MACHINE LEARNING
CS-6375.004
FALL 2017

FINAL PROJECT REPORT

FOR

WSDM – KKBOX'S CHURN PREDICTION CHALLENGE

TEAM MEMBERS

MUHAMMAD ARSALAN MALIK – mxm162431

MD SHIHABUL ISLAM – mxi170330

Contents

1. INTRODUCTION

We took part in one of the competitions that is being hosted at Kaggle as our term project for the Machine learning course, Fall 2017. The competition is named as "WSDM – KKBox's Churn Prediction Challenge". The competition requires us to come up with algorithm to successfully predict whether a subscription user will churn or not. Accurately predicting the churn probability of a user is vital to long-term success. Even minor variations in churn can drastically affect profits.

2. PROBLEM DESCRIPTION

The objective of this project is to predict whether a subscription user of KKBox will churn or not. KKbox is Asia's leading music streaming service with over 30 million songs in its library. They offer a generous, unlimited version of their service to millions of people, supported by advertising and paid subscriptions.

The churn is defined as, when a subscriber to a service discontinues their subscription to that service within a given period. The challenge in this competition is to predict whether a user will churn after their subscription expires. The data for this project has been provided by KKBox which contains details of the user logs, memberships and transactions.

3. DATASET DESCRIPTION

The datasets provided for this challenge are different than usual. There are multiple datasets which needed to be pre-processed and merged together to get the training dataset. Following datasets are given,

- **train**

Number of features = 2
Number of instances = 970,960

S.No	Name	Description
1	msno	user id
2	is_churn	Churn is defined as whether the user did not continue the subscription within 30 days of expiration.

- **sample_submission_zero (test dataset)**

  Number of features = 2
  Number of instances = 970,960

S.No	Name	Description
1	msno	user id
2	is_churn	Churn is defined as whether the user did not continue the subscription within 30 days of expiration.

- **transactions**

  Number of features = 9
  Number of Instances = 1,431,009

S.No	Name	Description
1	msno	User Id
2	payment_method_id	Payment Method
3	payment_plan_days	Length of membership plan in days
4	plan_list_price	In New Taiwan Dollar (NTD)
5	actual_amount_paid	In New Taiwan Dollar (NTD)
6	is_auto_renew	Auto Renewal setting on or off
7	transaction_date	Format %Y%m%d
8	membership_expire_date	Format %Y%m%d
9	is_cancel	Whether or not the user cancelledthe membership in this transaction

- **members**

  Number of features = 6
  Number of Instances = 795,090

S.No	Name	Description
1	Msno	User Id
2	City	City of residence
3	Bd	Age
4	Gender	Gender
5	registered_via	Registration Method
6	registration_init_time	format %Y%m%d

- **user_logs**

  Number of features = 9
  Number of Instances = 18,396,362

S.No	Name	Description
1	msno	User Id
2	date	Format %Y%m%d
3	num_25	# of songs played less than 25% of the song length
4	num_50	# of songs played between 25% to 50% of the song length
5	num_75	# of songs played between 50% to 75% of the song length
6	num_985	# of songs played between 75% to 98.5% of the song length
7	num_100	# of songs played over 98.5% of the song length
8	num_unq	# of unique songs played
9	total_secs	Total seconds played

3.1. CREATING TRAINING DATASET

The preprocessing of datasets performed to get the training and test datasets is discussed in detail in the next section. It also discusses feature engineering that was performed, i.e., creating new features from existing features and then removing those features.

To create the training the dataset from the given datasets, the members, train and user_logs dataset are merged with the train dataset on the common attribute 'msno'.

All the features in the above datasets are merged together on the 'msno' feature, giving 20 features in the merged dataset. The column 'is_churn' in the train dataset is the **class label**. Hence, 'is_churn' and 'msno' columns were removed to get the final training dataset. The resulting training dataset has,

Number of features = 18
Number of Instances = 4,784,765

3.2. CREATING TESTING DATASET

The test dataset that is provided by the organizers of the competition is different than normal. There are only 2 columns in the given dataset and the target label of the test set just have negative samples ('is_churn' = 0), i.e., no positive samples, which also effects the results (precision and recall percentages).

To create a proper test dataset from the provided 'sample_submission_zero', it was merged with members, transactions and user_logs datasets, similar to train dataset to get additional details about the users in the test dataset. The final test dataset has,

Number of features = 18
Number of Instances = 4,608,705

4. PRE-PROCESSING

4.1. FEATURE ENGINEERING

Feature engineering is an important part of the pre-processing of datasets. We did create new features from the existing features for different dataset. Details are as follows,

User_log Dataset,

In the user_log dataset, there was a need for a new feature that we called 'total_days_listened' which recorded the number of days a user listened to songs.

With this new feature included, the 'date' column was rendered useless and therefore deleted.

Also, there were a number of outliers in the 'total_secs' column, which were removed.

```
0 outliers in column date
0 outliers in column num_25
0 outliers in column num_50
0 outliers in column num_75
0 outliers in column num_985
0 outliers in column num_100
0 outliers in column num_unq
1122 outliers in column total_secs
```

Figure a: Number of outliers in each column (negative values)

Transaction_dataset,

In the transaction dataset, there were two columns namely, 'transaction_date' and 'expire_date' which helped in creating a new feature which we called 'membership_duration_days'. It gave a clear idea of the number of days the user was a member for a particular transaction. The two date columns were later removed from the dataset.

Members_dataset,

The 'registration_init_time' did not help in predicting churn for particular month or year. Therefore, we extracted two new features from it, 'registration_init__time_month' and 'registration_init_time_year'. This helped us to see during which month or which year users registered the most. The original column was then deleted.

4.2. STANDARDIZING NUMERICAL VALUES

The individual dataset were standardized using the scikit-learn pre-processing package's StandardScaler() function, with the mean being 0 and standard deviation equal to 1.

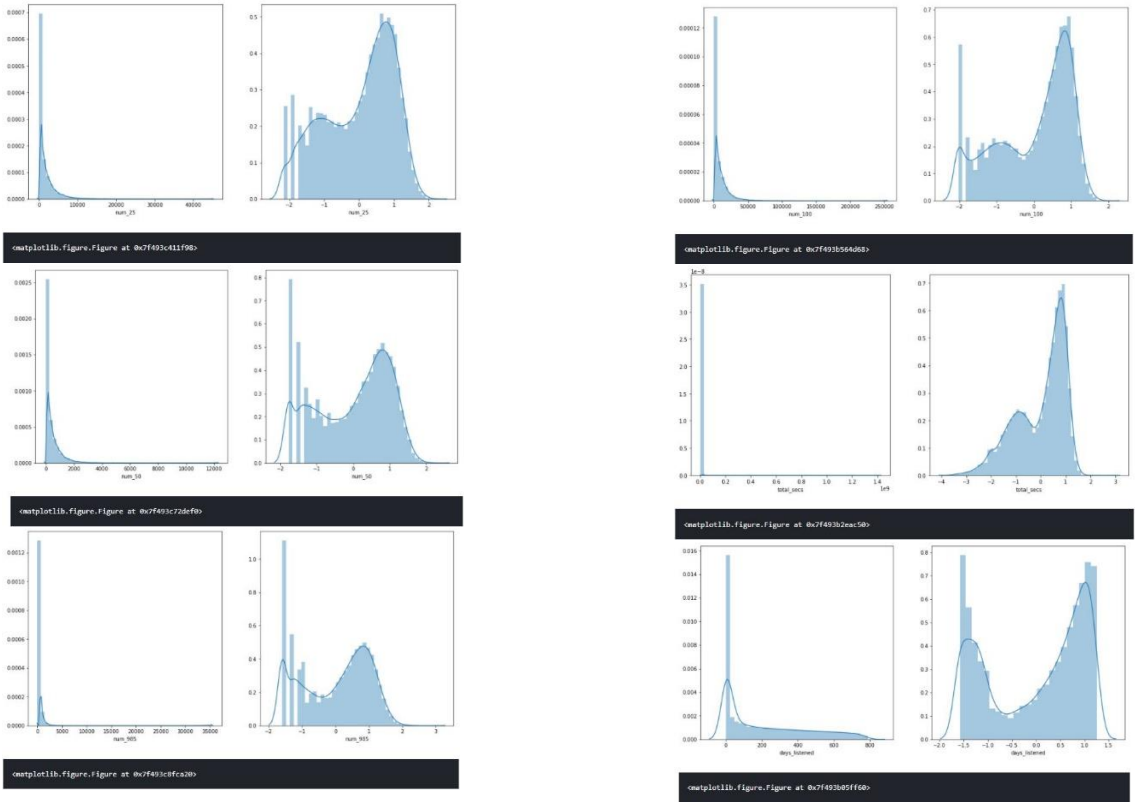

Figure b: The user_log dataset's columns before and after standardization

Package used: **sklearn.preprocessing.StandardScaler**

4.3. ENCODING CATEGORICAL VALUES

For encoding categorical values in the members and transactions dataset, we used the scikit-learn's preprocessing library. The preprocessing library's 'OneHotEncoder' was used to encode categorical values into numerical values so they can be fed into the classifiers. The other datasets had numerical columns only.

Package used: **sklearn.preprocessing.OneHotEncoder**

4.4. MERGING DATASETS

To get the training and testing datasets, the individual datasets needed to be merged together. After preprocessing the individual datasets, we merged the train dataset with user_logs, members and transactions dataset on the common feature 'msno'. For merging the datasets, we loaded them into individual data frames and then merged them one by one with the train dataset, giving us a huge train dataset (with 15 million records).

Similar steps were taken for the test dataset. The datasets user_logs, members and transactions were merged with 'sample_submission_zero' dataset.

Package Used: **Pandas**

4.5. REMOVING MISSING/DUPLICATE VALUES

After merging the datasets, there were a number of missing records and duplicate records. By removing the missing values, the dataset size went down considerably.

After deleting of missing values, there were multiple records for each user. Therefore, we decided to delete the duplicate records. The resulting test and training set dataset sizes went down from 4GB to 1GB (15 million records to 4 million records).

Package used: **Pandas** (dataFrame.drop_duplicates)

The 2 columns 'msno' and 'is_churn' were then deleted from the datasets after processing so that they can be used to train and test the model. The 'is_churn' column was extracted before deletion to be used as the class_label.

5. PROPOSED SOLUTIONS AND METHODS

The datasets provided in the competition have millions of records. Therefore, the processed training and test datasets were huge in size as well. Both have more than a million records. Due to this reason, we could only try and train four classifiers, details of which are discussed in detail below. Each of the classifiers took a considerable amount of time and processing power to train.

5.1. NEURAL NETWORKS

We used multi-layer perceptron classifier (MLP) as one of the classifiers for our project so that we can get a taste of deep learning in our project as well. Because of the size of our dataset, and considering that MLP takes longer time to train, the maximum number of layers we used in our project is 4.

Scikit-Learn package: **sklearn.neural_network.MLPClassifier**

5.2. BAGGING

Bagging classifier is a very powerful ensemble method that fits random subsets of the original dataset and predicting the result for each subset. The individual predictions are then aggregated together, which gives the final result. Since we could only try a handful of classifiers because of the size of datasets, we chose a strong classifier, i.e., bagging.

Scikit-Learn package: **sklearn.tree.BagingClassifier**

5.3. ADABOOST

Ensemble learners are very strong learners, that is why we used Adaboost classifier because it is a very powerful classification algorithm. It gives great results with little tweaking of the parameters.

Scikit-Learn package: **sklearn.ensemble.AdaBoostClassifier**

5.4. DECISION TREES

Decision trees are non-parametric classifiers, which perform great with categorical values as well. They create models faster than deep learning techniques and that is why we decided to use it as one of the classifier in our project.

Scikit-Learn package: **sklearn.tree.DecisionTreeClassifier**

6. EXPERIMENTAL RESULTS AND ANALYSIS

After feature engineering was done on the datasets and after raw data was pre-processed to get normalized, standardized and encoded data, the learners were trained to classify them. We trained and tested the datasets with classifiers like Neural Networks, Bagging, Adaboost and Decision Trees.

6.1. NEURAL NETWORKS

We used the multi-layer perceptron (MLP) classifier that is implemented in the scikit learn library. Following parameters were tuned using the Randomized Search function that is also provided by scikit-learn,

NAME	DECRIPTION
Hidden_layer_sizes	The ith element represents the number of neurons in the ith hidden layer
Activation	Activation function for the hidden layer
solver	The solver for weight optimization
Alpha	L2 penalty (regularization term) parameter
max_iter	Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations.

Different runs for the best parameters found by RandomizedSearchCV,

	Parameters					
#	**Hidden layer sizes**	**activation**	**solver**	**alpha**	**max_iter**	**Average Accuracy (%)**
1.	**(100, 150, 100, 50)**	**relu**	**lbfgs**	**1.0**	**500**	**99.9**
2.	(100, 150, 100, 50)	tanh	sgd	0.5	200	80.7
3.	(100, 50, 20)	logistic	lbfgs	0.05	500	92.9

Along with accuracy, we also considered precision and recall to evaluate the model. Following were the results obtained for precision and recall,

#	**Precision (average = 'micro') (%)**	**Precision (average = 'macro') (%)**	**Recall (average = 'micro') (%)**	**Recall (average= 'macro') (%)**
1	99.4	50	98.3	49.4
2	78.5	48.4	79.3	46.2
3	90.8	49.1	92.4	48.6

The highest accuracy achieved was 99.9%. The highest recall and precision achieved was 99.4% and 98.3% respectively. Due to the no true samples (is_churn=1) in the test dataset provided, the precision and recall was below 50% when the parameter average='macro'.

6.2. BAGGING

Bagging classifier was used because it a strong ensemble meta-estimator. Following parameters were tuned for the bagging classifier using the Randomized Search function that is also provided by scikit-learn,

NAME	DECRIPTION
Base_estimator	The base estimator to fit on random subsets of the dataset
Max_samples	The number of samples to draw from X to train each base estimator
N_estimators	The number of base estimators in the ensemble
bootstrap	Whether samples are drawn with replacement.

Different runs for the best parameters found by RandomizedSearchCV,

	Parameters				
#	**base_estimator**	**max_samples**	**n_estimators**	**bootstrap**	**Average Accuracy (%)**
1.	Logistic Regression	0.7	20	False	86.3
2.	KNN	0.5	5	True	92.1
3.	**Decision Tree**	**0.9**	**30**	**False**	**97.45**

Along with accuracy, we also considered precision and recall to evaluate the model. Following were the results obtained for precision and recall,

#	Precision (average = 'micro') (%)	Precision (average = 'macro') (%)	Recall (average = 'micro') (%)	Recall (average= 'macro') (%)
1	81.1	44.2	80.0	43.8
2	90.7	46.2	88.34	47.56
3	93.7	48.72	95.8	49.5

The highest accuracy achieved was 97.45%. The highest recall and precision achieved was 95.8% and 93.7% respectively. Due to the no true samples (is_churn=1) in the test dataset provided, the precision and recall was below 50% when the parameter average='macro'.

6.3. ADABOOST

Adaboost classifier is also an ensemble meta-estimator. It implements the algorithm called AdaBoost-SAMME. Following parameters were tuned using the Randomized Search function that is also provided by scikit-learn,

NAME	DECRIPTION
Base_estimator	The base estimator to fit on random subsets of the dataset
Max_samples	The number of samples to draw from X to train each base estimator
N_estimators	The number of base estimators in the ensemble
Bootstrap	Whether samples are drawn with replacement.

Different runs for the best parameters found by RandomizedSearchCV,

		Parameters				
#	**Cross Validation Fold**	**base_estimator**	**n_estimators**	**learning_rate**	**algorithm**	**Average Accuracy (%)**
1.	5	Decision Tree	100	1.5	SAMME	95.0
2.	5	Decision Tree	150	0.5	SAMME	95.0
3.	**5**	**Logistic Regression**	**200**	**1**	**SAMME.R**	**99.0**

Along with accuracy, we also considered precision and recall to evaluate the model. Following were the results obtained for precision and recall,

#	Precision (average = 'micro') (%)	Precision (average = 'macro') (%)	Recall (average = 'micro') (%)	Recall (average= 'macro') (%)
1	95	50	93.8	48.2
2	95	50	93.8	48.6
3	98.8	50	98.5	50

The highest accuracy achieved was 99.0%. The highest recall and precision achieved was 98.5% and 98.8% respectively. Due to the no true samples (is_churn=1) in the test dataset provided, the precision and recall was below or equal to 50% when the parameter average='macro'.

6.4. DECISION TREES

Decision trees performs great for binary classification that is why we used it in out project. Following parameters were tuned using the Randomized Search function that is also provided by scikit-learn,

NAME	DECRIPTION
Base_estimator	The base estimator to fit on random subsets of the dataset
Max_samples	The number of samples to draw from X to train each base estimator
N_estimators	The number of base estimators in the ensemble
Bootstrap	Whether samples are drawn with replacement.

Different runs for the best parameters found by RandomizedSearchCV,

		Parameters				
#	**Cross Validation Fold**	**criterion**	**min_samples_split**	**max_depth**	**splitter**	**Average Accuracy (%)**
1.	**5**	**entropy**	**10**	**10**	**best**	**99.37**
2.	5	entropy	2	50	random	97.29
3.	5	gini	2	20	best	96.68

Along with accuracy, we also considered precision and recall to evaluate the model. Following were the results obtained for precision and recall,

#	**Precision (average = 'micro') (%)**	**Precision (average = 'macro') (%)**	**Recall (average = 'micro') (%)**	**Recall (average= 'macro') (%)**
1	99.37	50	99	49.8
2	97.29	49.5	97.1	47.4
3	96.68	48.7	95.4	48

The highest accuracy achieved was 99.37%. The highest recall and precision achieved was 99% and 99.37% respectively. Due to the no true samples (is_churn=1) in the test dataset provided, the precision and recall was below or equal to 50% when the parameter average='macro'.

7. CONCLUSION

'Artificial Neural Network' (ANN) and Ensemble technique, 'Adaboost' gave the best results for churn prediction. We will submit our final algorithm from one of these two classifiers in the competition. Ensemble techniques, namely, 'Bagging' and 'AdaBoost', and ANN took more time to run compared to Decision Tree.

Data visualization tools helped greatly to figure out relations between different features and the output. As the class labels of test dataset all contain zeros, accuracies of all the classifiers are high but recall and precision values drops down significantly. A test set with positive samples ('is_churn'=1) also would train the models better.

Furthermore, the original size of the datasets were huge, pre-processing the datasets took a significant amount of our time. Even the processed datasets were too big for our systems to handle, which slowed down the training process significantly.

8. FUTURE WORK

- Process the dataset more to find out independent and meaningful features
- Try further to reduce the memory of the dataset in order to process faster
- After memory reduction, rry more classifiers for parameter tuning and classification
- Try other evaluation metrices e.g ROC, F1 measure, log_loss etc. for better understanding the results
- After merging all the datasets, try more feature engineering and data visualization

9. CONTRIBUTIONS

#	TASKS	TEAM MEMBER
1	Understanding the datasets	Arsalan, Shihab
2	Preprocessing data	Arsalan, Shihab
3	Project Status report	Arsalan, Shihab
4	Proposed solutions	Arsalan, Shihab
5	Classifier (Neural Network)	Arsalan
6	Classifier (Bagging)	Arsalan
7	Classifier (AdaBoost)	Shihab
8	Classifier (Decision Trees)	Shihab
9	Final Report	Arsalan, Shihab

10. REFERENCES

Scikit-Learn References,

- SVM Classifier, http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- Decision Tree Classifier, http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- Neural Network Classifier, http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
- Bagging Classifier, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

- AdaBoost Classifier, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
- Logistic Regression Classifier, http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- K-nearest Neighbors Classifier, http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- Model Evaluation, http://scikit-learn.org/stable/modules/model_evaluation.html
- Randomized Search, http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html#sklearn.model_selection.RandomizedSearchCV
- Grid Search, http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV

Kaggle references,

- https://www.kaggle.com/rastaman/churn-or-no-churn-exploration-data-analysis
- https://www.kaggle.com/jeru666/did-you-think-of-these-features
- https://www.kaggle.com/headsortails/should-i-stay-or-should-i-go-kkbox-eda
- https://www.kaggle.com/jagangupta/processing-huge-datasets-user-log
- https://www.kaggle.com/joshwilkins2013/churn-baby-churn-user-logs