# Wells Fargo NLP and Machine Learning Project

Yi-Ching (Leslie) Lin

University of Maryland – College Park

Master of Science in Information Systems

## Table of Contents

## 1. Executive Summary

The bank systems and customer experience nowadays are getting more advanced with more functions to improve customer satisfaction. One of the important functions that customers enjoy from their bank is the summary of their spending in a period. Customer can view their spending summary of what percent their spending is on what categories. This service not only enables customers to realize if they are overspending in a certain category but also save customers a bunch of time to document their daily spending. This service provides precise and informed information for customers to manage their finances, which plays a crucial role for every individual and household.

While to make sure this service works out great, the most important task is to make sure each transaction is classified into the correct category. With this goal in mind, the main object of this project is to correctly categorize each transaction based on historical data, such as spending amount and brand name. In this project, I mainly focused on brand names. I use NLTK and linear SVM model with text feature (coalesced_brand) in this project.

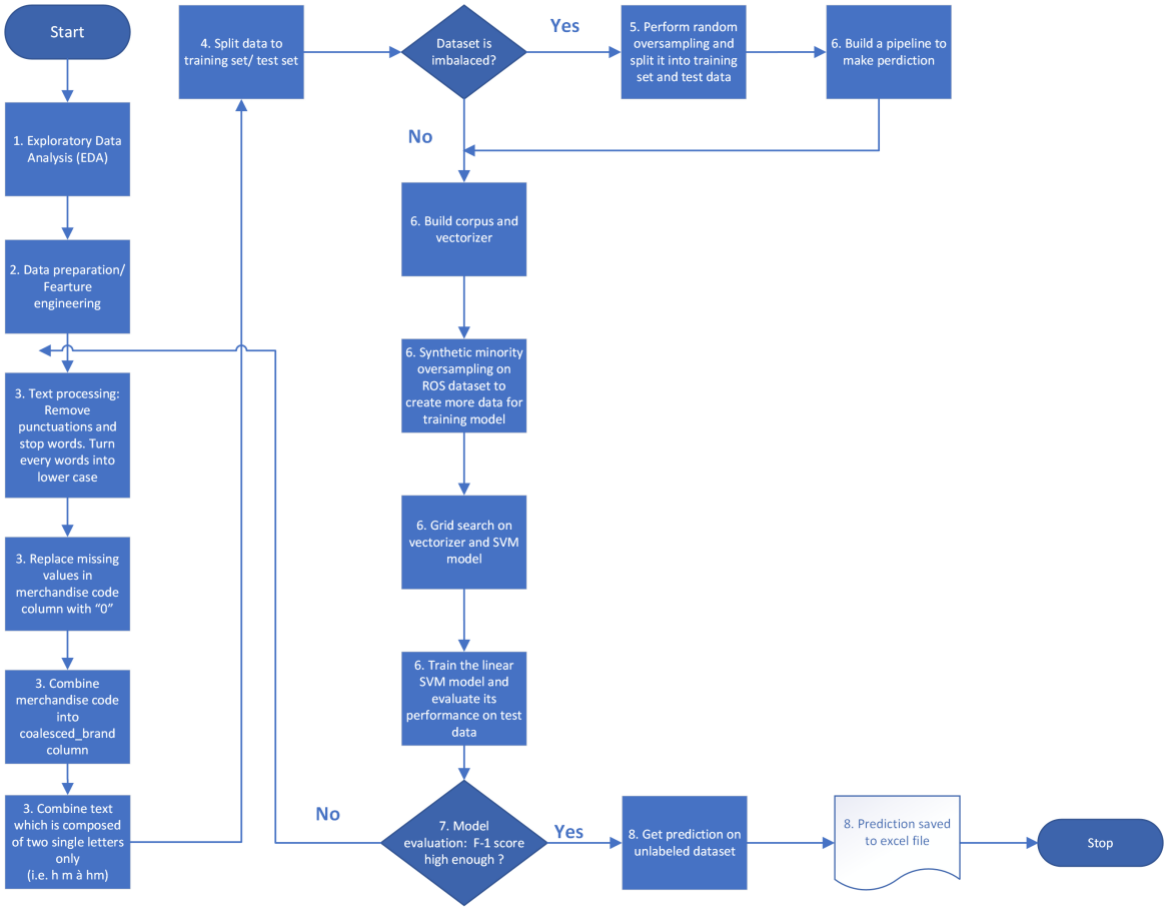
From the standpoint of Wells Fargo and its users, the main task is to satisfy and provided correct information for clients. This project will answer the question of what category is this transaction in and further provides the total amount customers spent on each category to device marketing strategy or promotion campaign.

I hope my work will inform Wells Fargo and its clients with their financial statements. Assist client with managing daily purchases and live a satisfying life with Wells Fargo as their life-long partner.

---

## 2. Method Overview

Analytic Process Flow-Diagram:




Steps used to analyze data and make predictions on each transaction by fitting machine learning model to cleaned training dataset.

1. Check missing values and explore data structure

2. Perform data preparation, feature engineering and text processing for text feature and numeric variable.

3. Explore the data by performing visualization on category distribution and word cloud for each category.

4. Apply over-sampling method to address imbalanced dataset issues.

5. Build a pipeline to prepare data before fed it to machine learning model. Use grid search to find the best hyperparameters then fit the best learn into machine learning model.

## 3. Exploratory Data Analysis (EDA)

- Visualize class distribution with bar chart
- Generate word cloud for each category after removing self-defined stop words.
    - Mask used in word cloud from: stock.adobe.com
    - Wells Fargo color from: schemecolor.com

## 4. Feature Engineering/ Over-Sampling

Removing features that are not helpful, regularize numeric feature to make the model more general for future used cases. The aim of feature engineering is to prevent the model from overfitting and enable the model to be as general as possible on future use cases.

- Regularize numeric variable

  Regularize numeric variable with StandardScaler. Instead of using RobustScaler or MinMaxScaler, StandardScaler performs the best for this dataset, by transforming mean to 0 and standard deviation to 1 across the amt column.

- Text processing

The final text processing method I used is NLTK. I used Genism at the beginning, assuming that it is the most useful NLP toolkit for structuring BOW but then it only got approximately 73% accuracy on Validation set. Therefore, I go with traditional NLTK on this project and turned out to be well.

I tried using Chi-Square score performing feature selection by filtering out unimportant words to build vectorizer. However, the result was worse than building vectorizer without using filtered vocabulary.

- Initialize the first vectorizer with trigram
- Set p-value threshold
- Build vectorizer with defined vocabulary

The code of Chi-Square score for reference as below:

```
# build vectorizer for BoW
vectorizer = feature_extraction.text.TfidfVectorizer(ngram_range = (1,3))
# get text feature to 1d array form
corpus = X_train_ros['coalesced_brand'].values
# fit texts into vectorizer and tranform them into tfidf form
X_train = vectorizer.fit_transform(corpus)
# get the test data that will be fed into model
X_test_corpus = X_test_ros['coalesced_brand']
# get a quick view on the vocabularies we just generated
dic_vocabulary = vectorizer.vocabulary_
# take only values of Category
X_names = vectorizer.get_feature_names()
p_value_limit = 0.95
dtf_features = pd.DataFrame()
for cat in np.unique(y_train_ros):
   # use tfidf class to calculate chi-square value
   name,p = chi2(X_train, y_train_ros==cat)

   dtf_features = dtf_features.append(pd.DataFrame(
            {"feature":X_names, "score":1-p, "Category":cat}
   ))
   dtf_features = dtf_features.sort_values(
             ["Category","score"],
             ascending=[True,False]
          )
   dtf_features = dtf_features[dtf_features["score"]> p_value_limit]
X_names = dtf_features["feature"].unique().tolist()
# build revised vectorizer
vectorizer = feature_extraction.text.TfidfVectorizer(vocabulary= X_names,
                                    stop_words= None,
                                    ngram_range= (1,3),
                                    max_df= 0.3, # very useful on imbalanced dataset
                                    min_df= 0
                                   )
```

```
##re-fit and transform corpus into revised tfidf class
vectorizer.fit(corpus)
# transform corpus into BoW
X_train = vectorizer.fit_transform(corpus)
```

- Try and Learn Process

    - Stop words: At first, I used the built-in stop words list from NLTk package, but the outcome didn't seem to work right on this dataset, so I decided to build customized stop words list. After trying many combinations of stop words inspecting from the top 10 text lists for each category, I decided to just go with "and" for stop word list.

    - Create a function to perform text cleaning with options to remove stop words, stemming, and lemmatizing the data.

    - After tried all kinds of combination of processing on text, the best way is not removing any stop word, inspect those misclassified transactions and modified the patterns of texts without lemmatization or stemming. This way will keep the brand names to their original nature forms without being transformed into something not even resemble them.

    - Combine category code to brand name (coalesced_brand) to get a more precise description for model to classify the transactions to correct category

- Over-sampling

Balance class distribution by randomly increasing minority class examples by replicating them. I first over sampled the data with ROS method, but the model performance didn't improve significantly. Then I try another over-sampling method – Synthetic Minority Over-Sampling. After adopting SMOTE in the model, the data size increased to 135,000 instances and the accuracy improved significantly.

    - By random over sampling minority classes transactions to overcome the bias that predictive model made on imbalanced data.

- Combine category code to brand name column (coalesced_brand)

- To achieve more precise predictions, combining category code to brand name will inform the model with more information besides brand name to classify transactions more precisely.

---

5. Grid Search

After performing grid search on model, the accuracy improved by 3%. Since this process will run for hours, I am pasting the grid search code executed here in abstract for reference and remove it from the final submission of the code.

The process of grid search in this project is as follows:
1. Perform grid search on model after achieving relatively high accuracy (around 85%)
2. Take the best hyperparameters generated from grid search
3. Inspect those misclassified transactions and fine tune text processing with the model adopted best hyperparameters from grid search

```
pipeline = Pipeline([
      ('vect', TfidfVectorizer(lowercase = True,
                        #vocabulary= X_names,
                         stop_words= lst_stopwords
                         )
         ),
        ('clf', SGDClassifier()
            )
])
parameters = {
   'vect__max_df': (0.4, 0.3, 0.2, 0.1, 0),
   'vect__min_df': (0,1),
   'vect__max_features': (None, 5000, 10000, 50000),
   'vect__ngram_range': ((1, 2), (1,3)),
   'clf__alpha': (0.00001, 0.000001),
   'clf__penalty': ('l2','l1'),
   'clf__max_iter': (100, 80,1000),
   'clf__tol': (0, 1e-1, 1e-2, 1e-3, 1e-5, 1e-6)
}
grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1, verbose=1, cv=5)
grid_search.fit(X_ROS['coalesced_brand'].values, y_ROS['Category'].values)
print(grid_search.best_params_)
print('*****')
print("Best score: %0.3f" % grid_search.best_score_) #0.731- 0.924- 0.95- 0.967
```

```
print('*****')
print("Best parameters set:")
best_parameters = grid_search.best_estimator_.get_params()
print('*****')
for param_name in sorted(parameters.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

## 6. Model Specification

I developed 5 models for predicting the category on each transaction, including logistic regression, random forest, XGboost and linear SVM.

After trying logistics regression model, ensemble method with random forest, and tried creating BOW with Genism package, linear SVM model with NLTK package combining grid search works the best, generating a model with 99.9% accuracy on test data!

Accuracy on unselected models:
Creating BOW with Genism and fed it into linear SVM model yielded 37% accuracy.
Both random forest model and logistic regression model yielded accuracy around 72%.

## 7. Validation and Model Evaluation

I first merged training set and test set and performed all feature engineering, data cleaning and text processing on full data available. Then split the data into its original training set and test set. With the original training data, I randomly split it into 20% for validation set and 80% training set. I trained the model first on the training subset only and evaluate the performance on validation set. Then train the model on whole dataset before making predictions on final submission on test set. Matrices used on model performance are precision, recall and F1 score. Since there is no apparently trade-off preference on recall or precision for this multi-class text classification project, adopting F1 score is optimal decision. Visualize the model performance by generating confusion matrix.

## 8. Conclusion

I have constructed a multi-classification model that yields 100% accuracy. The model will help Wells Fargo to successfully predict the category of transitions up to 100% category. Process found that will drastically increase accuracy besides fine tuning text processing in this project are:

- Over-sampling with SMOTE on imbalanced dataset
- Grid Search for best hyperparameters to fine tune model
- Adopt basic NLTK and Re package instead of Genism to generate vectorizer for multi-class text classification problems
- Build linear SVM model instead of using ensemble methods and logistic regression
- Instead of constructing stop words list, lemmatization or stemming to achieve high accuracy, searching for patterns in misclassified text is the best way to solve misclassification problem.