Prakash Dhimal
Manav Garkel
George Mason University
CS 657 Mining Massive Datasets

Project Report: Churn Prediction

Introduction

User churning is a significant loss of revenue for music subscription companies. The cost associated with keeping current subscribers is usually much lower than the cost associated with acquiring new customers. Thus, if music subscription companies (or any companies selling a subscription based product) are able to predict which users are likely to churn (cancel subscription), they can pursue targeted offers to these customers to prevent them from churning. The goal of this project is to build a prediction model to predict if a user is likely to churn i.e. it is a binary classification problem. The dataset used is for a fictional company Sparkify. Sparkify contains user interaction logs. This dataset was created by Udacity. The goal of the project is to use this user log data to extract features and labels and use that dataset to build and validate various supervised binary classification models to predict user churn. The entire project was completed using Pyspark. The results were obtained from running the script on AWS EMR cluster.

Data

The data file provided was in JSON format. This JSON data was read into a pyspark dataframe. This JSON data contained the raw user interaction logs with the following fields:

```
root
 |-- artist: string (nullable = true)
 |-- auth: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- itemInSession: long (nullable = true)
 |-- lastName: string (nullable = true)
 |-- length: double (nullable = true)
 |-- level: string (nullable = true)
 |-- location: string (nullable = true)
 |-- method: string (nullable = true)
 |-- page: string (nullable = true)
 |-- registration: long (nullable = true)
 |-- sessionId: long (nullable = true)
 |-- song: string (nullable = true)
 |-- status: long (nullable = true)
 |-- ts: long (nullable = true)
 |-- userAgent: string (nullable = true)
 |-- userId: string (nullable = true)
```

An example row from the dataset is shown below:

```
-RECORD 0----------------------------
 artist        | Popol Vuh
 auth          | Logged In
 firstName     | Shlok
 gender        | M
 itemInSession | 278
 lastName      | Johnson
 length        | 524.32934
 level         | paid
 location      | Dallas-Fort Worth...
 method        | PUT
 page          | NextSong
 registration  | 1533734541000
 sessionId     | 22683
 song          | Ich mache einen S...
 status        | 200
 ts            | 1538352001000
 userAgent     | "Mozilla/5.0 (Win...
 userId        | 1749042
```

The most significant feature from these user interaction logs was the page feature. This feature holds all Sparkify pages that customers have visited. In order to build a model that predicts if a user is going to churn given the history of interactions with the service, we use the *Cancellation Confirmation* page to indicate if a particular user churned.

```
+-------------------------+
|page                     |
+-------------------------+
|Cancel                   |
|Submit Downgrade         |
|Thumbs Down              |
|Home                     |
|Downgrade                |
|Roll Advert              |
|Logout                  |
|Save Settings           |
|Cancellation Confirmation|
|About                   |
|Submit Registration     |
|Settings                |
|Login                   |
|Register                |
|Add to Playlist         |
|Add Friend              |
|NextSong                |
|Thumbs Up               |
|Help                    |
|Upgrade                 |
+-------------------------+
```

Data Cleaning

The *firstname*, *lastname*, and *id_copy* fields were dropped from the dataset as these are unique to each user and will not be significant in churn prediction. We do retain the userID field in order to construct features for every user in the data. Since we are building a model that focuses on users, we remove any rows with null/na values in the userID column.

```
data = data.drop(*['firstName', 'lastName', 'id_copy'])
data = data.filter(data["userId"] != "")
```

Additionally, the time in *registration* and *ts* fields is given in milliseconds. We drop all null values and convert the left over fields to seconds.

```
data = data.filter(data['registration'].isNotNull())
time_unit_udf = F.udf(lambda x: float(x/1000), DoubleType())
data = data.withColumn("registration",
time_unit_udf("registration")). \
    withColumn("ts", time_unit_udf("ts"))
```

Next, our goal is to extract the label field from the user interaction logs. As mentioned above, we used the *Cancellation Confirmation* page to indicate that the user decided to churn their subscription. There are only two values for this label, churned or not churned. Therefore this is going to be a boolean column represented by 1 for churned and 0 for not churned. A challenge is that a particular user has multiple entries, since this is a database of user activities log. A user may have visited a number of other pages before reaching the *Cancellation Confirmation* page. For a user that has churned, we want to make sure that all of his/her logs record 1 for the churn column. For this we will use the Window function from sql.window.

```
flag_cancelation_event = F.udf(lambda x:
    1 if x == "Cancellation Confirmation" else 0, IntegerType())
data = data.withColumn("churn", flag_cancelation_event("page"))
windowval = Window.partitionBy("userId") \
        .rangeBetween(Window.unboundedPreceding,
                      Window.unboundedFollowing)
data = data.withColumn("churn", Fsum("churn").over(windowval))
label = data \
    .select('userId', col('churn').alias('label')) \
    .dropDuplicates()
```

Finally, we were also able to observe that the total number of unique users in the data was ***22,277.***

Feature Engineering

After completing the data cleaning and extracting the label, the next step is to extract meaningful features from the user log data that we can use as features for the classification models. As mentioned above, due to the fact that the dataset is user log data, each user has multiple rows of data. To ensure that all the features extracted from here on our are on a per user basis, we use Pysparks *groupBy* function.

The first feature we extracted from the dataset computes the time since the user first registered their subscription. Users that have been using the service for longer periods of time tend to stay with the service as a paying customer, than those that recently signed up. Every company selling products and services to customers has an idea of a "lifetime value" of a customer. We converted the times to seconds above. After gaining the lifetime of a user, we convert that to days.

```
time_since_registration = data \
    .select('userId', 'registration', 'ts') \
    .withColumn('lifetime', (data.ts - data.registration)) \
    .groupBy('userId') \
    .agg({'lifetime': 'max'}) \
    .withColumnRenamed('max(lifetime)', 'lifetime') \
    .select('userId', (col('lifetime')/3600/24).alias('lifetime'))
```

The next feature we create is an *add friend* feature. It is in our nature to refer products and services that we enjoy using to close friends and family. Businesses not only rely on their customers coming back for more goods and services, but also rely on users referring the business to friends and family. We use the *page* column specifically looking for the *Add Friend* page.

The more songs a user listens to, the more likely they are to enjoy the streaming service and keep their subscription. Thus we add a *total songs listened* feature.

The more songs a user likes on the streaming service, the more it potentially implies that they enjoy their subscription and the value they obtain from it. They are more likely to keep their subscription if they like more songs. Thus we create a feature to count the number of songs a user gives a *thumbs_up* to.

Similarly, the more songs a user dislikes, the less likely they are to enjoy their subscription and cancel it. We create a feature to count the number of songs a user gives a *thumbs_down* to.

Playlist length counts the number of times a user visited the add to playlist page indicating that they added a song to their playlist. A long playlist implies a user is enjoying several songs and wants frequent access to them. This would likely lead to them keeping their subscription. Thus, we create a feature to compute the length of the playlist.

Average songs per session allows us to measure how long each user session lasts from when they open the application and start a session. Users logging in long sessions could imply that the user is enjoying the application and is less likely to cancel.

Artist count measures the *amount of different artists* the user listens to. A user listening to a wide variety of artists could potentially be enjoying the music application more and is likely to keep their subscription. On the other hand, a user listening to only a handful of artists may not be happy with the current subscription and would be likely to churn.

Lastly, the *total number of sessions* computes how many times the user has started a session on the app. More sessions implies more visits to the application meaning they are less likely to churn and vice-versa.

After creating the above mentioned features, we were able to combine them into one large dataframe with the following schema:

```
root
 |-- userId: string (nullable = true)
 |-- num_sessions: long (nullable = true)
 |-- artist_count: long (nullable = true)
 |-- avg_songs_played: double (nullable = false)
 |-- playlist_length: long (nullable = true)
 |-- add_friend: long (nullable = true)
 |-- num_thumb_down: long (nullable = true)
 |-- num_thumb_up: long (nullable = true)
 |-- total_songs: long (nullable = true)
 |-- lifetime: double (nullable = false)
 |-- label: long (nullable = true)
```

Below is the feature table with first five rows:

userId	Num_ sessions	artist_ count	Avg_songs_ played	Playlist_ length	Add_ friend	Num_thumb _down	Num_thumb _up	Total_ songs	lifetime	label
1000280	22	767	48.6666	25	14	33	53	1317	77.303	1
1002185	17	1205	104.5882	49	25	14	92	2080	65.751	0
1017805	3	223	83.3333	5	13	4	7	320	54.266	0
1030587	11	1071	163.5555	46	23	16	66	1752	131.597	0
1033297	5	215	47.2	7	4	3	10	299	116.144	0

Below is a breakdown of the label counts:

```
+-----+-----+
|label|count|
+-----+-----+
|    0|17274|
|    1| 5003|
+-----+-----+
```

Dropped features:

It is important to note that there were other features such as listening time, paid level [paid or free], and gender that were later removed after looking at the feature importance reported by some of our models.

```
Feature importance:
lifetime :  0.353377269155215
total_songs :  0.051463528854462266
num_thumb_up :  0.04919289834333239
num_thumb_down :  0.1967225971727057
add_friend :  0.1282489709189149
playlist_length :  0.051164349369040844
listen_time :  0.0
avg_songs_played :  0.07770908809851405
gender :  0.005321908564168713
artist_count :  0.01906238766775075
level :  0.0
num_sessions :  0.06773700185589532
```

Vector Assembling and Scaling

Few more steps needed to be carried out with the dataste before it could be used for modelling. The first step was to create a feature column using a vector assembler.

```
-RECORD 0--------------------------------
 userID           | 1000280
 unScaled_features | [77.3037731481481...
 label            | 1
```

Additionally, the features were also scaled using Pyspark's standard scaler.

```
-RECORD 0--------------------------------
 userID            | 1000280
 unScaled_features | [77.3037731481481...
 label             | 1
 features          | [1.89088370641545...
```

Modelling

The data is now ready to be fed into different classification models. In order to fine tune and validate the models, the dataset was divided into training and test sets with a split of ***80-20*** respectively. Additionally, for each of the models, ***grid search cross validation with three folds*** was used to find the best set of hyper-paramters for the models.

Five different models were picked:

- Gradient Boosting Trees,
- Logistic Regression,
- Linear SVC,
- RandomForestClassifier
- Hybrid Model

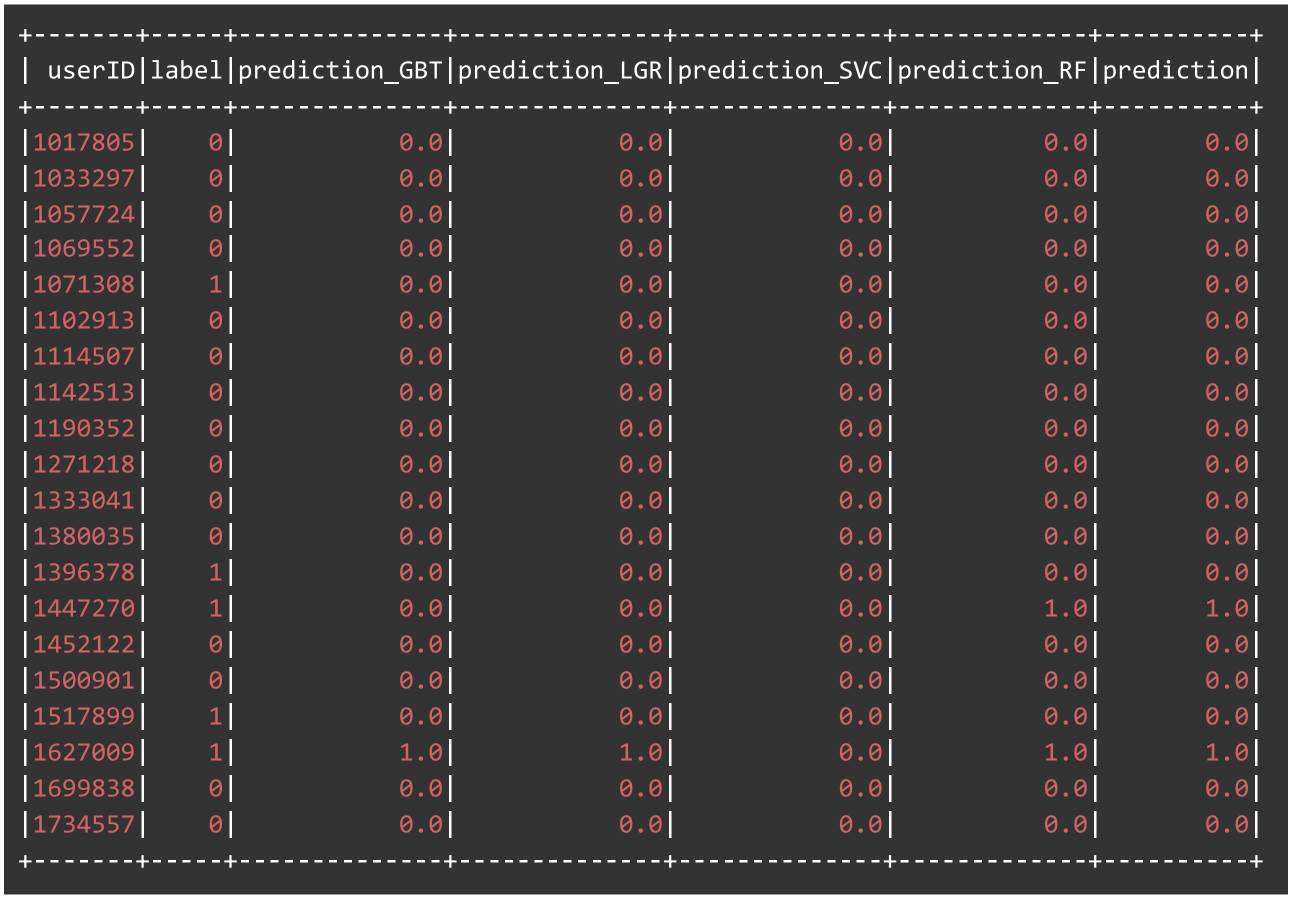
The hybrid model used was a simple OR function. This means that if any model has labelled if a user will churn, the hybrid model will label that user as churned. This was done intentionally to err on the side of caution i.e. if even a single model predicts that a user is likely to churn, we will label them accordingly. *This is done so that churners can be identified and targeted to ensure they do not cancel their subscription.* In this case, a few false positives is more economical for the company than false negatives i.e. missing a potential churner and losing their subscription is more costly than providing an already happy customer with target offers to keep them going.

We used accuracy and f1-score to evaluate the models. As can be seen in the class label table above, the classes are highly imbalanced. Thus, we are more focused on optimizing the F-1 score for each of our models.

Classifier	F-1 Score	Accuracy
Gradient Boosting Trees	0.80	0.82
Logistic Regression	0.68	0.78
Linear SVC	0.68	0.78
Random Forest Classifier	**0.83**	**0.83**
Hybrid Model	0.82	0.81

As can be seen above, logistic regression and linear SVC performed most poorly, whereas gradient boosting trees and random forest classifiers performed the best. Out of all the models, Random Forest classifier had the highest accuracy and F-1 score. The hybrid model was just 1% shy of the random forest classifier.

The poor performance from logistic regression and linearSVC indicates that the data may not be linearly separable. These models are generalized linear models that seek to find an optimal linear decision boundary between the two classes in the input space. Due to the poor performance of these models on this dataset, it is likely that the dataset is not linearly separable. In fact, it was observed that the SVM model did not predict any churners, whereas logistic regression predicted a very small number of churners.



```
+-------+-----+--------------+--------------+--------------+-------------+----------+
| userID|label|prediction_GBT|prediction_LGR|prediction_SVC|prediction_RF|prediction|
+-------+-----+--------------+--------------+--------------+-------------+----------+
|1017805|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1033297|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1057724|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1069552|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1071308|    1|           0.0|           0.0|           0.0|          0.0|       0.0|
|1102913|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1114507|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1142513|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1190352|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1271218|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1333041|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1380035|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1396378|    1|           0.0|           0.0|           0.0|          0.0|       0.0|
|1447270|    1|           0.0|           0.0|           0.0|          1.0|       1.0|
|1452122|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1500901|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1517899|    1|           0.0|           0.0|           0.0|          0.0|       0.0|
|1627009|    1|           1.0|           1.0|           0.0|          1.0|       1.0|
|1699838|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
|1734557|    0|           0.0|           0.0|           0.0|          0.0|       0.0|
+-------+-----+--------------+--------------+--------------+-------------+----------+
```

On the other hand, Gradient Boosted Trees and Random Forest classifiers produced significantly better results. Both of these classifiers utilize different ensemble strategies to create a classification model. Gradient boosted trees are based on using an ensemble of weak learners (high bias, low variance). At each iteration, a new learner is added to the model which builds on the results of the previous learner, thereby reducing the bias error across each learner. On the other hand, a random forest classifier works in somewhat of an opposite manner. Random forests, works with an ensemble of fully/almost fully grown decision trees (low bias, high variance). Each decision tree is trained on a random subset of the data and this uncorrelation among the trees allows the classifier to maximize the decrease in variance. These ensemble properties of gradient boosted trees and random forests allows them to outperform the other classifiers.