CS229 Project Report

MOOC Dropout Prediction

Zixun Yang
jasonyzx@stanford.edu

***Abstract* — In this project, I built model to predict dropout in Massive Open Online Course(MOOC) platform, which is the topic in KDD cup 2015. Different from full dataset in KDD, I only had partial dataset (36% enrollments). With my feature engineering result on this complicated three-dimensional dataset, I first explored different models and optimized parameters selection to reach best performance. With few models of good prediction on validation data, I ensembled them together using XGBoost Classifier to do a second level learning based on the first level training prediction. In addition, I implemented Long Short Term Memory(LSTM) Recurrent Neural Networks with Keras on 30 days univariate Time series data and reached 0.857.**

I. Introduction

Massive Open Online Course (MOOC) has been revolutionizing the way people getting education. However, it also raises up concern that MOOC has very high dropout rate relative to traditional classes. An accurate prediction of dropout becomes very important because it can help MOOC course developers to adaptively tune web designs to students with high dropout rate.

II. Related work

Student dropout prediction has been tapped using machine learning technique with different approaches. Dekker et al. (2009) [1] used decision trees algorithm to predict university students' dropout based on their behavior features extracted from their previous courses. However, this detailed information is usually missing for MOOC students. On the other hand, MOOC course has its own advantage on recording students' behavior: every single behavior is done online and thus tractable. Monn et al. (2014)[4] used natural language processing techniques to analyze student's post in the course discussion forum. Yang et al. (2013) [5] used NLP to predict whether a question posted in the discussion forum is resolved or not. But none of the NLP based work directly predicted dropout. Kim et at. (2014) [3] analyzed students' watching video pattern to do the dropout prediction. These patterns include skipping, zooming, playing, panning, pausing and quitting. They use a small dataset because it is very demanding to extract these behavior patterns.

Liang et at. (2015) [6] has published their work based on the same dataset. They reached 89% accuracy in dropout prediction task with gradient boosting decision tree model. The winning solution of KDD Cup 2015 used multi-stage model ensemble to do the prediction. They achieved the accuracy of 90.92%. The only write up of their work is a brief white paper. They announced that they trained 64 classifiers with 8 different algorithms and different subsets of extracted features; then they blended predictions of classifiers with the multi-stage ensemble.

III. Dataset and features

A. Dataset Description

The data comes from online open source. There are totally three major files used for training,

- enrollment_list.csv:

each line is a course enrollment record with an enrollment_id E, a user_id U and a course_id C, indicating that U has enrolled in C.

- activity_log.csv:

each line is a behavior record called "event". Each event contains the following information: enrollment_id, time, and event. Here event can be as follow,

problem - working on course assignments
video - watching course videos.
access - accessing other course objects except videos and assignments
wiki - accessing the course wiki
discussion - accessing the course forum
navigation - navigating to other part of the course.
page_close - closing the web page

- train_label.csv:

each line is a dropout record of the enrollments in the test set, which you are required to predict.

B. Dataset Preprocessing

Different from entire dataset of in KDD cup 2015 (8,157,278 logs from 120,543 enrollments as training and 5,387,848 logs from 80,363 enrollments as test), I only get partial dataset (36% enrollments compare to original dataset) and I split it into 46,288 enrollments as train set, 11,572 enrollments as dev set and 14,465 enrollments as test set. As observed from dataset, the maximum length of one course is 30 days. And with course period of 30 days, I process dataset and divide them into three subsets with different information in 30 days per each enrollment ID,

- count for each of seven events frequencies in 30 days (30x7)
- count for total events frequencies in 30 days (30x1)
- total consuming time each day in 30 days (30x1)

In addition, I also have count and time for each week of 30 days. Since there are only 39 types of courses in total, I treated it as a categorical feature. Thus, I one-hot encoded course type. For each student, course type feature is a vector of length of 39.

I also extracted a few other features of interests. For example, unique event count indicates how many types of activity this student has conducted during the whole learning process. A student how has never used the discussion panel might have higher chance of dropout than a student who are making fully use of all the functions in the MOOC. See Table 1 for feature summary.

Activity-count (Numeric)				
Feature Description	Overall Count By day	Overall Count By Week	Total Overall Count	Total Count for 7 types
Number of Features	30	4	1	7
Online-Time (Numeric)				
Feature Description	Overall Count By day	Overall Count By Week	Total Overall Count	Last - First Time (in days)
Number of Features	30	4	1	for each activity 7
Other Features				
Feature Description	One-hot Encoded Course Type(Categorical)		Unique Activity	Active Days
Number of Features	39		1	1

Table 1 *Feature Summary*

C. Feature Selection

As I have extracted about 320 features from the dataset, I need to do the feature selection, not only for training speed, but also improve the prediction accuracy. Basically, what I did in this stage was separating the features in each enrollment into two sets, with drop and not drop case. I plotted the bar plot and bivariate kernel density estimate plot.

For bar plot, I can see the bar size difference between drop and not drop case in specific features. With similar bar size between drop case and not drop case, it means this feature would not differentiate drop case from not drop case and vice versa. I mainly observed daily-base data in bar plot in 30 days length. For example, Figure 1 shows the average daily total event frequency count and total online time in each day of 30 days course length on drop (blue) and not drop (red) case. As you can see in both plots, besides day one, data in every other days have large size difference.

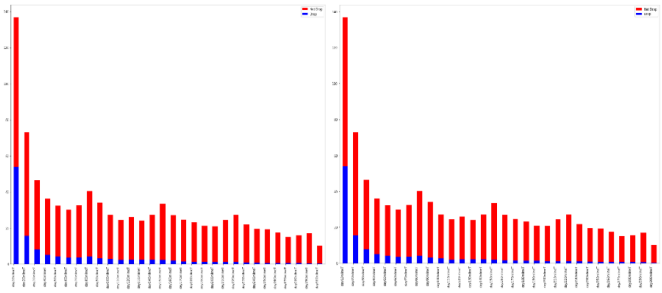

Figure 1 *bar plot of daily event count and daily online Time between drop(blue) and not drop (red)*

I also came out with bar plot about average count for each of 7 events in each day in 30 days period between drop and not drop case. I came to similar conclusion as shown above.

In addition, for kernel density estimate plot, I can see the density distribution in specific feature between drop case and not drop case. If it is a good feature, peak in the plot between drop and not drop case will be different. The larger the gap between peak of drop and not drop, the better the feature. I mainly observed other general features total or average in 30 days. For example, as you can see from the below Figure 2, two features, total days in 30 days with access activity and average event frequency count in 30 days, has different density distribution. It is clearly that access_days here would be a better indicator for enrollment drop than average event count.

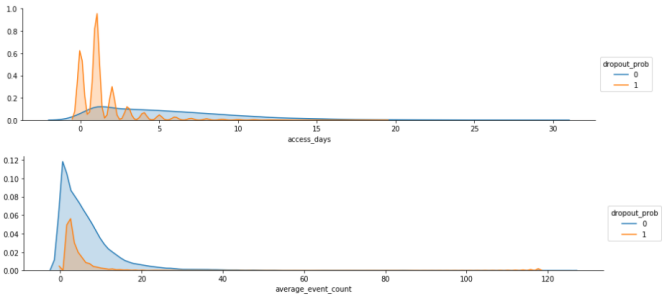

Figure 2 *Histogram for features access_days and average_event_count in drop and not drop case*

IV. METHODS

A. LSTM

An LSTM layer consists of a set of recurrently connected blocks, known as memory blocks. [8] These blocks can be thought of as a differentiable version of the memory chips in a digital computer. [8] Each one contains one or more recurrently connected memory cells and three multiplicative units – the input, output and forget gates – that provide continuous analogues of write, read and reset operations for the cells. [8]

One shortcoming of conventional RNNs is that they are only able to make use of previous context. [9] Bidirectional RNNs (BRNNs) do this by processing the data in both directions with two separate hidden layers, which are then fed forwards to the same output layer. [9] Combining BRNNs with LSTM gives bidirectional LSTM, which can access long-range context in both input directions. [9]

By taking advantage of LSTM and BRNN, I built LSTM model with following framework,

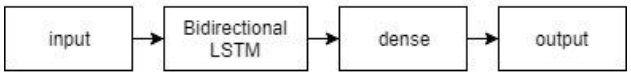

Figure 3 *LSTM framework*

B. Model ensemble learning

Ensemble learning helps improve machine learning results by combining several different models. It would help decrease variance (bagging), bias (boosting), or improve predictions (stacking). [10]

1) Bagging

Bagging is the very first and simple way of output aggregation. It's way to reduce the variance of an estimate is to average together multiple estimates. One simple bagging example is shown in following,

$$agg(x) = \frac{1}{M}\sum_{i=1}^{M} f(x)$$

In this project, I used three classifiers to do the bagging ensemble, which are Logistic Regression Classifier, Gaussian Naïve Bayes and Support Vector Classifier. In each of the models, I tuned the parameter using GridSearchCV algorithm to reach a competitive performance in single model training before bagging ensemble them together.

a) Logistic Regression

Logistic Regression is an simple and very first model I tried and it can help us have a basic impression and expectation about the data and model. Logistic Regression prediction can be demonstrated with following two formula,

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$$

b) Gaussian Naïve Bayes

Gaussian Naïve Bayes is an extension of Naïve Bayes, with addition assumption of Gaussian distribution. It's the common and easy way to work with. The likelihood of Gaussian Naïve Bayes can be expressed as following formula,

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

c) Support Vector Classifier

Support Vector Classifier is a large margin classifier, optimizing distance between positive and negative hyperplane.

With geometric margin of (w, b) with respect to training set $S = \{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$, SVM can be demonstrated by following formula,

$$\begin{aligned} \max_{\gamma,w,b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \ldots, m \\ & ||w|| = 1. \end{aligned}$$

2) Boosting

Boosting is another way of model ensemble which can help optimized the weak part that each of the classifier learned in the last round. In another word, more weight is given to examples that were misclassified by earlier rounds.

a) Gradient Boost

Gradient boosting Classifier would first optimize its loss function, predict the weak learning and subsequently add weak learner to help optimize the loss. Detail algorithm can be demonstrated as below,

Algorithm 1: Gradient_Boost

$F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^{N} L(y_i, \rho)$
For $m = 1$ to M do:
  $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$
  $\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
  $\rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$
  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
endFor
end Algorithm

Table 2 *Gradient Boost Algorithm [11]*

b) XGBoost

The system is optimized for fast parallel tree construction, and designed to be fault tolerant under the distributed setting. XGBoost can handle tens of millions of samples on a single node, and scales beyond billions of samples with distributed computing. [12] Following is the algorithm about how XGBoost works,

Algorithm 1: Parallel Tree Split Finding Algorithm on Single Machine

Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m *in parallel* **do**
  $G_L \leftarrow 0, \; H_L \leftarrow 0$
  for j *in sorted*(I_k, *ascent order by* $\mathbf{x}_{jk}$) **do**
    $G_L \leftarrow G_L + g_j, \; H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L, \; H_R \leftarrow H - H_L$
    $gain \leftarrow \max(gain, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split and default direction with max gain

Table 3 *XGBoost algorithm [12]*

In this project, I did a two-level learning where XGBoost model was used for second-level boosting of four models. Detailed design is shown as following figure,

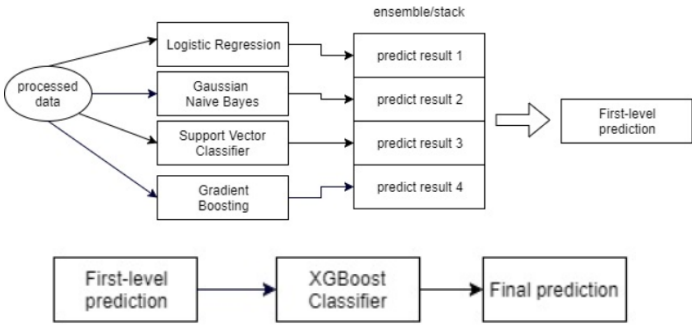

Figure 4 *XGBoost two-level learning framework*

V. Experiments & Results

With the selected features, I started to implement the two ways of dropout rate prediction described in above section. Together with total labels of ground truth dropout result, I first split the extracted features with enrollment ID into two parts, use set and test set, with portion of 4:1. In addition, I continued to separate use set into train set and validation set. When doing the model training, I fit model using the train set and used the validation set to do the cross validation to get the score (logloss, accuracy or AUC). During the training stage, in order to reach best performance of one model, I used GridSearchCV to tune parameters and got the best parameters as model parameters before stacking them together. When evaluating the model, score and confusion matrix would be used.

A. LSTM

Long short-term memory (LSTM) is a special case for Recurrent Neural Network and it's designed to avoid long-term dependency problem. It can remember information for long

period of time. This is because in this architecture a unit can transfer the information back to itself in different time steps.

Long-short Term Memory is also a variant RNN which is easier to train because it is immune to the exploding and vanishing gradient problem of traditional RNN.

In this project, given the daily-based features I extracted in the III.B (daily seven event frequency count, daily online time and total event frequency count), I first tried a LSTM networks first with all the daily activity measures. I used Keras with Tensorflow as backend. The model summary can be seen in Table,

Layer (type)	Output Shape	Param #
bidirectional_1 (Bidirection	(None, 4)	528
dense_1 (Dense)	(None, 1)	5

Total params: 533
Trainable params: 533
Non-trainable params: 0

Train on 65092 samples, validate on 7233 samples

Table 2 *LSTM model summary*

There are four variables for LSTM model, number of neuron, number of epochs, number of batch and whether LSTM is bidirectional or not. I tuned each of the parameters one by one using gridsearch. The best result is given by the model with the stacked two-layer bidirectional LSTM followed by a sigmoid unit, with batch size=200. The input size is 7(daily activity count for each event) + 7(daily online time) * 30 (days) and output is the probability of dropout.

Figure 2 is the learning curve for the training and validation stage.

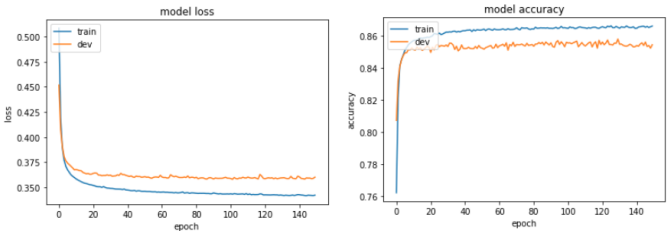

Figure 5 *Log loss and accuracy for two-layer bidirectional LSTM Model*

The evaluation accuracy is 85.8%. It is not my best model but I don't consider it a failure. I think a potential improvement is that I should use real time-series data which I extracted from the same time points for every observation instead of defining day1 to day30 according to their individual start time. This way I will end up with different lengths of sequences for each student. And I will pad them to feed in the LSTM model.

B. Model ensemble learning

1) Bagging

With GridSearchCV on three models in bagging, Logistic Regression would reach its best performance in 'C=1' and Support Vector Classifier would reach its best performance in 'C=0.001' and Linear Kernel. With mean bagging of Logistic Regression Classifier, Gaussian Naïve Bayes and Support Vector Classifier, the result can only slightly improve 0.1%.

2) Boosting

a) Gradient Boost

Beside Gradient Boost, I also tried other decision tree classifiers, like Extra Tree and Adaboost; while Gradient Boost has better performance on accuracy and learning curve, as you can see in the figure. The best parameters for my GBDT is deviance loss with 180 trees of maximal depth 5 and min samples leaf of 2. The max features are 60% of total features.

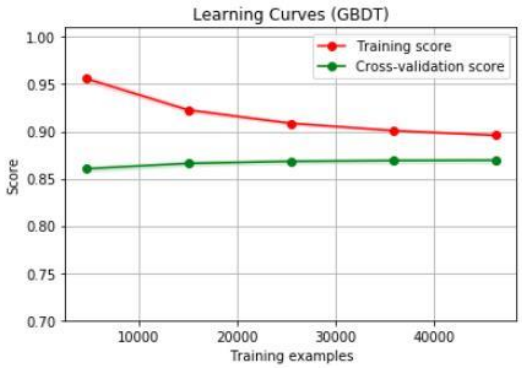

Figure 6 *Learning curve of GBDT*

Confusion matrix for GBDT in final test set is true negative number is 1,597, false positive number is 1,429, false negative number is 492 and true positive number is 10,947. Precision = TP/(TP+FP) = 0.873, Recall = TP/(TP+FN) = 0.957 and false positive rate = FP/(FP+TN) = 0.472. Normalized confusion matrix figure is shown in the following,

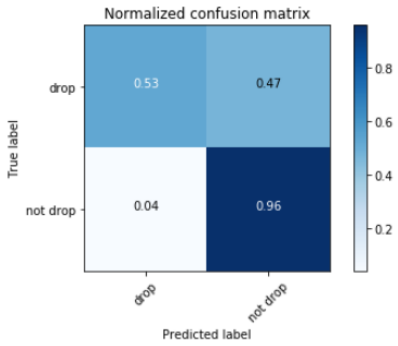

Figure 6 *Confusion matrix (GBDT)*

b) XGBoost

As mentioned in the previous part, Gradient boosting classifier is used as second level boosting ensemble to further predict the output from the earlier 4 first-level predictions from linear Logistic Regression, Support Vector Classifier, Gaussian Naïve Bayes and Gradient Boosting Decision Trees. The final evaluation accuracy is 0.876.

C. Discussion

As observed by the result, I can see that model ensemble, especially decision tree classifiers are comparably better than

one single classifier like Logistic Regression, Naïve Bayes and Support Vector Classifier. This is because decision tree can handle non-linear features and have interactions between different features, comparing to simple classifiers like Logistic Regression.

For the issue of overfitting and underfitting, I mostly observed it from the training error/score and validation error/score to see if there's any overfitting, as suggested in the classes. The Stacked LSTM model also seems over-fit the problem. It can be seen from the learning curve (Figure 5) that even though the model already converged, my evaluation accuracy is still lower than the training accuracy. Looking back, I think it is because I didn't shuffle the data before every epoch and my model is a little complex.

The first lesson I learned is that in the real problem, we always need to go back and forth with feature selection (even feature engineering) and model tuning. Even though in general some features are better than some others, I found out that different model have a slightly different favor for features. I turned out need to do feature selection for each model.

The second lesson I learned is that good features is more important than fancy models. I spent too much time trying and tuning different models only to find out eventually that some of my activity count features are bad. And when I finally realized that, I turned out don't have enough time to tune each model for my new feature space.

VI. Conclusion

In this project I worked on building models to get best of prediction about dropout probability in MOOC platform. Given the interest irregular three-dimension dataset, I tried different models, tuned the parameters and ensemble them together to reach good accuracy. In addition from that, I implemented the LSTM network to predict dropout probability based on the daily data in 30 days. In the future, I believe diving into LSTM when predicting the dropout prediction based on day-base data will be promising.

References

[1] G. Dekker, M. Pechenizkiy, J. Vleeshouwers, "Predicting students drop out: a case study", *Educational Data Mining 2009*, 2009.

[2] R. F. Kizilcec, C. Piech, E. Schneider, "Deconstructing disengagement: analyzing learner subpopulations in massive open online courses", *Proceedings of the third international conference on learning analytics and knowledge*, ACM, pp. 170-179, 2013.

[3] J. Kim, P. J. Guo, D. T. Seaton, P. Mitros, K. Z. Gajos, R. C. Miller, "Understanding in-video dropouts and interaction peaks in online lecture videos", *Proceedings of the first ACM conference on Learning@ scale conference*, ACM, pp. 31-40, 2014.

[4] S. Moon, S. Potdar, L. Martin, "Identifying student leaders from mooc discussion forums through language influence", *EMNLP 2014*, pp. 15, 2014.

[5] D. Yang, M. Wen, C. Rose, "Towards identifying the resolvability of threads in moocs", *EMNLP 2014*, pp. 21, 2014.

[6] Liang, J., Li, C., & Zheng, L. (2016, August). Machine learning application in MOOCs: Dropout prediction. In Computer Science & Education (ICCSE), 2016 11th International Conference on (pp.52-57). IEEE.

[7] KDD Winner white paper: http://www.conversionlogic.com/wp-content/uploads/2016/06/Whitepaper- KDD2015-JYL.pdf

[8] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM networks," Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., Montreal, Que., 2005, pp. 2047-2052 vol. 4. doi: 10.1109/IJCNN.2005.1556215

[9] A. Graves, A. r. Mohamed and G. Hinton, "Speech recognition with deep recurrent neural networks," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, BC, 2013, pp. 6645-6649. doi: 10.1109/ICASSP.2013.6638947

[10] D. P. Gaikwad and R. C. Thool, "Intrusion Detection System Using Bagging Ensemble Method of Machine Learning," *2015 International Conference on Computing Communication Control and Automation*, Pune, 2015, pp. 291-295. doi: 10.1109/ICCUBEA.2015.61

[11] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Annals of statistics, 38(2):337– 374, 2000.

[12] Tianqi Chen , Carlos Guestrin, XGBoost: A Scalable Tree Boosting System, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 13-17, 2016, San Francisco, California, USA