Liz Garcia

Udacity Data Science Nanodegree Program

Starbucks Capstone Challenge

Introduction

The purpose of the Starbucks Capstone Challenge is to use the data to identify which groups of people are most responsive to which type of offer, and how best to present each type of offer. Starbucks wants to understand their customer on an individual level to understand what type of offers excite them the most. The data used in this capstone project is simulated customer behavior on the Starbucks mobile reward app.

Business Understanding

The goal of this project is to identify which Starbucks' offer resonates with which demographic on the mobile app by understanding the customer attributes and buying behavior, and how much that customer will spend based on their demographics and offer type. My strategy was to ***identify the demographic data*** for the first question in a series of charts in the "Data Understanding" section of this blog. The second half of this blog will address the second question, ***customer attributes and buying behavior***, which is solved using classification and machine learning techniques.

Data Understanding

This project contained three files:

The 'profile.json' file: Rewards Program Users (17000 users x 5 fields)

The 'portfolio.json' file: Offers sent during 30-day test period (10 offers x 6 fields)

The 'transcript.json' file: Event log (306648 events x 4 fields)

Data Preparation

Data Cleaning

After reviewing the raw data in each of the files, the next step was to clean, format, and rename the data.

For the ‘profile.json’ file:

- Customer data with missing information such as income and gender attributes.
- Renamed ‘id’ column to ‘customerid’
- Converted ‘became_member_on’ column to a datetime object
- Converted ‘gender’ column to numerical

For the ‘portfolio.json’ file:

- Changed the ‘id’ column name to ‘offerid’
- Removed underscores from column names
- One hot encoded the ‘offertype’ column and ‘channels’ column

For the ‘transcript.json’ file:

- Changed ‘person’ column to ‘customerid’
- Changed ‘time’ column to ‘timedays’
- Removed ‘customer id’ data that are not in the customer profile dataframe
- Converted time variable from hours to days
- Created two dataframes for offers and customer transaction events
- One hot encoded offer events

Modeling

Identifying the demographic data

To Identify the customer demographic, I wanted to plot features such as age, gender, membership timeline in the charts below:

Customer attributes and buying behavior

Communicating the customer attributes and buying behavior required me to use machine learning techniques that was taught in this course. The process I took are as outlined:

- Transformed skewed continuous features for ‘membershipstartyear’ and ‘membershipstartMonth’ columns
- Normalized numerical features
- Shuffled and split data
- Evaluated model performance

    - Naïve Predictor Performance
    - Created a training and predicting pipeline
    - Initial Model Evaluation
- Improving Results (grid search)

Evaluation

For the evaluation phase, I observed 4 algorithms and identified which was the best model to use. Two kpi's I used to identify the best model were accuracy and the F-beta score. For the F-beta score, beta was set to 0.5 to include offers to be sent to customers where income is above average.

- **Accuracy:** Percentage of total items classified correctly

Accuracy = (TP+TN)/(N+P)

- **F-beta score** as a metric that considers both precision and recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

Deployment

The following four algorithms were used:

1. Naive Predictor

A Naive predictor assumes every offer being sent is successful irrespective of any features and this kind of sets a baseline to evaluate the performance of the other two models that I build.

Accuracy: 0.464

F-Score: 0.520

2. Decision Trees (supervised learner):

- real-world application: predicting Default Risk in banks like provide a recommendation on which loan applicants the bank should lend to.
- Pros: able to handle both numerical and categorical data. People can easily understand why a decision tree classifies an instance as belonging to a specific class
- Cons: unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal decision tree. Calculations can get very complex.
- This model is a good candidate because a decision tree can handle both numerical and categorical data and the data content numerical and categorical data.

3. Random Forest (supervised learner):

- Computer Vision.
- Pros: Parallel training is possible, Work with large datasets with lot of attributes.
- Cons: Needs time to train since it builds multiple classifiers.
- Given that we have a large dataset such that a single Decision Tree will most likely overfit, Random Forests will perform comparatively better as it is an ensemble of Trees trained on samples of the dataset.

4. AdaBoost (supervised learner):

- real-world application: it is used in the areas of video and image recognition.
- Pros: it can be used in conjunction with many other types of learning algorithms to improve performance in some problems it can be less susceptible to the overfitting problem than other learning algorithms.
- Cons: AdaBoost is sensitive to noisy data and outliers.
- This model is a good candidate because it will be able to take the full dataset. And multiple weak learners can help make one strong learner. Also, the training data is of a high-quality (there are no invalid or missing entries).

Creating a Training and Predicting Pipeline

In the code block below:

- Import fbeta_score and accuracy_score from sklearn.metrics.
- Fit the learner to the sampled training data and record the training time.
- Perform predictions on the test data X_test, and also on the first 300 training points X_train[:300].
- Record the total prediction time.
- Calculate the accuracy score for both the training subset and testing set.
- Calculate the F-score for both the training subset and testing set.
- Make sure that you set the beta parameter!

```
from sklearn.metrics import fbeta_score, accuracy_score
from pprint import pprint
from time import time
def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    '''
    inputs:
       - learner: the learning algorithm to be trained and predicted on
       - sample_size: the size of samples (number) to be drawn from training set
       - X_train: features training set
       - y_train: income training set
       - X_test: features testing set
       - y_test: income testing set
    '''
    results = {}
    beta = 0.5
    #Fit the learner to the training data using slicing with 'sample_size'
    start = time() # Get start time
    learner = learner.fit(X_train[:sample_size], y_train[:sample_size])
    end = time() # Get end time
    #Calculate the training time
    results['train_time'] = end - start
    # Get the predictions on the test set,then get predictions on the first 300 training samples
    start = time() # Get start time
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train[:300])
    end = time() # Get end time
    #Calculate the total prediction time
    results['pred_time'] = end - start
    #Compute accuracy on the first 300 training samples
    results['acc_train'] = accuracy_score(y_train[:300], predictions_train)
    #Compute accuracy on test set
    results['acc_test'] = accuracy_score(y_test, predictions_test)
    #Compute F-score on the the first 300 training samples
    results['f_train'] = fbeta_score(y_train[:300], predictions_train, beta=beta)
    # Compute F-score on the test set
    results['f_test'] = fbeta_score(y_test, predictions_test, beta=beta)
    # Success
    print (learner.__class__.__name__)
    print(" trained on ")
    print(sample_size)
    print(" samples.")
    # Print results
    pprint(results)
    # Return the results
    return results
```

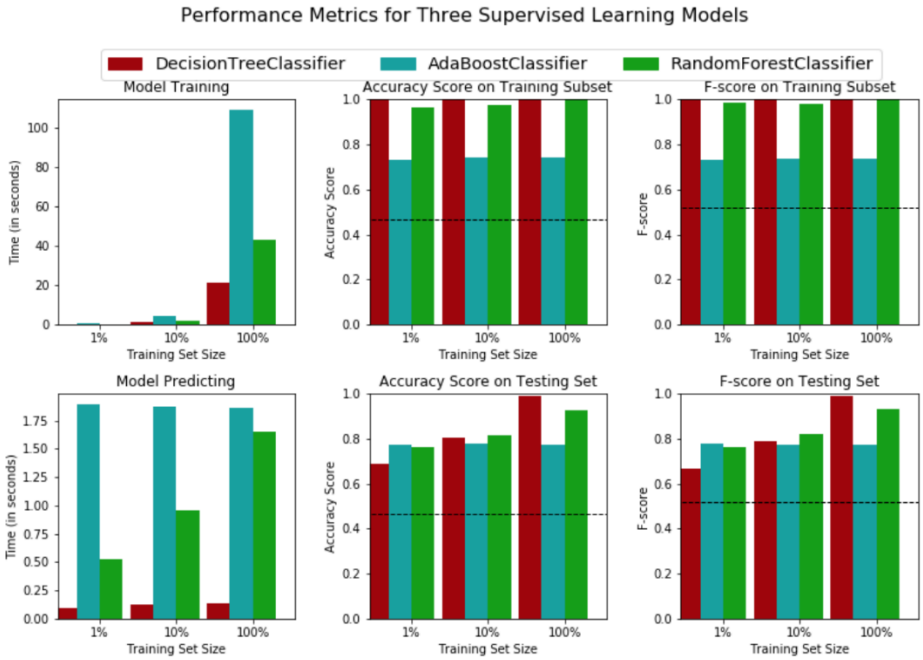
Initial Model Evaluation

In the code cell, you will need to implement the following:

- Import the three supervised learning models you've discussed in the previous section.
- Initialize the three models and store them in 'clf_A', 'clf_B', and 'clf_C'.
- Use a 'random_state' for each model you use, if provided.
- **Note:** Use the default settings for each model — you will tune one specific model in a later section.
- Calculate the number of records equal to 1%, 10%, and 100% of the training data.
- Store those values in 'samples_1', 'samples_10', and 'samples_100' respectively.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier

# TODO: Initialize the three models
clf_A = SVC(random_state = 40)
clf_B = AdaBoostClassifier (random_state = 40)
clf_C = RandomForestClassifier(random_state = 40)

# TODO: Calculate the number of samples for 1%, 10%, and 100% of the training data
# HINT: samples_100 is the entire training set i.e. len(y_train)
# HINT: samples_10 is 10% of samples_100 (ensure to set the count of the values to be `int` and not `float`)
# HINT: samples_1 is 1% of samples_100 (ensure to set the count of the values to be `int` and not `float`)
samples_100 = int(len(X_train))
samples_10 = int(len(X_train)/10)
samples_1 = int(len(X_train)/100)

# Collect results on the learners
results = {}
for clf in [clf_A, clf_B, clf_C]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([samples_1, samples_10, samples_100]):
        results[clf_name][i] = \
        train_predict(clf, samples, X_train, y_train, X_test, y_test)

# Run metrics visualization for the three supervised learning models chosen
evaluate(results, accuracy, fscore)
```

The Results:



- F-score on the testing when 100% of the training data is highest in Random Forest Classifier and Decision Trees.
  Decision Trees model Predicting time and training time is Less than another.
  With high accuracy and F-score and time, Random Forest Classifier is the most appropriate supervised learning model for the this problem.
  The best model is Random Forest Classifier Model.

Improving Results

Model Tuning

A Random Forest Classifier's instance is created and parameters like n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf are tuned to fit the training data using RandomizedSearchCV.

- max_features (auto): The number of features to consider when looking for the best split.
  - max_depth : The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- n_estimators: The number of trees in the forest.
- min_samples_split: The minimum number of samples required to split an internal node.
- min_samples_leaf: The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

Model built is as below:

```
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer

# Initialize the classifier
clf = RandomForestClassifier(random_state=2)
#Create the parameters list you wish to tune
max_depth=[5,8]
max_depth.append(None)
scorer = make_scorer(fbeta_score, beta = 0.5)
parameters={'max_features': ['auto'],
            'max_depth' : max_depth,
            'n_estimators': [10,50],
            'min_samples_split': [2,10],
            'min_samples_leaf': [1,4],
           }
# Perform grid search on the classifier using 'scorer' as the scoring method
grid_obj = GridSearchCV(clf, parameters, scoring=scorer, cv=3, verbose=2)
# Fit the grid search object to the training data and find the optimal parameters
grid_fit = grid_obj.fit(X_train, y_train)
# Get the estimator
best_clf = grid_fit.best_estimator_
# Make predictions using the unoptimized and model
predictions = (clf.fit(X_train, y_train)).predict(X_test)
best_predictions = best_clf.predict(X_test)
# Report the before-and-afterscores
print ("Unoptimized model\n")
print ("Accuracy score on testing data:", (accuracy_score(y_test, predictions)))
print ("F-score on testing data: ",format(fbeta_score(y_test, predictions, beta = 0.5)))
print ("\nOptimized Model\n")
print ("Final accuracy score on the testing data: ",(accuracy_score(y_test, best_predictions)))
print ("Final F-score on the testing data:",(fbeta_score(y_test, best_predictions, beta = 0.5)))
# Report scores on training dataset
train_predictions = best_clf.predict(X_train)
print ("\nTraining Data\n")
print ("Final accuracy score on the training data: ",(accuracy_score(y_train, train_predictions)))
print ("Final F-score on the training data: ",(fbeta_score(y_train, train_predictions, beta = 0.5)))
```

```
Unoptimized model

Accuracy score on testing data: 0.928714399218
F-score on testing data:  0.9305657443987535

Optimized Model

Final accuracy score on the testing data:  0.943577343845
Final F-score on the testing data: 0.93914596824

Training Data

Final accuracy score on the training data:  0.999909278862
Final F-score on the training data:  0.999909169036
```

- The optimized model has an accuracy of 94.36% on the test set and an F-score of 0.939.
- Theses scores are marginally better little than those of unoptimized model.
- But, as compared to the naive predictor, the optimized model performs really well. The naive predictor has accuracy of < 46%. The F-score too is much better, for the optimized model.

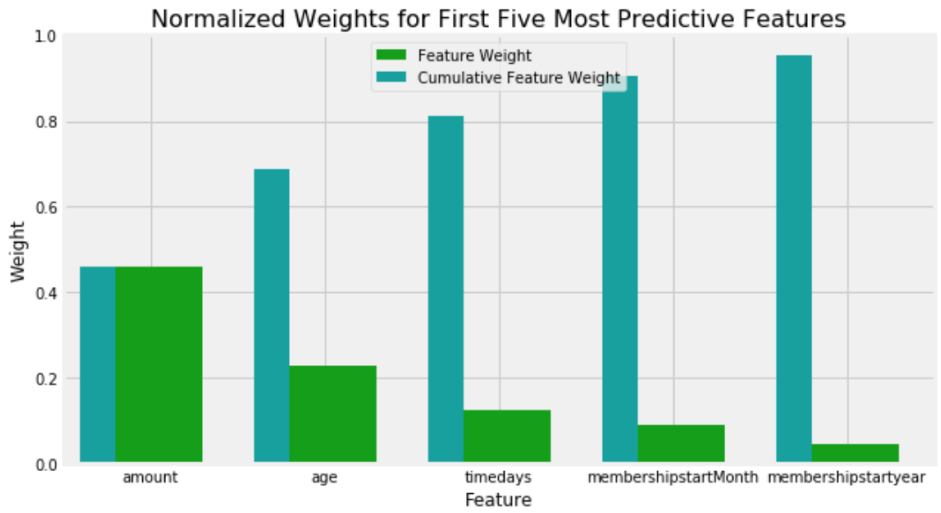
Feature Importance

Extracting Feature Importance

in this code as below:

- Import a supervised learning model from sklearn if it is different from the three used earlier.
- Train the supervised model on the entire training set.
- Extract the feature importances using '.feature_importances_'.

Normalized Weights for First Five Most Predictive Features:



Feature Selection

```
from sklearn.base import clone

# Reduce the feature space
X_train_reduced = X_train[X_train.columns.values[(np.argsort(importances)[::-1])[:5]]]
X_test_reduced = X_test[X_test.columns.values[(np.argsort(importances)[::-1])[:5]]]

# Train on the "best" model found from grid search earlier
clf = (clone(best_clf)).fit(X_train_reduced, y_train)

# Make new predictions
reduced_predictions = clf.predict(X_test_reduced)

# Report scores from the final model using both versions of data
print ("Final Model trained on full data\n------")
print ("Accuracy on testing data: {:.4f}",(accuracy_score(y_test, best_predictions)))
print ("F-score on testing data: {:.4f}",(fbeta_score(y_test, best_predictions, beta = 0.5)))
print ("\nFinal Model trained on reduced data\n------")
print ("Accuracy on testing data: {:.4f}",(accuracy_score(y_test, reduced_predictions)))
print( "F-score on testing data: {:.4f}",(fbeta_score(y_test, reduced_predictions, beta = 0.5)))
```

The Results:

```
Final Model trained on full data
------
Accuracy on testing data: {:.4f} 0.943577343845
F-score on testing data: {:.4f} 0.93914596824

Final Model trained on reduced data
------
Accuracy on testing data: {:.4f} 0.985175253636
F-score on testing data: {:.4f} 0.986222762852
```

Note that the change is small in model's F-score and accuracy score on the reduced data using only five features compare to those same scores when all features are used. If training time was a factor, I use the reduced data as your training set.

Results

My analysis suggests that the resulting random forest model has an training data accuracy of 0.944 and an F-score of 0.939. The test data set accuracy of 0.929 and F-score of 0.931 suggests that the random forest model I constructed did not overfit the training data.

Conclusion

In conclusion, the random forest model predicted how much someone will spend based on their demographics and offer type. The strategy for solving this problem was two-fold: the first step was to combine the portfolio, profile, and transaction data and the second step involved assessing the accuracy and F-score from the chosen model.

After comparing the results from each model, the random forest model had the best training data accuracy, F-score and training time. The final model trained on reduced data, based on the random forest model, had accuracy of 0.985 and an F-score of 0.986. To further improve this project, I would recommend selecting and evaluating different features such as real income.