Springboard Data Science Career Track | Nikki Seegars

Capstone Project I: Machine Learning

Model Selection

The purpose of this project is to use Fannie Mae loan data to predict loans to risky loans and identify the applications that are at higher risk of default. This is a classification problem with data that has labels. Therefore several different supervised machine learning models were likely candidates including logistic regression, k nearest neighbors, support vector machine, decision tree, and random forest. Although the question to this problem is binary, with each loan being classified as defaulted or not defaulted, the multiclass algorithm, random forest was chosen as the classification model. Random forest is an ensemble model that combines multiple decision trees and uses majority vote to determine the final classification for each sample. It is a highly scalable and simplistic algorithm in which its accuracy increases as the number of trees used increases.

Features and Target

The number of features in the model comes from two sets of data files, one containing the loan acquisition data and another containing monthly loan performance data. Initially there were 55 features in the dataframe, however the number was reduced to 29 then to 11 to determine the prediction. By using pandas profiling to quickly review some of the key statistics , many features in the datasets that were either highly correlated or high cardinality were dropped from the dataframe. The initial interest value was not included in the prediction due to its high correlation with the ultimate classification and this value would not have been available prior to the mortgage loan being approved.

The target value was created from the loans in which a foreclosure date was captured. This date was changed to a 1 in a new series labeled as “Classification”. All other loans were classified as a 0.

```
# Identify features and target to be included in classification model
features = ['Borrower_Credit_Score', 'Coborrower_Credit_Score','Orig_Debt_Inc_Ratio','Orig_LTV','Orig_UPB',
           'First_Time_Home_Buyer_Ind', 'Loan_Purpose', 'Property_Type','Num_Borrowers','Occupancy_Type', 'Zip_Code']
X = df[features]
y = df['Classification']
```

Undersampling

The initial run of the RandomForestClassifier generated a very low f1 score (composite score representing both recall and precision) of 4% for the default classification due to most of the training dataset loans being classified as not default. To counter this imbalance, undersampling with the NearMiss module was used to match the number of loans not defaulted to the number that were defaulted.

	precision	recall	f1-score	support
0	0.97	1.00	0.98	70885
1	0.68	0.02	0.04	2292
avg / total	0.96	0.97	0.95	73177

```
# Resample data for undersampling
from imblearn.under_sampling import NearMiss
nr = NearMiss()
X, y = nr.fit_sample(X, y)
```

Hyperparameters

The RandomForestClassifier in Python has many hyperparameters, which are the parameters that are set before the machine learning model is executed. The hyperparameters that were focused on in this project are n_estimators (the number of trees), max_features (maximum number of features to use per each random sampling), and min_samples_split (minimum number of samples to use at each node split). All other hyperparameters were set at their defaults.

In the effort to limit processing time, the number of trees was evaluated first by looking at a small set of values and calculating each one's f1 score to capture the one with the highest value. The values of 10, 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000 were evaluated with 400 returning with the highest f1 score.

Next, nested for loops were used to determine the best max_features and min_samples_split for n_estimators of 400.

```
# Random forest classifier
def random_forest(n_estimator, max_feature, min_samples_split):
    global y_test, y_predict, model
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=0)
    model = RandomForestClassifier(n_estimators=n_estimator, random_state=0, max_features=max_feature,
                                   min_samples_split=min_samples_split)
    model.fit(X_train, y_train)
    y_predict = model.predict(X_test)
    return y_test, y_predict, model
```

```
# Tune max_features and min_sample_splits to determine highest f1 score
max_features = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
min_samples_splits = [2, 4, 8, 16, 32, 64, 128, 256]
score = 0
max_feat = 0
min_ss = 0

for max_feature in max_features:
    for min_samples_split in min_samples_splits:
        random_forest(n, max_feature, min_samples_split)
        f1 = f1_score(y_test, y_predict)
        if f1 > score:
            score = f1
            max_feat = max_feature
            min_ss = min_samples_split
print(score, ': f1_score')
print(max_feat, ': max_feature')
print(min_ss, ': min_sample_split')
```

The final f1 score of 79.7% for the default classification was obtained with n_estimators = 400, max_features=2, and min_samples_split=64.

The model's feature importance values was calculated to highlight how much weight each feature contributed to the model.

```
for i in range(len(features)):
    print(round(model.feature_importances_[i],4), features[i])
```

```
0.1412 Borrower_Credit_Score
0.1347 Coborrower_Credit_Score
0.0423 Orig_Debt_Inc_Ratio
0.1254 Orig_LTV
0.3142 Orig_UPB
0.0082 First_Time_Home_Buyer_Ind
0.049 Loan_Purpose
0.0183 Property_Type
0.009 Num_Borrowers
0.0322 Occupancy_Type
0.1256 Zip_Code
```