Machine Learning Final Report

Tzu-Ming Harry Hsu (B00901168), Yi Hsiao (R04945027), and Li-Yuan Liu (R04944018)

1 Abstract

In this report, we will briefly introduce the methods we used in the final project of the machine learning course, fall 2015, in which we predict whether a user drops from an online course. There will be three parts, namely ***Feature Engineering***, where we dig deep into the log files; ***Algorithms***, where we build a few single-models with three kinds of algorithms, along with rating their applicability; ***Blending***, where we utilize models built from the previous sections, aggregating into a large yet better model. Finally we conclude on the building process of the models. All project materials can be found on: `https://github.com/hsiaoyi0504/ML2015Final`.

2 Feature Engineering

In our experiments, we have tried two kinds of features: one made by TA, denoted as `featTA`, and one made by us, incorporating user information, course information, and enrollment-specific information. This feature is afterwards denoted as `feat1`. We hereby introduce `feat1`, which expands to ***1926 dimensions*** in total, in detail. The feature names are followed by its dimensions and description. Three subsets, `x1`, `x2`, and `x3` are included, where `x1` is 1-dimensional features, and `x2` and `x3` are 2-dimensional features, potentially used by convolutional methods.

- A suggested training/validation index set
  - `perm_train_train (91613,)`:
    Suggested development index set, ***95%*** of the original training data.
  - `perm_train_val (4821,)`:
    Suggested validation index set, ***5%*** of the original training data.
  - `perm_test (24108,)`: Original testing set
- `x1 (216,)`: A feature subset
  - `enrollment_id (1,)`: Enrollment id
  - `enrollment_time (1,)`:
    The time of first log after the course released
  - `enrollment_log_num{norm_erm} (3, )`:
    Number of logs in this enrollment, `norm_erm` is specified below
    - `/`: No normalization
    - `/user`: Divided by `user_log_num`
    - `/course`: Divided by `course_log_num`
  - `enrollment_log_{act}_num{norm_erm} (9 x 3,)`:
    Number of logs of the specified activity, and `act` is specified below
    - `browser_access`
    - `browser_page_close`
    - `browser_problem`
    - `browser_video`
    - `server_access`
    - `server_discussion`
    - `server_nagivate`
    - `server_problem`
    - `server_wiki`
  - `enrollment_log_{obj_cat}_num{norm_erm} (5 x 3,)`:
    Number of logs of the specified object category, and `obj_cat` is specified below
    - `chapter`
    - `combinedopenended`
    - `problem`
    - `sequential`
    - `video`
  - `sess_time_hist (30,)`:
    The logarithmic histogram of the time for the objects involved to be interacted since its release
  - `sess_duration_hist (30,)`:
    The logarithmic histogram of the duration of log sessions from this enrollment
  - `sess_interval_hist (30,)`:
    The logarithmic histogram of the interval of log sessions from this enrollment
  - `user_id (1,)`: User id
  - `user_log_num (1,)`:
    Number of logs summed over all enrollment related to this user
  - `user_log_{act}_num{norm_user} (9 x 2,)`:
    Number of logs of the specified activity summed over all enrollment related to this user, `norm_user` specified below
    - `/`: No normalization
    - `/user`: Divided by `user_log_num`

- `user_log_{obj_cat}_num{norm_user} (5 x 2,)`:
  Number of logs of the specified object category summed over all enrollment related to this user
- `user_course_num (1)`:
  Number of courses taken by this user
- `user_course_active_num{norm_user}@win (2 x 3,)`:
  Number of other active courses before, during, and after this enrollment
- `user_course_active_log_num{norm_user}@win (2 x 3,)`
- `user_course_drop_num{norm_user} (2,)`:
  Number of ***other*** courses dropped by the user
- `course_id (1,)`:
  Course id
- `course_module_num (1,)`:
  Number of modules in this course
- `course_log_num (1,)`:
  Number of logs summed over all enrollment related to this course
- `course_log_{act}_num{norm_course} (9 x 2,)`:
  Number of logs of the specified activity summed over all enrollment related to this course, `norm_course` specified below
  - `/`:
    No normalization
  - `/course`:
    Divided by `course_log_num`
- `course_log_{obj_cat}_num{norm_course} (5 x 2,)`:
  Number of logs of the specified object category summed over all enrollment related to this course
- `course_user_num (1,)`:
  Number of users taking this course
- `course_user_drop_num{norm_course} (2, )`:
  Number of users dropping this course

- `x2 (45, 30)`:
  A time-expanded feature subset, created by binning the logs into several fixed bins (here, 30) rather than summing them together
  - `enrollment_log_num{norm_erm}@time (3, 30)`
  - `enrollment_log_{act}_num{norm_erm}@time (9 x 3, 30)`
  - `enrollment_log_{obj_cat}_num{norm_erm}@time (5 x 3, 30)`
- `x3 (4, 90)`: Another time-expanded feature subset
  - `user_course_active_num{norm_user}@win@time (2, 3 x 30)`
  - `user_course_active_log_num{norm_user}@win@time (2, 3 x 30)`

To verify the effectiveness of our `feat1`, we train a simple linear L_2-regularized primal logistic regression model on the development set with default parameters, and evaluate the classification accuracy with development set.

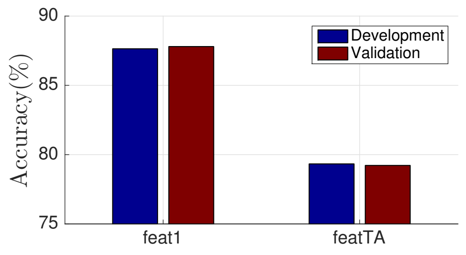

Figure 1: Comparison between features

From Figure 1 we easily see significant improvement of `feat1` over `featTA`, proving our feature is superior and can help a lot in the following tasks.

3 Algorithms

3.1 Linear Logistic Regression

In the first model, we follow the principle of ***'linear first'***. We combine the ***Principal Component Analysis*** (PCA) [1] and L_2-penalized logistic regression (LR). PCA is a well-known dimension reduction method, which seeks good representation of data with the criteria maximizing the variance. It is an unsupervised learning method. To probe our problem, which we have some labeled data, we combine PCA with a supervised learning method, that is, linear logistic regression. The reason why we choose logistic regression is due to the requirement of track 1 which somewhat implies us to generate a 'probability like' output. Another reason why we construct such model is efficiency. The feature set we generate is very high dimensional, and this will result in long training time. PCA's dimension reduction makes the number of features used in logistic regression fewer. On the other hands, PCA find maximum variance data representation can be view as feature selection.

3.1.1 Environment

We implement such a pipeline model by Python using Scikit-learn [2].

3.1.2 Settings

We use the absolute error for track 1 and 0/1 error for track 2 to train our model. We use grid search cross validation method to pick all parameters for our model such as the number of dimensions N in PCA and C in logistic regression formula below.

$$\min_{\mathbf{w},c} \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C\sum_n \log\left(\exp\left(-y_n\left(\mathbf{w}^\top \mathbf{x} + c\right)\right) + 1\right)$$

The results of the grid search is reported below.

Table 1: The optimal parameters

	`featTA`	`feat1`
Track 1	$N = 31, C = 10^{-1}$	$N = 122$, $C = 10^{2}$
Track 2	$N = 10, C = 10^{-4}$	$N = 185$, $C = 10^{4}$

3.1.3 Performance

Before uploading to the online judge system, we evaluate the model with validation data. Track 1 is based upon 0/1 error, or $E^{0/1}_{\text{val}} = \sum_n [\![\tilde{y}_n \neq y_n]\!]$; and track 2 on absolute error, or $E^{\text{abs}}_{\text{val}} = \sum_n |\tilde{y}_n - y_n|$, where $\tilde{y}_n$ is the predicted class and y_n is the ground truth. We then report the public score S^{pub} on the online judge system.

Table 2: E_{val} and S^{pub} for logistic regression

	`featTA` + LR	`feat1` + LR
Track 1	$E^{\text{abs}}_{\text{val}} = 0.309$, $S^{\text{pub}}_1 = 0.957$	$E^{\text{abs}}_{\text{val}} = 0.290$, $S^{\text{pub}}_1 = 0.960$
Track 2	$E^{0/1}_{\text{val}} = 0.155$, $S^{\text{pub}}_2 = 0.837$	$E^{0/1}_{\text{val}} = 0.152$, $S^{\text{pub}}_2 = 0.856$

Since we have yet to obtain satisfactory results, we decide to build a bootstrapped model with 100 small models of PCA and linear logistic regression. Each model is trained by using bootstrapped data with size equal to training set. The models are selected by 5 fold cross validation and a loose criteria that the $E^{0/1}_{\text{val}} < 0.17$ or $E^{\text{abs}}_{\text{val}} < 0.32$. The dimensionality of PCA is determined randomly between $\frac{D}{4}$ and $\frac{D}{2}$ where D is the original dimensionality. The generation of small models is somewhat like the behavior of random forest. The performance of the improved bootstrapped LR (BS-LR) is shown below.

Table 3: E_{val} and S^{pub} for BS-LR

	`feat1` + BS-LR
Track 1	$E^{\text{abs}}_{\text{val}} = 0.1839$, $S^{\text{pub}}_1 = 0.9666$
Track 2	$E^{0/1}_{\text{val}} = 0.1220$, $S^{\text{pub}}_2 = 0.8783$

3.2 `libsvm` [3]

Since we have tried linear models at the first sight and the results, yet not awful, do not look promising, we decide to empower the classifiers with kernels. That is, we try the performance with different kernel types in `libsvm`. However, we must keep in mind that using kernels entails potential overfitting problems, so we must keep an eye on both the development error and validation error. $E^{0/1}_{\text{val}}$ is reported below.

Table 4: E_{dev} and E_{val} for different kernels

Kernel Type	$E^{0/1}_{\text{dev}}$	$E^{0/1}_{\text{val}}$
Linear	$0.06 \sim 0.10$	$0.30 \sim 0.40$
Polynomial	0.23	$0.30 \sim 0.35$
Radial Basis Function	$0.15 \sim 0.20$	$0.15 \sim 0.25$

It is obvious that radial basis function performs better than other kernel types. So we use it and keep adjusting parameters to improve. We only adjust parameters γ and C in C-SVM, because they improve the result more. The best $E^{0/1}_{\text{val}}$ is 0.20 with parameters $\gamma = 10^{-2} \sim 10^{-1}$ and $C = 10$. For regression, we use `libsvm` epsilon-SVR to train a single model. Using the model, we get the result below:

Table 5: E_{val} and S^{pub} for `libsvm`

	`feat1` + `libsvm`
Track 1	$S^{\text{pub}}_1 = 0.9329$
Track 2	$E^{0/1}_{\text{val}} = 0.20$, $S^{\text{pub}}_2 = 0.8775$

3.3 Random Forest Regressor

Classification and regression trees [4] are known for their training efficiency, and, while the testing data exhibits similar distributions, evaluates fast with high accuracy. Having taught in class and implemented well by toolkits, we do not elaborate on how it is built here.

3.3.1 Environment

The regression trees are implemented in Scikit-learn [2] under Python.

3.3.2 Settings

We, in fact do not modify default settings for building a regression tree in Scikit-learn, because it internal uses k-fold cross-validation to avoid overfitting the data. Pruning the tree is implemented internally to remove unnecessary nodes. Since $E_{\text{val}}^{0/1}$ for a single regression tree is only around 0.15, we implement bootstrap on the regression tree, with a sample size of the original development set with replacement. 128 regression trees are aggregated to form a ***Random Forest Regressor*** (RFR).

3.3.3 Performance

We report the results of the random forest regressor here.

Table 6: E_{val} and S^{pub} for RFR

	feat1 + RFR
Track 1	$E_{\text{val}}^{\text{abs}} = 0.1926,\ S_1^{\text{pub}} = 0.9645$
Track 2	$E_{\text{val}}^{0/1} = 0.1226,\ S_2^{\text{pub}} = 0.8787$

3.4 Convolutional Neural Network

There are many reasonable excuses for dropping a course, such as

- No interest on the first sight.
- Having other courses of larger interests.
- Having other personal issues which disallows his/her access to MOOC site.
- Having obtained unsatisfactory marks and is unwilling to continue.

There are many other reasons, however, almost every single one of them need some more detailed information, or, ***time-related information***. In digital speech processing, we usually obtain 13-dimensional Mel-frequency cepstral coefficients, succeeded by first-order and second-order differentiations to obtain 39-dimensional coefficients. Similarly, if we have higher-order features, we may be able to predict with higher accuracy. To further exploit the time-relevant features such as `feat1.x2` and `feat.x3`, we resort to the recently popular ***Convolutional Neural Network*** (CNN) [5].

3.4.1 Environment

Keras built on top of Tensorflow [6] is adopted to build the CNN. For efficiency, CUDA 7.5 toolbox is used, running on a NVidia GTX 980 Ti CPU.

3.4.2 Data pre-processing

Training and testing feature vectors and feature maps are normalized together such that each dimension possesses a mean of zero a variance of unity.

3.4.3 Settings

We design the CNN considering the `GoogLeNet` [7] inception architecture. The abbreviations in the figure are:

- `FC`:
  Fully connected layer followed by a dropout layer
- `Conv (height`×`width`+`stride/padding)`:
  Convolutional layer
- `MaxPool (height`×`width`+`stride/padding)`:
  Max pooling layer
- `Merge`: Merge inputs along depth

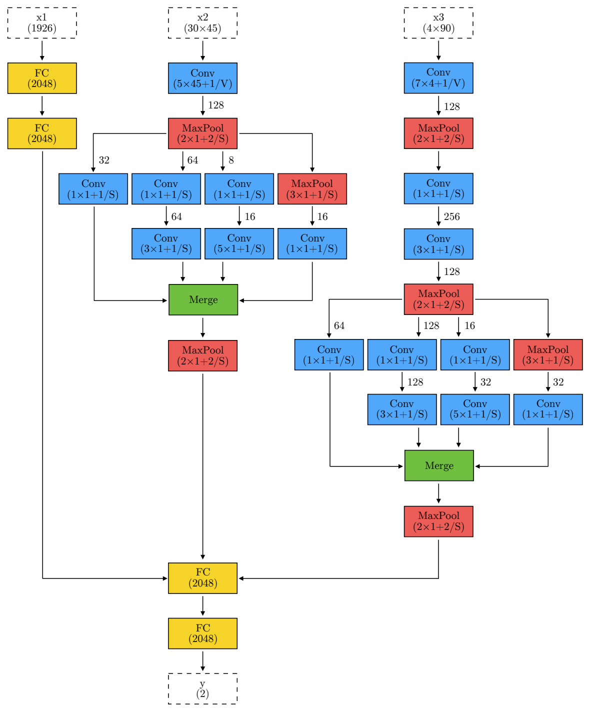

Figure 2: CNN Architecture

After the neural network is built, we randomly initialize its weights, and then train with ADAM [8] optimizer under learning rate 10^{-5} over the development set.

3.4.4 Performance

To determine how many epochs should be trained before stopping, we use the validation set error on the output. As we can see, the development loss drops steadily while the validation loss is already saturated after $3 \sim 5$ epochs. Due to the limitation of the Keras toolbox, we are unable to evaluate the loss to a finer level, and by observing its verbose output, we know training for 5 epochs is enough.

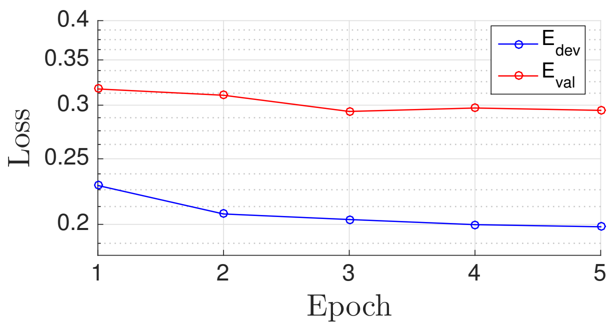

Figure 3: CNN training losses

Similarly, we evaluate $E_{\text{val}}^{0/1}$ and $E_{\text{val}}^{\text{abs}}$ for a single CNN model and report its public score S^{pub}.

Table 7: E_{val} and S^{pub} for CNN

	`feat1` + CNN
Track 1	$E_{\text{val}}^{\text{abs}} = 0.1904$, $S_1^{\text{pub}} = 0.9686$
Track 2	$E_{\text{val}}^{0/1} = 0.1120$, $S_2^{\text{pub}} = 0.8847$

3.5 Comparison

SVM does not perform well under these settings. Comparing these models, combination of PCA and logistic regression is a linear model and the output from logistic regression gives some physical meaning of probability. The linear model gives the interpretability but cannot calibrate non-linear behavior, making its performance limited.

Table 8: A comparison between methods

	BS-LR	RFR	CNN
Efficiency	★★★★★	★★★☆☆	★☆☆☆☆
Scalability	★★☆☆☆	★★★★☆	★★★★★
Popularity	★★★☆☆	★★★☆☆	★★★★★
Interpretability	★★★☆☆	★★★★☆	★☆☆☆☆
Performance	★★☆☆☆	★★★☆☆	★★★★★
Track 1 S_1^{pub}	0.9666	0.9645	0.9686
Track 2 S_2^{pub}	0.8783	0.8787	0.8847

4 Model Blending

Over the mere several days we, in total have trained 73 bootstrapped logistic regression (BS-LR) models, 29 random forest regressors (RFR), and 81 convolutional neural networks (CNN).

In the beginning, we try to use uniform blending to aggregate models using the average predictions of the models as the blending result. Then we try to use linear blending. Using validation accuracy as the weight, we calculate the mean of all models. If the mean is more than 0.5, then the prediction is 1 otherwise 0. However, the uniform blending and linear blending do not improve the result much, because the predictions of every model are too similar. Then the final blending result is almost equal to the best result.

4.1 Settings

Finally we resort to ***neural network blending***, that is, we train a small neural network on top of the prediction results from 183 models. The model built is a mere $183-128-2$ network with `tanh` activation units, with a learning rate of 10^{-3}, allowing soft and gentle training. In fact, we use half the validation data for blending model training, and half the validation data for blending model validation. Instead of using stochastic gradient descent, we use the whole training validation data for gradient calculation.

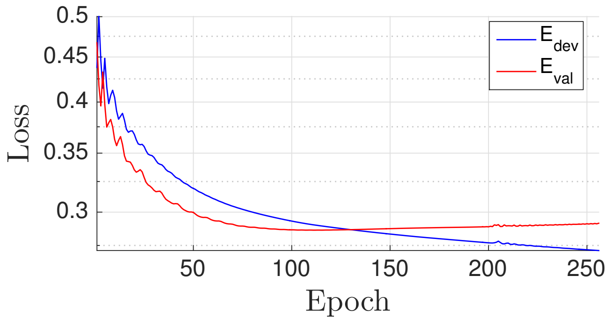

Figure 4: Blending training losses

4.2 Performance

To determine how many epochs should be trained before stopping, we use the validation set error on the output. As we can see, the development loss drops steadily while the validation loss is already rising at 100 epochs. We decide to train the blending model with all the validation data for 64 epochs at last, which is the half of 128, since we only had half the data at blending model validation. In the end we are able to achieve a better result.

Table 9: E_{val} and S^{pub} for Blending model

	`feat1` + Blend
Track 1	$E_{\text{val}}^{\text{abs}} = 0.1712$, $S_1^{\text{pub}} = 0.9703$
Track 2	$E_{\text{val}}^{0/1} = 0.1084$, $S_2^{\text{pub}} = 0.8879$

5 Conclusion

Finally we are able to achieve a score of $S_1^{\text{pub}} = 0.9703$ and $S_2^{\text{pub}} = 0.8879$ despite the fact that we did not specifically optimize for neither tracks according to their measures. What we can do is we pair the samples and, if they have different ground truths, we maximize their margins, and in this way the results for track 1 can be optimized. On the other hand, we can reweight the samples in track 2 according to the inverse of courses taken to emphasize on people with lesses courses. While regression methods are generally better for track 1 and classification methods are for track 2, we only took regression methods for both tracks, and that may degrade our result a little. At the end, for the neural network blending, we can train a large ensemble of small neural networks and make them collaborate to give a better final result.

6 Division of Work

- Tzu-Ming Harry Hsu (B00901168): Feature engineering, RFR, CNN, NN blending, and report compilation
- Yi Hsiao (R04945027): Feature engineering, and PCA/LR
- Li-Yuan Liu (R04944018): Feature engineering, `libsvm`, and linear blending

References

[1] I. Jolliffe, *Principal component analysis.* Wiley Online Library, 2002.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[4] R. J. Lewis, "An introduction to classification and regression tree (cart) analysis," in *Annual Meeting of the Society for Academic Emergency Medicine in San Francisco, California*, pp. 1–14, 2000.

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems, 2015," *Software available from tensorflow. org.*

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *arXiv preprint arXiv:1409.4842*, 2014.

[8] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.