Grid Search Technique we were able to experiment with different parameters of a neural network like the number of hidden layers, number of neurons in a particular layer, regularization parameter, number of epochs, different activation functions like logistic, etc.

6.5 Random Forest

Random forest algorithm uses a large number of individual decision trees that operate as an ensemble. Each tree predicts a class and the class with the most votes becomes the final prediction. We have tuned the hyperparameters using grid search and inferred the optimal values:n_jobs= -1, max_depth=5, min_samples_split= 5

6.6 Gradient Boosting

The common ensemble techniques like random forests rely on simple averaging of models in the ensemble. Boosting is based on a different, constructive strategy of ensemble formation and the main idea of boosting is to add new models to the ensemble sequentially. At every iteration, a new base-learner model is trained concerning the error of the whole ensemble learned so far. Gradient boosting Decision Trees (GBDT) is one of the most powerful techniques which uses ensemble tree learning algorithms to build predictive models. GBDT uses a gradient descent approach to minimize the error to train new trees. In this project, we have tuned the hyperparameters and used these values to train our model: learning_rate=0.1, n_estimators=150,max_features='auto', min_samples_split= 2

6.7 Extreme Gradient Boosting(XGBoost)

XGBoost is a highly optimized version of gradient boosting. It is a scalable end-to-end tree boosting system that builds base-learners parallelly and employs a backward tree pruning strategy with *'max_depth'* as the reference parameter. The algorithm is also sparsity-aware, i.e., XGBoost learns best missing value depending on training loss and detects sparsity patterns efficiently. XGBoost is also enhanced using L1 and L2 regularization to avoid overfitting. Furthermore, XGBoost is hardware optimized by building cache awareness to store gradient statistics.[3]
For the given sampling strategy we have tuned the hyperparameters to the following values: learning_rate=0.3, max_depth=4 and gamma=0.5.

7 Results

We have evaluated the individual models and selected few models which performed well based on the defined metric. These selected models were used as part of voting classifier to give better predictions.

7.1 Individual Model Performance

We evaluated the results of individual classification algorithms before using them as estimators for the voting classifier. We used the metrics as discussed in section 5 for the evaluation of the models. First, we used the models with different parameters in a pipeline and performed K-Fold cross-validation. Based on the cross-validation results, we tuned the hyperparameters to reduce the in-sample error. Then, we evaluated the tuned models using the learning curves to determine the overfitting of each model. The results of each model are seen on a heatmap to see the variance in the results before selecting the models for the voting classifier.

7.1.1 Cross-Validation Results

We have done a 5-fold validation on the training data using the metrics defined in section 5. Based on the results obtained, we did a grid search over the parameters for individual models to obtain the best model. The metric which we tried to improve during the grid search is Area Under Curve of ROC curve. The resultant parameters were set to individual modes and we did the cross-validation again to see the improvement in the results. You can refer to **Figure 9** for the variation in the results of various metrics for the different folds.

From the results above, we can infer that the boosting algorithms like gradient boosting and XGBoost performed better than other algorithms concerning any metric. For the boosting algorithms the AUC was as close to 84% [**Figure 10**]

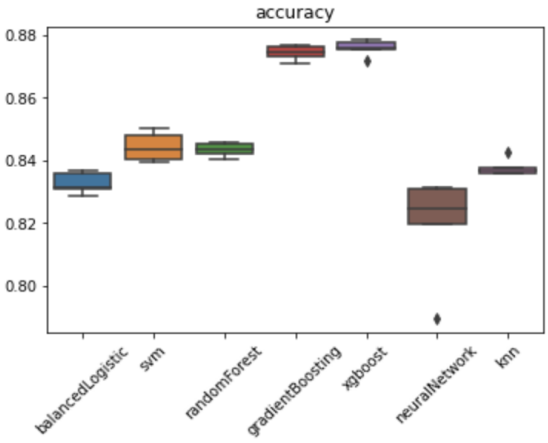


Figure 9: Cross Validation Accuracy

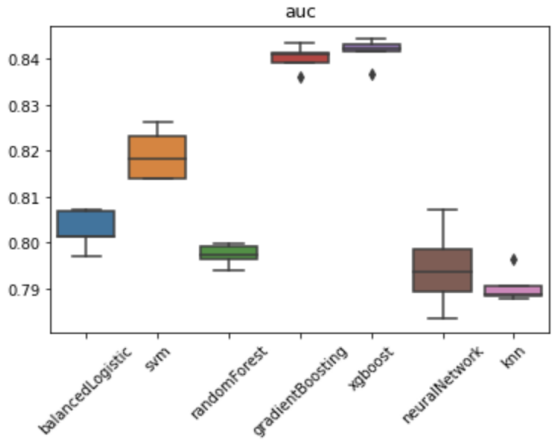


Figure 10: Cross Validation AUC

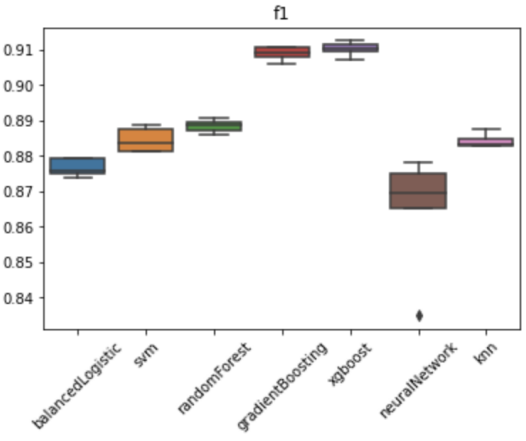


Figure 11: Cross Validation F1 Score

and F1 score was close to 91% [**Figure 11**]. Similarly, KNN and neural network's performance was bad whose AUC was as low as 79% [**Figure 10**]. We will consider the learning curves and result variation before eliminating any of these models being considered for the voting classifier.

7.1.2 Learning Curves

We have plotted the rate at which the classification error varies for training and validation sets. We have used a 4 fold cross-validation in the learning curve function of sklearn. We haven't done shuffling here as it is done at the sampling stage. The data obtained from this helps us identify the model with high bias (we would have this because of the class imbalance of training examples even after shuffling), and high variance. The high variance would result in overfitting of models which can be inferred from the huge difference between the training and validation curve. This is because of the saturation in the improvement of error after a few data points.

The learning curve for the individual model with different data points can be seen in **Figure 12** - **Figure 18**.

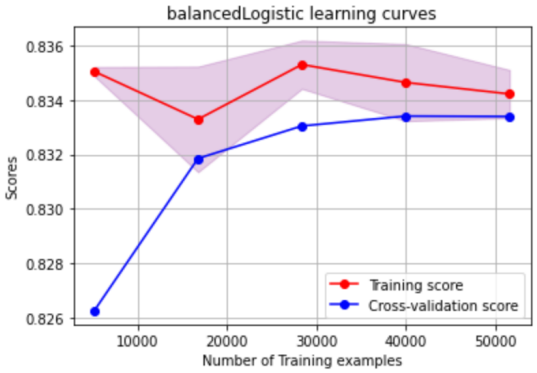


Figure 12: Learning Curve for Logistic Regression

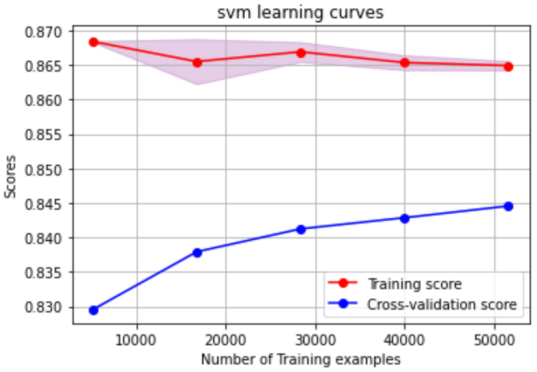


Figure 13: Learning Curve for SVM

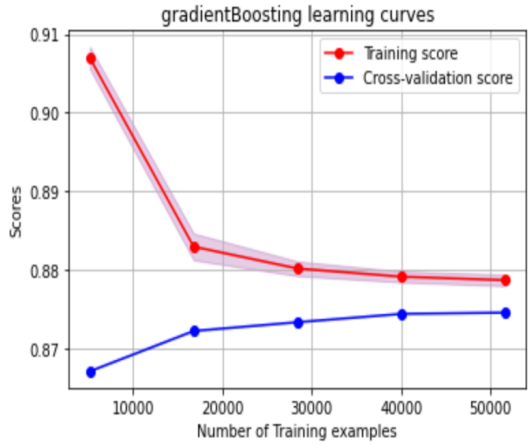


Figure 14: Learning Curve for Gradient Boosting

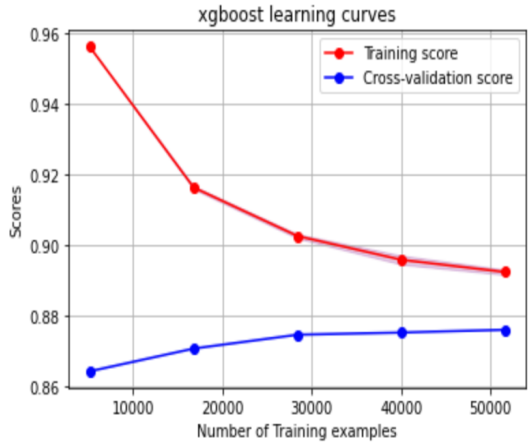


Figure 15: Learning Curve for XGBoost

Based on the learning curves of the models, we can observe that neural network [**Figure16**], SVM [**Figure 13**] and KNN [**Figure 18**] has high variance(since the validation and the training error doesn't decrease with the increase in the number of the examples). On the contrary other models like Gradient Boosting [**Figure 14**], XGBoost [**Figure 15**] and Random Forest [**Figure 17**] show the feasibility of generalization of the built models.

7.1.3 Comparison of Results of Models

We have plotted the correlation of each of the models predicted results of the test data set. This helps in identifying the variations that the individual models bring in for the voting classifier. The models which are highly correlated can be not included into the voting classifier as they correspond to the same result. You can see the correlation matrix in the figure below.

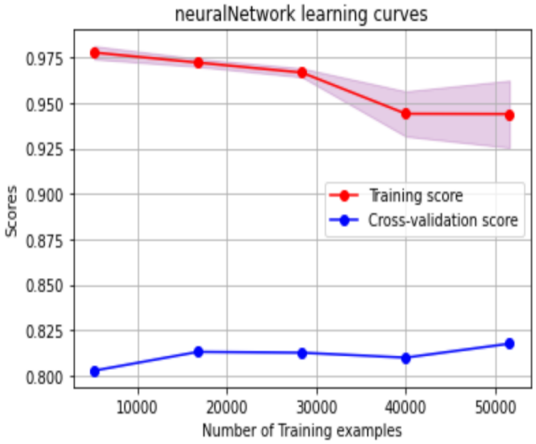


Figure 16: Learning Curve for Neural Network

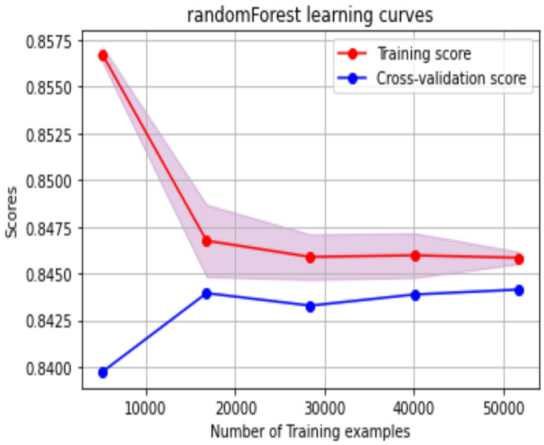


Figure 17: Learning Curve for Random Forest

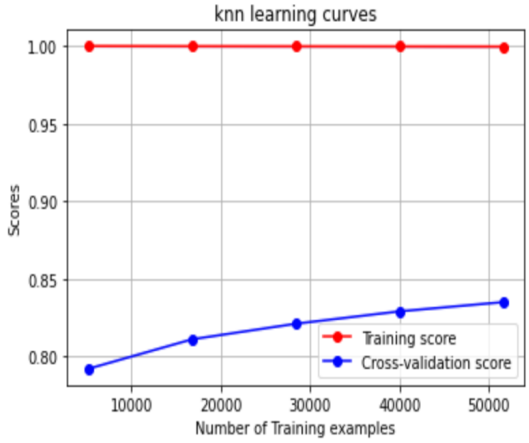


Figure 18: Learning Curve for KNN

Table 1: Comparison of Individual Model Results

Model	AUC[%]	F1[%]	Accuracy[%]	Precision[%]	Recall[%]
Logistic Regression	87.12	91.2	85.91	92	90
Gradient Boosting	87.94	92	87.66	95	90
XGBoost	87.84	93	87.82	95	90
Random Forest	87.12	92	87.18	91	94
SVM	84.18	89	82.89	93	85
Neural Network	75.95	74	64.57	91	62
KNN	79.48	91	84.9	88	94

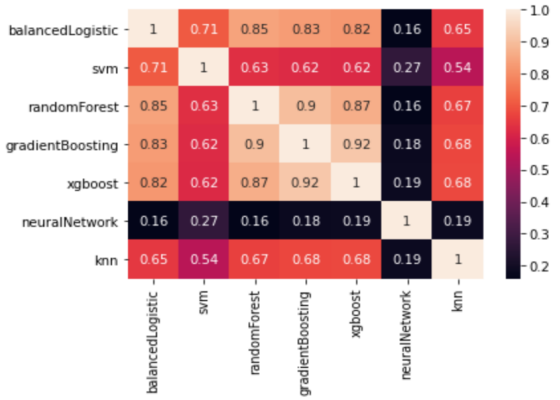


Figure 19: Correlation Matrix of Models

The **table 1** shows the evaluation metrics of individual models. We can observe the scores of KNN, SVM and neural networks are not as high as the other 4 models.

Based on the above metrics, correlation matrix, learning curves and validation scores, we can clearly see that the SVM, KNN and neural network don't perform well on this data set. We have removed these models from the voting classifier and used the remaining models in the next phase.

7.2 Voting Classifier Performance

As a final step in our model, we have used sklearn's voting classifier to ensemble the results produced by individual hyper-parameter tuned classifiers. It aggregates the prediction of each such classifier passed into the Voting Classifier and predicts the output class based on the highest majority of voting.

Soft voting method is used for the voting classifier so that the output class is the prediction based on the average of probability given to that class. In addition, we have also inculcated weights to the estimators so that all the participating estimators do not have equal importance. Also, we have seen that such weighted voting classifiers have performed better over vanilla voting classifiers. In our model, weights for the four participating estimators Logistic Regression, Gradient Boosting, Random Forest and XGBoost are set to = [2, 1, 3, 4] respectively.

7.2.1 Accuracy and ROC Curve

Individual estimators are fit on the training data and the results produced by these estimators on the test data, were averaged and final prediction is given out. This model obtained an accuracy of 87.8 and a AUC score of 87.9, as shown in **Figure 20** and **Figure 21**.

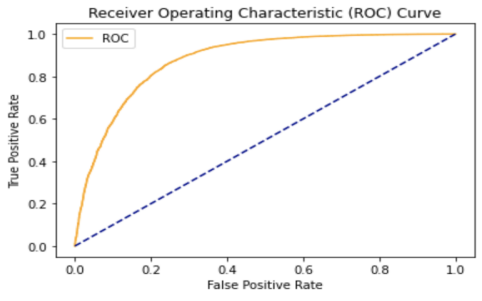
```
accuracy: 0.877483030025403
logloss: 4.231646301069891
              precision    recall  f1-score   support

           0       0.73      0.63      0.68      4902
           1       0.91      0.94      0.92     19111

    accuracy                           0.88     24013
   macro avg       0.82      0.79      0.80     24013
weighted avg       0.87      0.88      0.87     24013

Confusion matrix, without normalization
[[ 3098  1804]
 [ 1138 17973]]
Normalized confusion matrix
[[0.63 0.37]
 [0.06 0.94]]
```

Figure 20: Voting Classifier Accuracy



AUC-ROC : 0.8797598596240166

Figure 21: Voting Classifier AUC ROC

7.2.2 Confusion Matrix

Figure 22 refers to the confusion matrix obtained from the Voting Classifier predictions.

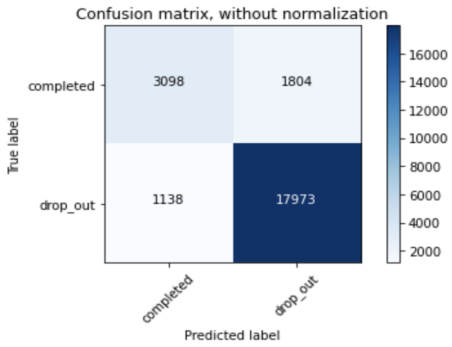


Figure 22: Voting Classifier Confusion Matrix

8 Conclusion

In this project, we aimed at building a model to predict the MOOC drop rate by using the ensembling modeling technique. After model selection, we used Grid Search to tune the hyperparameters of each model and plotted the relevant graphs to understand the behavior. Once done with hyperparameter tuning, we put these models to a voting classifier to obtain the final ensemble model. And, we were able to achieve really good results. The AUROC of our model was 87.97%, which was 2.7% less than the actual KDD cup winners and we were also able to achieve pretty good out sample error.

9 Division of Labor

Throughout the project, each member explored different approaches to solve this classification problem. Each member of the team has contributed equally as mentioned in the title of the report. We can summarize the contributions as below:

- Srivathsa - Generated time-based features for the given test and train data. Implemented KNN, Logistic Regression, and SVM algorithms and performed grid search to tune the hyperparameters.
- Niroop - Generated count-based features for the given test and train data. Implemented the XGBoost algorithm and performed grid-search to tune the hyperparameters.
- Prashanth - Generated category-based and completion-based features for the given test and train data. Implemented a Neural Network and performed grid-search to tune the hyperparameters.
- Sushma - Generated user course interaction features for the given test and train data. Implemented Random Forest and Gradient Boosting algorithms and performed grid-search to tune the hyperparameters.

Finally, the report was also split equally among all the four members.

References

[1] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, Jun 2002. ISSN 1076-9757. doi: 10.1613/jair.953. URL http://dx.doi.org/10.1613/jair.953.

[3] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016. URL http://arxiv.org/abs/1603.02754.

[4] W. Li, M. Gao, H. Li, Q. Xiong, J. Wen, and Z. Wu. Dropout prediction in moocs using behavior features and multi-view semi-supervised learning. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3130–3137, 2016.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[7] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. L-bfgs-b - fortran subroutines for large-scale bound constrained optimization. Technical report, ACM Trans. Math. Software, 1994.