Churn Predictive Model in Telecom Industry

Qiuyi Lu,
Rady school of management,
UCSD, CA, United States
q2lu@ucsd.edu

Zifan Qian
UCSD
CA, United States
z7qian@eng.ucsd.edu

ABSTRACT

In the involving world, one thing doesn't change is that all companies need think about how to retain old customers in addition to attracting new customers. No matter which industry the company belongs to, it's necessary for it to understand the reason why it is losing customers. Thus, we choose the churn analysis, which has a wide application in the business world, as our project.

Currently, marketing research and practice aims at maximizing the correct classification of churners and non-churners. [1] And there are platforms such as Firefly.ai and tray.io to help company do churn management.

In the project, our goal is to build a model that can help companies more precisely predict if the customer is leaving and then adopt more effective incentives to retain the customer. If they churn, how many values will we lose. We used data-driven classification models to make predictions. Our result showed that our models are capable of detecting potentially churning customers. Companies can also combine our model with other CRM tools to retain those customers who bring greatest values to them.

KEYWORDS: Churn Prediction, Customer Analytics, Logistic Regression, Boosting, Deep Learning

1. Data Exploratory Analysis

The dataset we used is 51047 rows* 58 columns, which is a Cell to Cell dataset[2] in telecom industry from Kaggle. The label is the column "churn" with "yes" representing that the customer left and vice versa. The columns include demographic information and customer behaviors of making phone calls.

Before we analyze the data, we simply cleaned it by dropping over 1000 records contain NAs. There is also a column called "HandSetPrice" containing 20,000 missing value. There's no explanation of the column, but, by checking the other columns, we thought it should be the price to buy the phone when signing a contract. The price would be highly related with "MonthlyRevenue", "Handsets" and "HandsetModels". These 3 columns contain no missing value, so we think we can discard the column when building our model rather than fill 20,000 nulls with mean value, which will instead distort our model.

1.1 Demographic insights

Do people who have a higher or lower credit rating tend to be churners? The answer is NO. Besides, there are no difference in their age, marriage status and income.

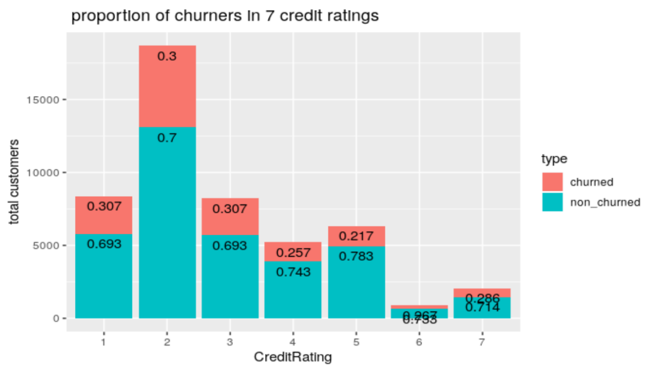

Figure 1: proportion of churners in 7 credit ratings

Are churners concentrated in some areas?
We filtered service areas with higher than the average total customer number and higher than the average proportion of churners. And ranked top10 problematic areas:

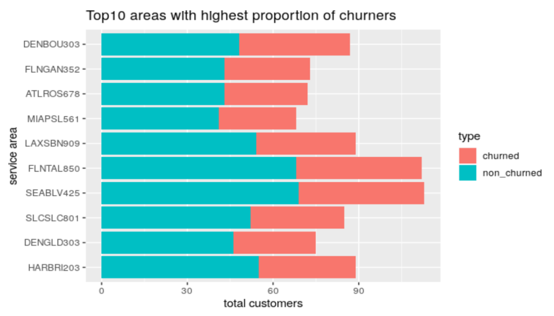

Figure 2: Top10 areas with highest proportion of churners

The company should dive deep into these areas to check if the signal in that area is bad or something else.

1.2 Customer behaviors

We raised some questions such as “Are churners more sensitive by price? Do they make more phones? Are they annoyed by cold calls or customer-care phones? Do they receive more phones than calling? Do they use more functions of the phone call? Whether the longer the person use the phone, the less likely he will leave?”
But we didn’t find significant differences between churners and non-churners in the following features: the average monthly revenue, monthly minutes, blocked calls, CustomerCareCalls, ReceivedCalls, OutboundCalls, InboundCalls, UnansweredCalls, Three-way calls, MonthsInService, NewCellphoneUser.
It seemed that they are not quite different in the behaviors of using phones.
However, the feature “**PercChangeMinutes**” showed large difference. Even if we use median, churners still showed greater percentage decrease in minutes than non-churners.
But we don’t know if the feature is a leading or a posterior factor. If it is a posterior one, we suggest the company to track the percentage change every week, which might indicate the customer is leaving. In our model, we use the feature to do prediction.

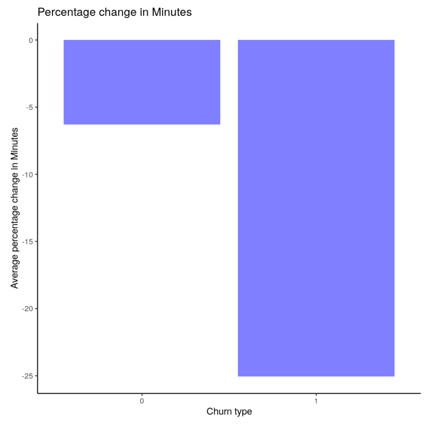

Figure 3: Average Percentage change in minutes between churners and non-churners

1.3 Product Experience:

Is retention offer help to reduce churn rate? Unfortunately, among the people who the company gave retention calls, the churn rate is higher than the group that didn’t receive retention calls. The interesting thing is that in the group that people accepted the retention offer, 40.7% left, while in the group that people didn’t accept the retention offer, only 28.4% left.
But we cannot say the retention call is not effective or the calls cause their leaving. Perhaps the company detect the group of people would leave so they made calls to them. If they don’t do so, the rate might be higher.

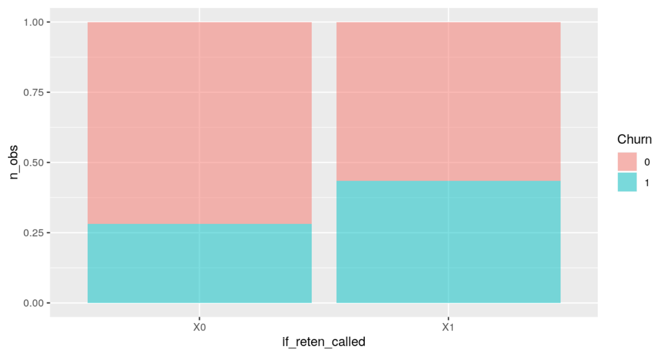

Figure 4: churned rate in groups with and without retention offers

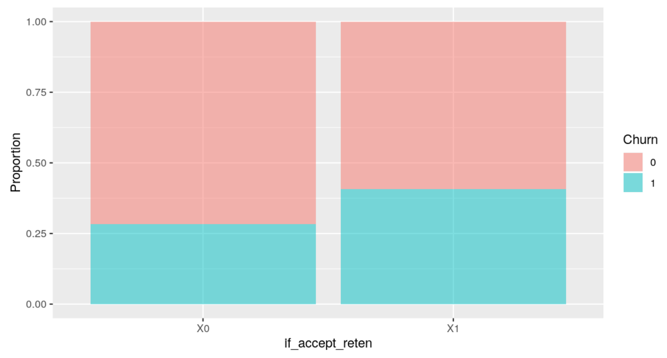

Figure 5: churned rate in groups with and without accepting retention offers

2. Related Works and Literature

The dataset we used comes from Kaggle, provided by Teradata and Duke University. People used the data to build predictive models, found important features that determine the churn possibility and cluster customers based on the risk of losing them.

Other similar datasets include churn data in music subscription service industry provided by KKBOX, churn data in bank and employee turnover in companies. Usually, people apply regression, boosting and deep learning model on the data to predict which customer will leave and build a lifecycle model for the company to retain customers. The state-of-the-art algorithm is to build a binary classifier using stochastic gradient boosting[1]. It not just predicted whether the customer will leave, but put to maximize company’s profit on the first place. It constructed a customer heterogeneous profit-based loss function that incorporates the gain to retain the customer and the cost of incentives to the targeted customer. Then it used the stochastic gradient boosting optimization algorithm to minimize the losses and finally decide the number of customers to target in the retention program.

Another state-of-the art method is to use convolutional neural networks to extract temporal user behavior[4]. Given a user's N dimensional behavior in a time period, for example 30 days, we can reshape it into a 30 by N array. Then, convolution kernels are applied to local areas to extract temporal information. The author also proposes that with trained convolutional layers the general factors of churn can be found out by the kernel weights. However, this architecture requires customer behavior in a time period which is not available in our dataset.

The conclusions are predictions that underestimate the churn propensity of high-value and highly-responsive future churners should be favored less than predictions that underestimate the churn propensity of low-value and non-responsive future churners. Likewise, predictions that overestimate the churn propensity of non-churners, whatever their value to the firm, should also be avoided, especially when the action cost is large and the response rate of these non-churners is high.

The conclusion is different from ours because we think the recall rate is important. For the telecom company, "Redeem your free 100MB data pack" almost cost nothing. If he will stay even if we don't give him the benefit, we will lose the profit of 100MB data. But if we lose the customer, we will lose more profit. So, in our model, we emphasize the recall rate. But the profit-based model gave us inspirations that we can make some assumptions such as the retention cost and each customer's respond possibility to retention offers to help our model decide whether we should retain this customer.

3. Predictive Tasks and Metrics

The main purpose of studying this dataset is to learn some models that can predict if a customer is going to churn, given the customer's recent behavior. By treating customer behavior as features and the "churn" column as label, we can model it as a binary classification problem.

3.1 Evaluation and Metrics

For classification tasks, we usually use accuracy as the major performance measure. However, accuracy is not a good choice in this case because (1) the dataset is imbalanced thus accuracy is biased by the major class, and (2) we care more about the minor class (churn) than the major class (no churn). The second reason is under the assumption that failing to detect a churning customer leads to more business loss than falsely classifying a non-churner as churner. Due to this reason, we would like our system to detect as many churning customers as possible, namely, maximizing the recall. We could directly optimize recall by predicting all customers as churners but it will make our model useless by producing too many false positives. Therefore, it is ideal to optimize both the recall and the false positive rate. For this reason, we are going to use the Receiver Operating Characteristic curve (ROC) to determine model performance. ROC is true positive rate (recall) as a function of false positive rate. If our classifiers output some scores, for example probability for logistic regression and margin for SVM, we can use multiple thresholds to get multiple classification results. ROC is obtained by calculating recalls and false positive rates of all possible thresholds. To optimize both recall and false positive rate, we aim to maximize the area under the ROC curve, defined as ROC-AUC. Similarly, we also could lower the number of false positives by increasing precision. When we plot precision as a function of recall for all the thresholds, we get the Precision-Recall (PR) curve. The area under PR curve, defined as PR-AUC, can be used as another performance measure.

For evaluation, we randomly choose 20% of total samples for testing ROC-AUC and PR-AUC, and the rest for training. For hyperparameter tuning, we use cross validation to optimize ROC-AUC. Cross validation is to divide the training set into N chunks, then use each chunk as the validation set and train a model on the rest of the chunks, and finally use the average on all N chunks as the validation result. In our case we use 5-fold cross validation (N = 5).

3.2 Proposed models

In this assignment, we would like to train and experiment with three models: logistic regression, gradient boosted trees, and neural networks.

Logistic regression is one of the most common models for binary classification problems. It learns the weights of a linear combination of features, and outputs the sigmoid of the linear combination as the probability of the positive class. It is trained by gradient descent. We use logistic regression as our baseline model.

Gradient boosted trees are decision trees trained by the gradient boosting algorithm. A decision tree is a tree-like decision structure whose internal nodes are threshold splits on features and leaves are class labels. Decision trees are non-linear models thus they can fit very complex feature patterns. With training algorithms such as bagging and boosting, decision trees can achieve high accuracy and low overfitting.

Neural networks have been extremely popular in recent years. Common feed-forward networks consist of sequential linear transformations followed by nonlinear activation functions. With a large number of parameters, neural networks have very strong fitting performance. Deep neural networks even have the capability of feature extraction.

3.3 Data and Feature Preparation

Our dataset has 58 columns in total. The column "Churn" represents the class labels. After removing the "HandsetPrice" column which contains too many (over 50%) missing values, and the “CustomerID” column which is irrelevant, we have 55 features left. Among these features, we have 20 categorical features and the rest are numerical. After applying one-hot encoding to all the categorical features, we obtain 805-dimensional feature vectors. Then, we standardize all dimensions such that each of them has zero mean and unit variance. However, with such a large number of dimensions it is likely to encounter overfitting, thus we used PCA for dimensionality reduction. By keeping 70% of the variance explained we lower the dimensionality down to 483.

4. Models and Experiments

4.1 Logistic Regression

Generic logistic regression is a binary classifier. It has a weight vector W, whose length is equal to the feature length, and a bias term b. The output is $\mathrm{sigmoid}(W^T X + b)$ where $\mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}}$, interpreted as the probability of the positive class given the features. Logistic regression is a basic model for binary classification thus we use it as the baseline. This model is trained by gradient descent that minimizes the cross entropy loss. It is also feasible to add a regularization term to the loss. We use grid search cross validation to optimize the regularization hyperparameter and get ROC-AUC = 0.6 and PR-AUC = 0.36 with L2 regularization and C = 1. “class_weight” is set to “balanced” for better performance on the imbalanced dataset. The performance is not too good but we did not encounter overfitting. Since logistic regression is a linear model, it lacks fitting performance in complex feature space but can usually generalize well.

4.2 Gradient Boosted trees

The decision tree model is one of the most practical supervised learning methods. It has a tree-like structure and each input enters at the root. At each node, a split is made based on one of the features, and the branches ultimately lead you to one of the leaves which are predicted class labels.

Decision trees as a nonlinear model can fit very complex feature distributions by increasing the depth, but it also leads to overfitting, thus it is considered a low-bias high-variance model. To ease the effect of overfitting, bagging (bootstrap aggregating) [3] techniques are usually applied to decision trees. Bagging is to uniformly sample the data, with replacement, to make a subset that has equal number of samples as the training set, and then train a model on the subset. Repeating this procedure N times we get N models. For prediction, each sample is fed into all N models, and the label is decided by the majority. A famous example of bagging is random forest[6], which is an ensemble learner of multiple decision trees trained by bagging. In addition, random forest also randomly samples a subset of features during training. By doing so, different trees tend to learn different features hence higher diversity in the system.

Another important idea of ensemble learning is boosting[5], which is the key of gradient boosted trees. Boosting is an iterative process that trains one model and evaluates model performance at every iteration. The result of boosting is given by a linear combination of all trained models. Therefore, at each iteration, boosting adds a model (along with its weight) based on all existing trained models and monotonically increases performance. Another key idea of boosting is sample weights. At every iteration sample weights are adjusted such that misclassified samples get higher weights. By doing so, “hard” samples can be noticed by future iterations and optimized better. Gradient boosting is a state-of-the-art learning algorithm for boosted decision trees. It performs boosting but learns the residual of the sum of all existing trained models. We use XGBoost[7], which is a framework that implement optimized gradient boosting, to train our model.

We follow the grid search cross validation pipeline to train our model. First, we guess some major hyperparameters and use them to run cross validation with early stopping. Then, we set the number of iterations before stopping as the number of boosted trees “n_estimators”. After that, we perform grid search on following hyperparameters:

(1) “max_depth” as the maximum depth of trees, optimized as 4
(2) “min_child_weight” as minimum sun of instance weight needed in a child, optimized as 4
(3) “gamma” as minimum loss reduction needed for further node partition, optimized as 0
(4) “subsample” as the ratio of training instances for bagging, optimized as 0.77
(5) “colsample_bytree” as the ratio of features for bagging, optimized as 0.92
(6) “reg_alpha” as L1 regularization, optimized as 1e-5
(7) “reg_lambda” as L2 regularization, optimized as 1
(8) Learning rate optimized as 0.1

After confirming all these hyperparameters, we run cross validation again and confirm the number of boosted trees “n_estimators”. Sample weights of positive samples are set to the ratio of number of negative/positive samples for better performance on the imbalanced dataset. With this model we get ROC-AUC = 0.68 and PR-AUC = 0.46 which are significantly better than logistic regression. With early stopping we limited the number of boosted trees as well as maximum depth of trees, so we did not encounter overfitting in this case.

4.3 Neural Networks

Neural networks are very powerful parametric models for fitting. According to the universal approximation theorem, a neural network with one hidden layer that contains a sufficient number of neurons can approximate all continuous functions. However, since generic neural networks have linear layers, they have a huge number of trainable parameters as matrices. As a result, neural networks are more prone to overfitting compared to other models if the training set is not big enough. In addition, the large number of parameters also significantly increases the solution space and makes the space more non-convex, hence difficult optimization.

One model of neural networks we experiment with is multi-layer perceptron. Perceptron is a classification model that uses a hyperplane to separate two classes in feature space, but it does not work if samples are not linearly separable. Multi-layer perceptron is sequential linear transformations followed by nonlinear activation functions. The activation functions add nonlinearity into the model so that nonlinear decision boundaries are possible. For our MLP model, we use the architecture: input → hidden layer (64 neurons) → ReLU → output layer(1 neuron) → sigmoid → output. Class weights are added to the loss function for balancing data. With learning rate = 1e-4 and weight decay (L2 regularization) = 2e-2 we train the network with adam optimizer for 500 epochs. We get ROC-AUC = 0.6 and PR-AUC = 0.37, which is little increase compared to logistic regression. Once we try to increase the hidden dimension to 128, the model begins to overfit as the validation loss starts to increase after 300 epochs.

Another model we use is a deeper neural network with feature embedding. One thing we noticed during data cleaning is that one categorical feature, named "ServiceArea", has more than 700 categories, while other categorical features have less than 5. This single feature makes the dimensionality extremely large after encoding and could potentially have caused overfitting. In addition, these services areas might have connections while one-hot encoding assumes independence among categories. Embedding layer is a trainable transformation that takes the index of a category and returns a vector of a given dimension, called the embedding. After training, similar categories will have embeddings closer to each other. One famous application of this is Word2vec[8], the embedding of vocabularies. We would like to see if deep neural networks can extract implicit features automatically with embedding and more layers. In this case, we assign ordinal indices for "ServiceArea" and one-hot encoding for other categorical features. With embedding size 7, the dimensionality is reduced to 90. We use the architecture: input → embedding → hidden layer (1024 neurons) → ReLU → hidden layer (512 neurons) → ReLU → hidden layer (128 neurons) → ReLU → hidden layer(32 neuron) → ReLU → hidden layer (4 neurons) → ReLU → output layer(1 neuron) → sigmoid → output. We also add Dropout with probability 0.2 after each activation to avoid overfitting. With learning rate = 1e-4 and weight decay (L2 regularization) = 1e-4 we train the network with adam optimizer for 200 epochs. We get ROC-AUC = 0.62 and PR-AUC = 0.38, which is better than MLP but not as good as XGBoost. It is noticeable that this model also encounters overfitting but the testing performance is not too poor. In this case, overfitting can be further reduced by careful tuning.

5. Results and Discussion

5.1 Logistic Regression

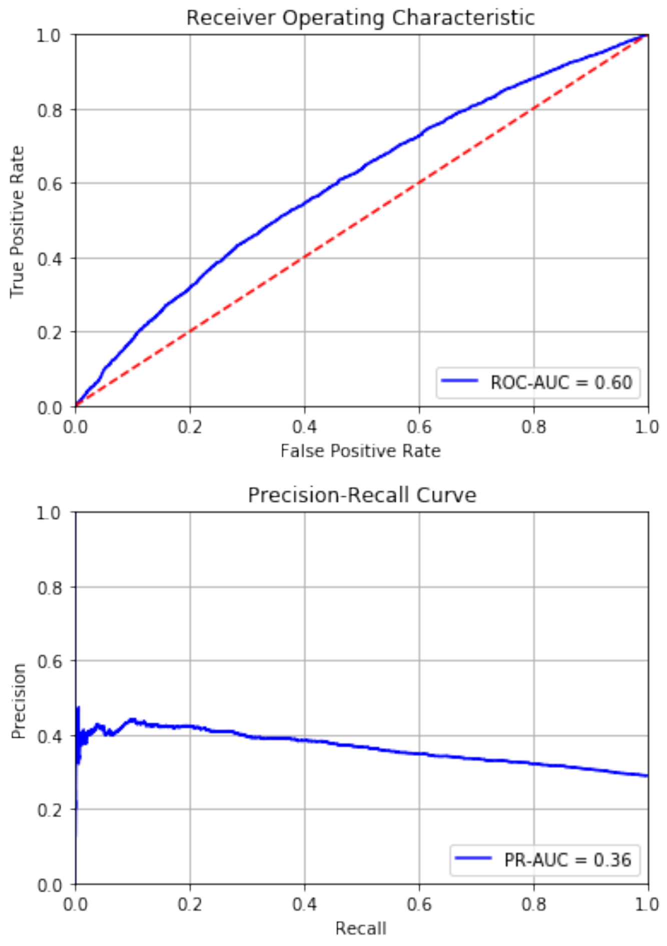

Figure 6: Logistic Regression results

5.2 Gradient boosted trees

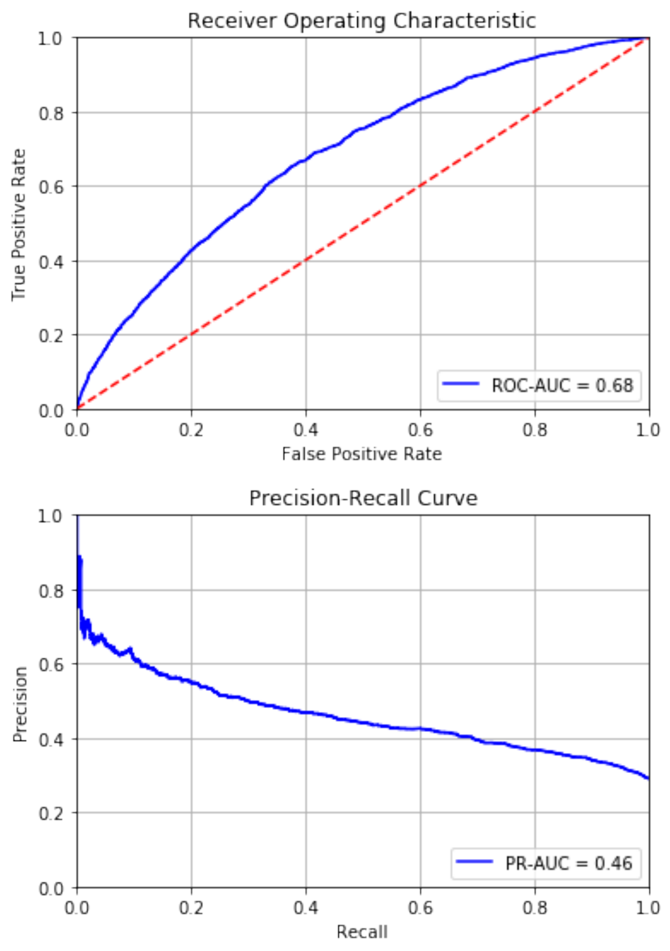

Figure 7: XGBoost results

5.3 Neural Networks

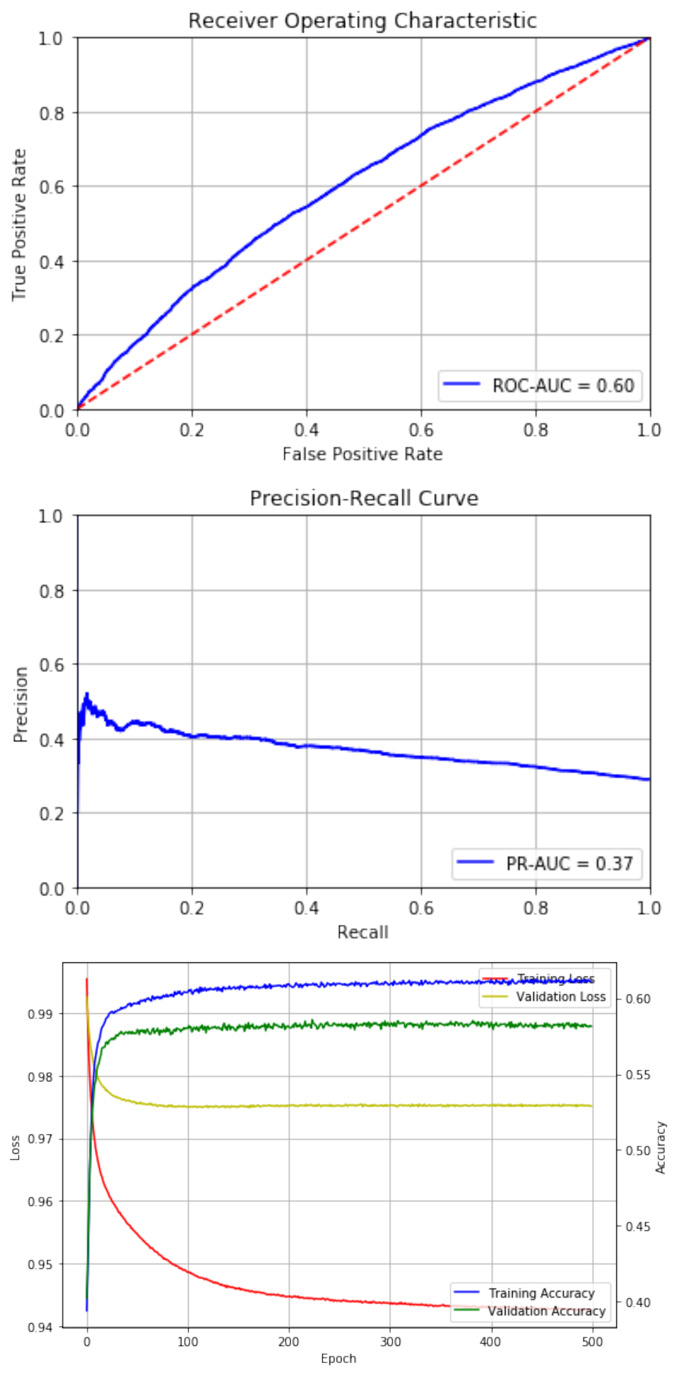

Figure 8: MLP results

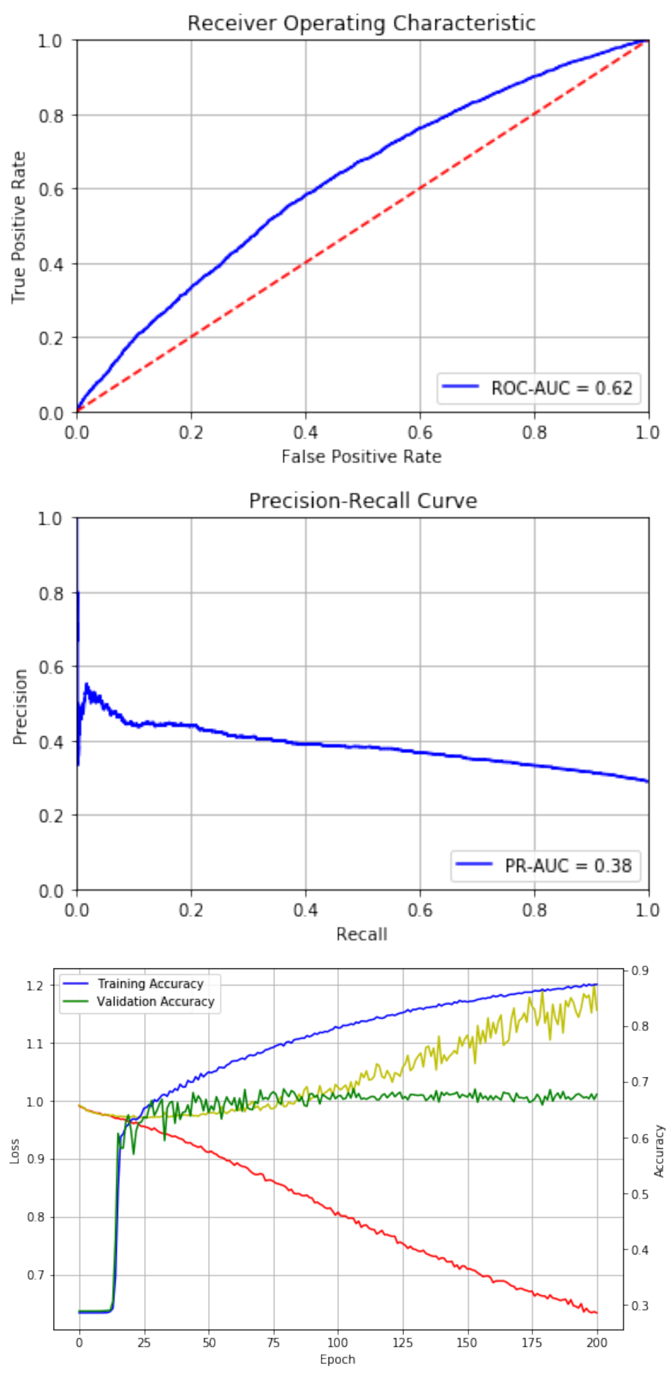

Figure 9: Deep+Embedding results

5.4 Discussion

Among all the models experimented with, Gradient boosted trees model achieves the best result, followed by deep neural network with embedding. Multi-layer perceptron performs as well as the baseline logistic regression.

With our best model, we can detect 2/3 of the churning customers if a 0.4 false positive rate is tolerable. The best threshold can be determined only if we can compare the loss of losing a customer and the loss of detecting a retained customer as a churner. Even if we cannot, it is still reasonable to assume the former is greater than the latter by common sense. Therefore, our model is capable of tracking customer behavior and maximizing profits in terms of customer retention.

It also signifies that Gradient boosted trees model, as one of the most widely used methods in industry, is very robust and applicable in practice. Boosting allows the model to choose simple weak learners (individual trees) while achieving complex and successful classification through ensemble learning. Simple decision trees, in this case with a maximum depth of 5, can generalize very well. Bagging further reduces overfitting by randomly sampling samples and features, and helps feature selection.

Compared to Gradient boosted trees, neural networks have better potential but are more prone to overfitting, especially in this case we only have a relatively small dataset of 50000. By observing the learning curve of deep network with embedding we can see that it is aiming for 90%+ training accuracy but it begins to overfit after 100 epochs. It indicates that the embedding layer and the deep network are capable of learning feature extraction thus high fitting performance. With stronger regularization, Dropout, pruning of neurons, the model is expected to achieve better performance. However, it will not be significantly better since the number of parameters of the deep network is much larger than our dataset, thus overfitting does not seem to be avoidable in this case.

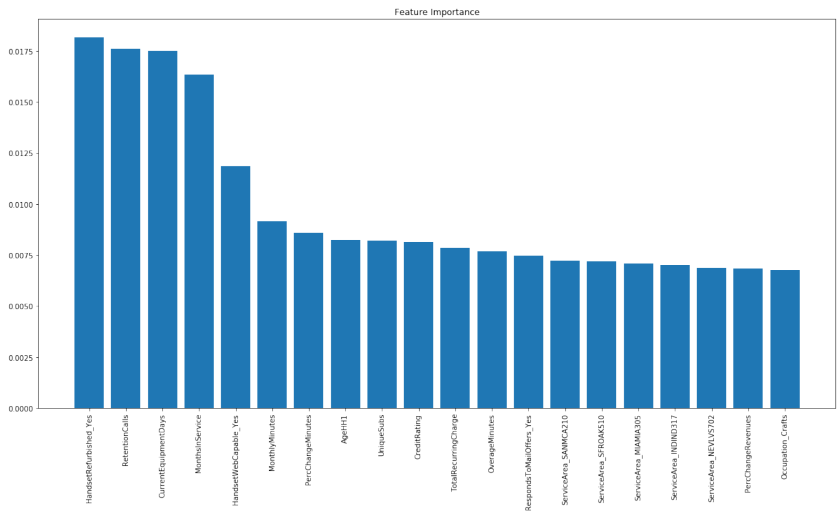

Figure 10: Most important features selected by XGBoost

By observing the important features selected by XGBoost we can see that the model has selected most relevant features to construct decision trees. For example, “RetensionCalls” is directly related to user’s intention to churn. “MonthInService” indicates that old customers are less likely to churn. “HandsetRefurbished” is related to the reliability of devices. Therefore, parameters of our model make sense.

We actually have three feature representations: features selected by XGBoost’s own boosting and bagging scheme, features transformed by embedding and multiple linear layers, and PCA features directly used by logistic regression and MLP. In terms of our metrics, the first two are better than directly using PCA features. Both XGBoost and deep learning have feature extractors while logistic regression and MLP are pure classifiers. It signifies that nowadays the bottleneck of machine learning is feature engineering instead of fitting performance. Manually constructing and creating features is very difficult since it requires a lot of domain knowledge. With models that include feature extractors it is easier to achieve better performance by simply optimizing the models themselves. It is the reason why deep learning is so popular and effective now.

References

[1] Managing Churn to Maximize Profits(2013) Aurélie Lemmens, Sunil Gupta Harvard Business School Working Paper 14-020

[2] Kaggle.com. (2019). *telecom churn (cell2cell)*. [online] Available at: https://www.kaggle.com/jpacse/datasets-for-churn-telecom [Accessed 4 Dec. 2019].

[3] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2), pp.123-140.

[4] Arxiv.org. (2019). [online] Available at: https://arxiv.org/pdf/1604.05377.pdf [Accessed 4 Dec. 2019].

[5] Yoav Freund and Robert E. Schapire (1997); *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*, Journal of Computer and System Sciences, 55(1):119-139

[6] Ho, Tin Kam (1995). *Random Decision Forests* (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. Archived from the original (PDF) on 17 April 2016. Retrieved 5 June2016.

[7] Xgboost.readthedocs.io. (2019). *XGBoost Documentation — xgboost 1.0.0-SNAPSHOT documentation*. [online] Available at: https://xgboost.readthedocs.io/en/latest/index.html [Accessed 4 Dec. 2019].

[8] "Computing numeric representations of words in a high-dimensional space"