Machine Learning Final Project

Team 6 - Sean Jung, Jiayan Han, Ryan Chen, Kasandra Woo

12/19/20

Data Understanding/Preparation

Our first step was to gain an understanding of the type of data we were dealing with. We found that all variables were categorical variables. Additionally, we realized that there were some variables that would not have predictive power. We determined that these variables would not have predictive power because they were unique to each record. The data included in these variables act as a unique identifier for the record and therefore would not be useful in making predictions. Therefore, we decided to eliminate them.

The variables included: *id, device_ip,* and *device_id*. Next, we performed some exploratory analysis in order to better understand the underlying data. We looked at the distribution of every variable. We found that for many variables, a few categories within the variable accounted for the majority of the records. Additionally, we noticed that there were several variables with a very large number of categories. We realized that we could not turn each category in the variable into a dummy variable because we would have an issue with dimensionality.

We considered a couple of different options on how to approach this. We could use the distribution of the categories within each variable to make decisions on how to handle these variables. If there are a few categories that account for the majority of the records, we could group the remaining categories together as "*other*". We also could use the frequency of each category within the variable. We could group together these categories by their frequency. Those

categories with similar frequencies would be grouped together. Once the categories were grouped together, we could reassign them to the group and create a smaller list of dummy variables. We also considered the option of grouping by probability.

We also considered whether or not we had an issue with an imbalance Y variable. However, after looking into our training data, we realized we had enough of each class to make predictions. While exploring our data, we also discovered that there were categories that existed in the test data that did not exist in the training data. We considered our options of how to handle this issue. One approach would be to change these previously unseen categories (in the training data) to the most common category. Another approach would be to put these in an "other" category.

Data Transformation of Base Models

After considering our options, we made decisions on how to transform our data. We decided to remove variables that would not have predictive power. Then, we decided to use the approach of grouping categories within each variable into three buckets based on frequency: Most common, less common, least common. Additionally, for variables that have categories in the test data that were not in the training data, we changed the values to the "less frequent" category. After selecting a method for transforming the data, we focused on figuring out which model would give us the best predictions. We evaluated the performance of the model using the metric of log loss. We performed hyperparameter tuning to find the best performing models.

Base Models and Evaluation

After transforming the data, we used samples of the training data and split it into training, test, and validation data. We then used those same datasets to test the performance of different models. This was so that the performance could be compared to one another. Additionally, we

made sure to evaluate the performance using the same metric - log loss. This consistency allows us to make better comparisons between our models.

Logistic Regression

One method we tried was using logistic regression to make predictions. We tuned the parameters *"C"* and *"penalty"* to alter the model. The *"C"* parameter is the inverse of the regularization strength, and the *"penalty"* parameter determines the type of regularization method used. Regularization is performed in order to help avoid overfitting. We changed the parameter values and after changing the parameter checked to see if the model performed better or worse. For example, for the *"C"* parameter, if the model performed better, we increased the parameter value. If the performance worsened, we stepped the value back and checked the performance again. This was repeated until the performance did not improve anymore. The performance was measured by using log loss. The smaller the loss, the better the performance. After doing this, the best performing model out of the logistic regression models tested had a log loss of 0.425.

```
In [41]: # Model 15 *BEST MODEL SO FAR
         clf15 = LogisticRegression(multi_class='ovr', C=575, solver="liblinear", penalty = 'l2')
         clf15 = clf15.fit(trainX, trainY)

In [42]: y_pred_prob15 = clf15.fit(trainX, trainY).predict_proba(valX)
         sklearn.metrics.log_loss(valY, y_pred_prob15, normalize=True, sample_weight=None, labels=None)

Out[42]: 0.42519176410131376
```

Random Forest

Another model we evaluated was a random forest. The main idea of this approach is that it forces the model to take different paths through the variables. Again, we tuned the parameters *"n_estimators"* and *"min_samples_split"* and checked the performance after changing the value

of the parameter using log loss. Out of the different versions of the random forest model we evaluated, we got the best results from the model where *n_estimators = 700* and *min_samples_split = 8*. This model had a log loss of 0.424. This model performed better than the logistic regression model.

```
In [41]: # Train model
         clf12 = RandomForestClassifier(n_estimators = 700, min_samples_split = 8, random_state=42)
         clf12 = clf12.fit(traindataX, traindataY)

In [42]: # Make predictions with valdata
         pred_prob12 = clf12.fit(traindataX, traindataY).predict_proba(valdataX)
         sklearn.metrics.log_loss(valdataY, pred_prob12 , normalize=True, sample_weight=None, labels=None)

Out[42]: 0.4244950061275683
```

XGBoost

We also implemented XGBoost, which is an implementation of gradient boosted decision trees, designed for speed and performance. We performed parameter tuning for the XGBoost model by using GridSearchCV in Python to search through various values for different parameters. We tried tuning *'learning_rate', 'max_depth'*, and *'min_child_weight'*, using the log loss to make the parameter selections. Our final tuned parameters were *'learning_rate' = 0.3*, *'max_depth' = 8*, and *'min_child_weight' = 7*.

When training our XGBoost model, we applied a boost of 1500 rounds. To avoid over-fitting, we also used a validation data set and set early stopping to 80 rounds. This means that if validation performance is not able to be improved in 80 consecutive rounds, then the algorithm will use the last best model. Boosting for our model stopped after 1014 rounds. Lastly, we made our predictions using the validation data set, and our XGBoost model resulted in a log loss of 0.421.

Decision tree classifier

```
In [8]: # Optimize decision tree

        from sklearn.linear_model import LogisticRegression
        from sklearn.model_selection import GridSearchCV, KFold, cross_val_score
        from sklearn.tree import DecisionTreeClassifier

        gs = GridSearchCV(estimator=DecisionTreeClassifier(random_state=0),
                          param_grid=[{'max_depth': [4, 6, 8, 10, None], 'criterion':['gini','entropy'],
                                       'min_samples_leaf':[1,2,3,4,5],
                                       'min_samples_split':[2,3,4,5]}],
                           scoring='neg_log_loss')

        gs = gs.fit(X,y)
        print("Optimal Parameter: ", gs.best_params_)
        print("Optimal Estimator: ", gs.best_estimator_)

        Optimal Parameter:  {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_spli
        t': 2}
        Optimal Estimator:  DecisionTreeClassifier(max_depth=10, min_samples_leaf=5, random_state=0)
```

```
In [11]: from sklearn.tree import DecisionTreeClassifier
         import sklearn
         from sklearn.metrics import log_loss

         dt = DecisionTreeClassifier(max_depth=10, min_samples_leaf=5, random_state=0)

         pred_dt = dt.fit(X,y).predict_proba(X_test)

         sklearn.metrics.log_loss(y_true, pred_dt, normalize=True, sample_weight=None, labels=None)

Out[11]: 0.42261763178623923
```

We optimized the decision tree by trying different parameters such as maximum depth, minimum samples per leaf and minimum samples per split. When *max_depth=10*, *min_samples_leaf=5*, the model gives us the best log loss, which is 0.422.

Support Vector Machine:

SVM runs extremely slow even with just a few thousand records. We eventually increase the number of records to 10000 due to the limitation of our computing power, and was able to optimize it and get a log loss of 0.45.

```
# Optimize SVM

from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn.svm import SVC

op_sup = [{'kernel': ['rbf','linear','poly','sigmoid'], 'gamma': ['scale', 'auto
                      'C': [1, 10, 100, 1000, 10000]}]

sup = GridSearchCV(SVC(random_state=0,probability=True),
                   op_sup,
                    scoring='neg_log_loss')
sup = sup.fit(train_X,train_y)
print("Optimal Parameter: ", sup.best_params_)
print("Optimal Estimator: ", sup.best_estimator_)
```

```
# Performance of SVM

from sklearn import svm
import sklearn
from sklearn.metrics import log_loss

svc = svm.SVC(probability=True, C=100, gamma='scale', kernel='rbf', random_state
y_pred3 = svc.fit(train_X, train_y).predict_proba(X_sample)

sklearn.metrics.log_loss(y_sample, y_pred3, normalize=True, sample_weight=None,
```

```
0.45752086747900683
```

Neural nets:

Also, neural nets don't run much faster but it was able to include more records than SVM. At the end, a NN model with half a million records was developed and it also gave us a log loss of 0.45.

```
# Optimize Neural Nets

import numpy as np
from sklearn.neural_network import MLPClassifier
import sklearn
from sklearn.metrics import log_loss
from sklearn.model_selection import GridSearchCV

parameters = {'solver': ['lbfgs','sgd', 'adam'], 'max_iter': [500,1000,1500], 'a
              'hidden_layer_sizes':np.arange(5, 12), 'activation': ['identity',

net = GridSearchCV(MLPClassifier(random_state=1), parameters, n_jobs=-1,
                   scoring='neg_log_loss')

gs_net = net.fit(train_X, train_y)

print("Optimal Estimator: ", gs_net.best_estimator_)
```

```
# Performance of Neural Nets

net_1 = MLPClassifier(alpha=0.1, hidden_layer_sizes=5, max_iter=500, random_stat

y_pred6 = net_1.fit(train_X, train_y).predict_proba(X_sample)

sklearn.metrics.log_loss(y_sample, y_pred6, normalize=True, sample_weight=None,
```

```
0.45602347354500145
```

Naive Bayes classifier:

Naive bayes's performance was extremely poor so we didn't put too much effect into optimizing it.

In [4]:
```
# Performance of Naive Baynes

from sklearn.naive_bayes import GaussianNB
import sklearn
from sklearn.metrics import log_loss

gnb = GaussianNB()
y_pred2 = gnb.fit(X, y).predict_proba(X_test)

sklearn.metrics.log_loss(y_true, y_pred2, normalize=True, sample_weight=None, labels=None)
```

Out[4]:
```
2.195075728470081
```

Final Predictions and Conclusion

Our XGBoost model performed the best compared to the other models we evaluated. Therefore, we selected the XGBoost model to make our final predictions. We transformed the test data and then used it to make these predictions.

Appendix

- **Project-Code1-Team6.ipynb**
    - Created logistic regression models. Changed values of parameters and checked performance using log loss.
- **Project-Code2-Team6.ipynb**
    - Created random forest models and changed the values of the parameters to improve the model.
- **Project-Code3-Team6.ipynb**
    - Created XGBoost model, changed parameters and checked performance using log loss. Made predictions on test data.
- **Project-Code4-Team6.ipynb**
    - Created models based on the concepts of neural nets, naive bayes, support vector machine and decision tree.
- **Project-Code5-Team6.r**
    - Data cleaning and data preparation