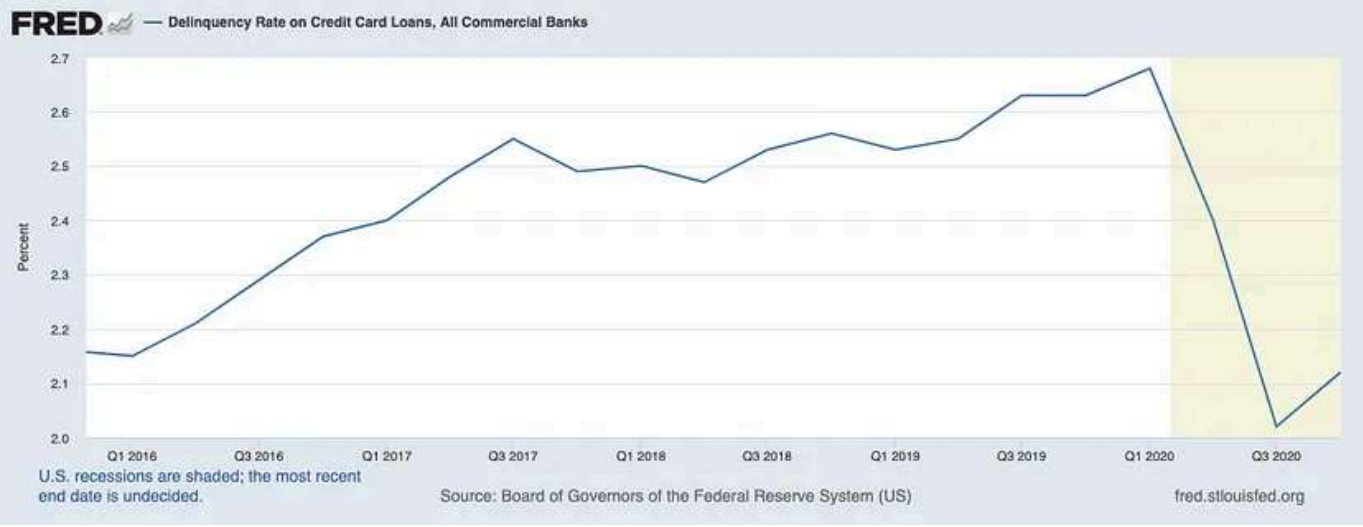
Predicting Credit Card Defaults with Machine Learning

Marcos Dominguez · Follow

Published in The Startup · 6 min read · Feb 26, 2021

According to Federal Reserve Economic Data, credit card delinquency rates have been increasing since 2016 (sharp decrease in Q1 2020 is due to COVID relief measures).



Delinquency Rate on Credit Card Loans, All Commercial Banks | FRED | St. Louis Fed

Units: Frequency: Suggested Citation: Board of Governors of the Federal Reserve System (US), Delinquency Rate on Credit...

fred.stlouisfed.org

The bank performs a charge-off on delinquent credit cards and eats the losses. If only there was a way to predict which customers had the highest probability of defaulting so it may be prevented...

Problem

(such as forbearance or debt consolidation, etc.). I will use various machine learning classification techniques to perform my analysis.

Data

Source, classic dataset from UC Irvine's machine learning repository:

https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

Target: Did the customer default? (Yes=1/Positive, No=0/Negative)

Features:

1. Credit Limit: Amount of the given credit (in dollars): it includes both the individual consumer credit and his/her family (supplementary) credit
2. Sex (1=male; 2=female)
3. Education (1=graduate school; 2=university; 3=high school; 4=other)

4. Marital Status (1=married; 2=single; 3=others)
5. Age (years)
6. History of past payment: The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above
7. Amount of bill statement (dollars) for past 6 months
8. Amount of previous payment for the past 6 months

Methodology

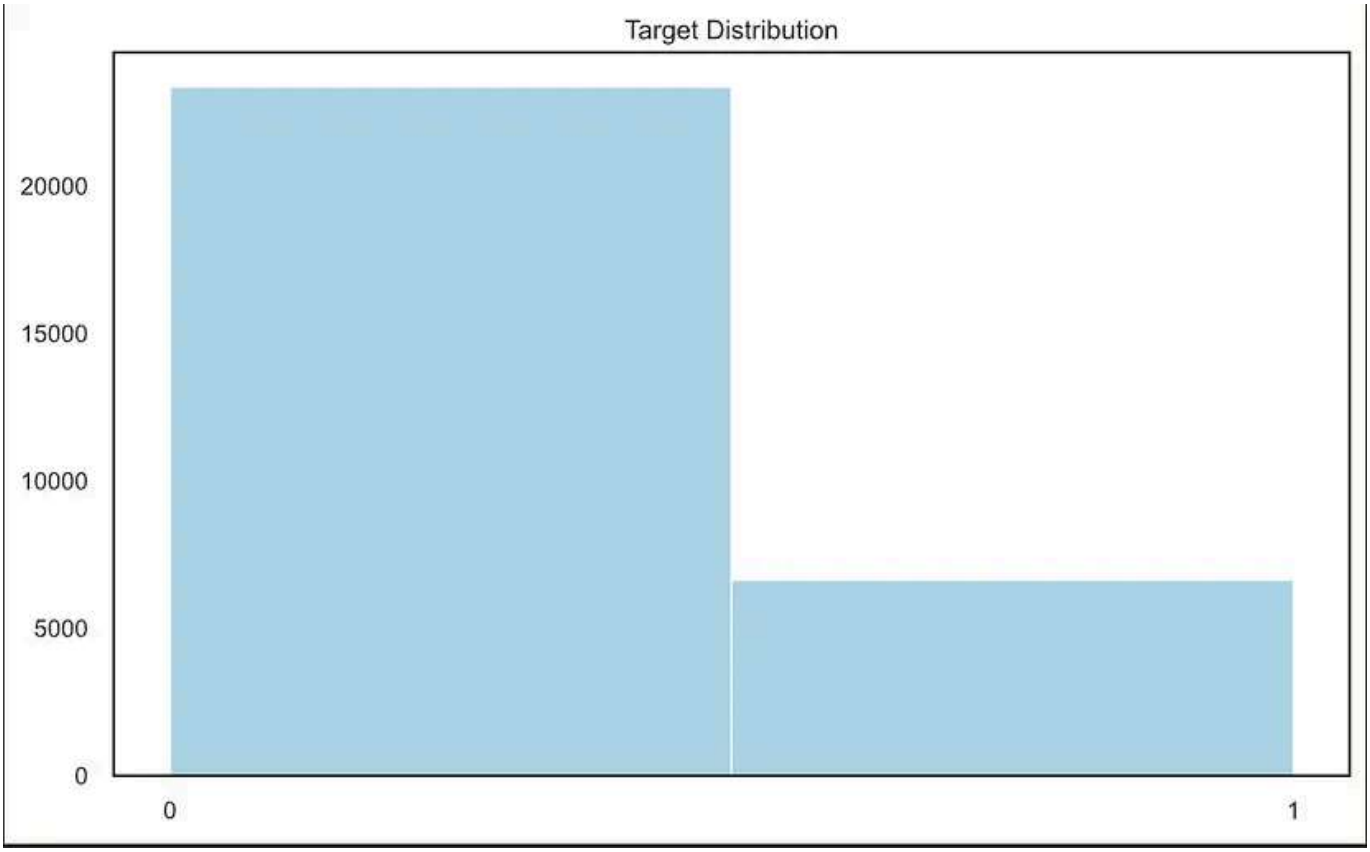
1. Exploratory Data Analysis
2. Baseline Model
3. Performance Metrics
4. Optimization
5. Feature Importance
6. Hyperparameter Tuning
7. Class Imbalance
8. Analyze Results

Exploratory Data Analysis

The following are some key findings in the data. Shout to Gabriel Preda from Kaggle for the awesome visualization ideas. Check out his work here:

https://www.kaggle.com/gpreda

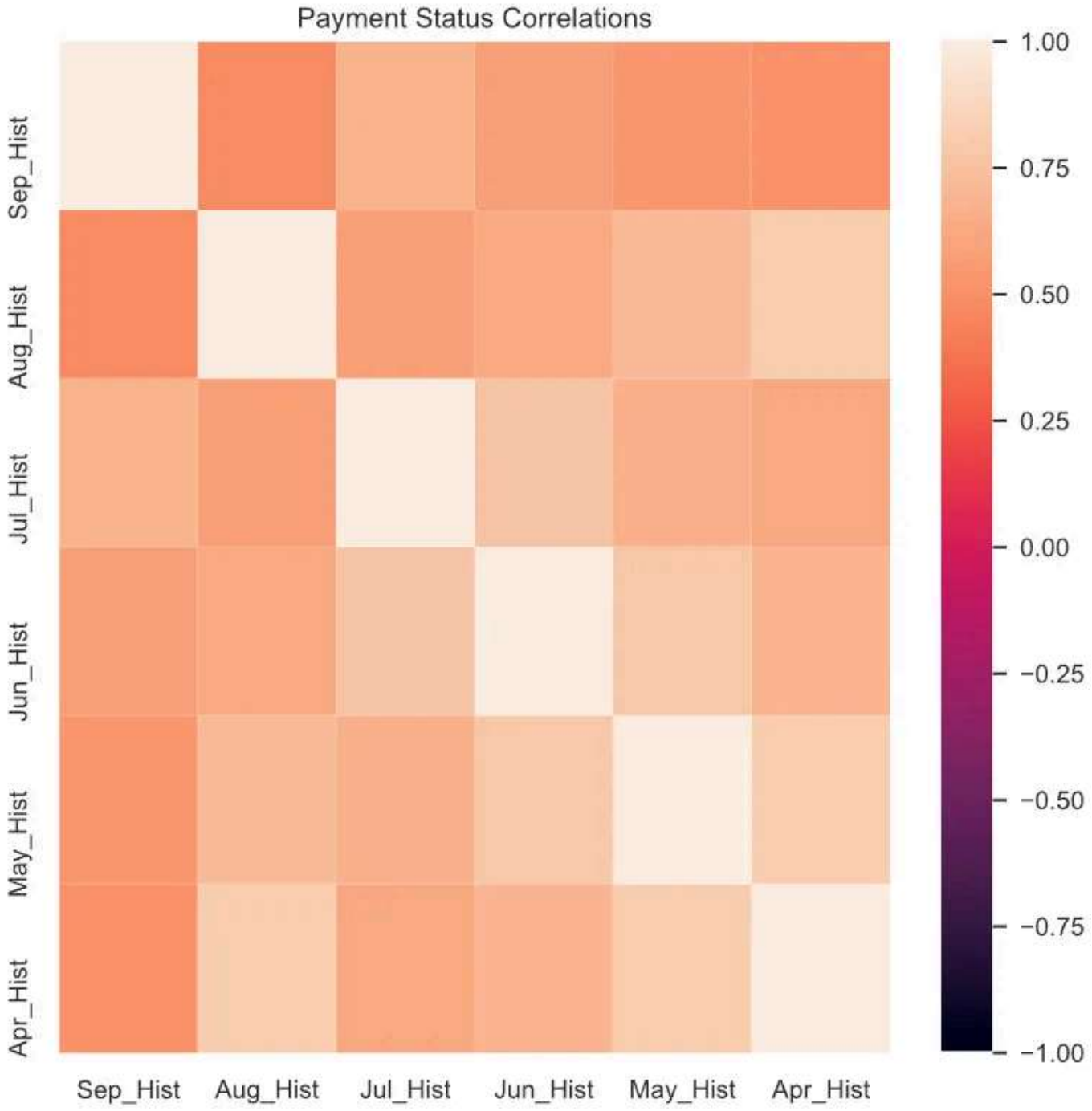
Distribution of target classes is highly imbalanced, non-defaults far outnumber defaults. This is common in these datasets since most people pay credit cards on time (assuming there isn't an economic crisis).



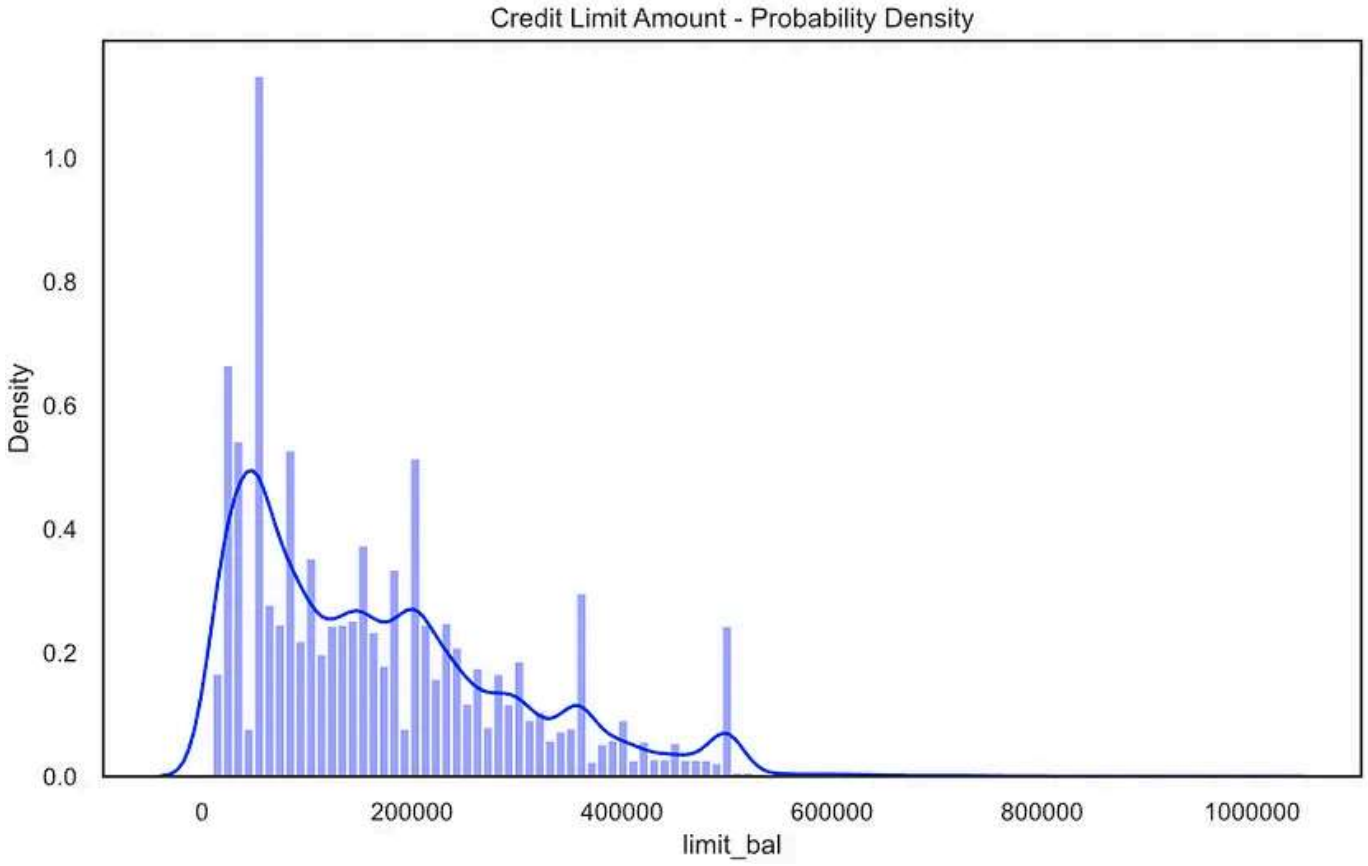
0=No Default, 1=Default

Payment status. Correlation strength increases the closer the months are in time. Makes sense. For example, one could assume a late payment in August

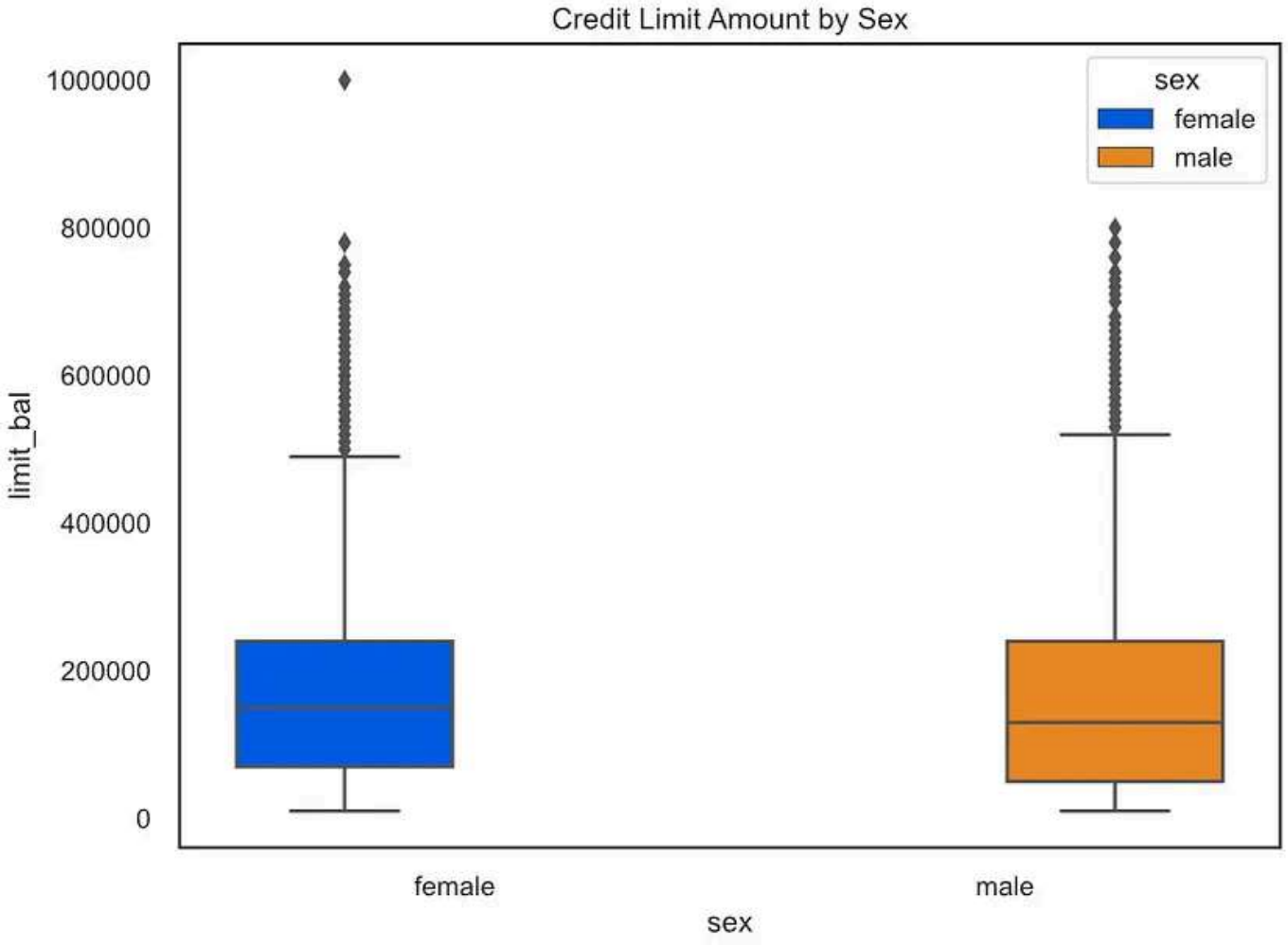
would likely lead to a late payment in September. However, it is less clear we can make the same assumption for April and September



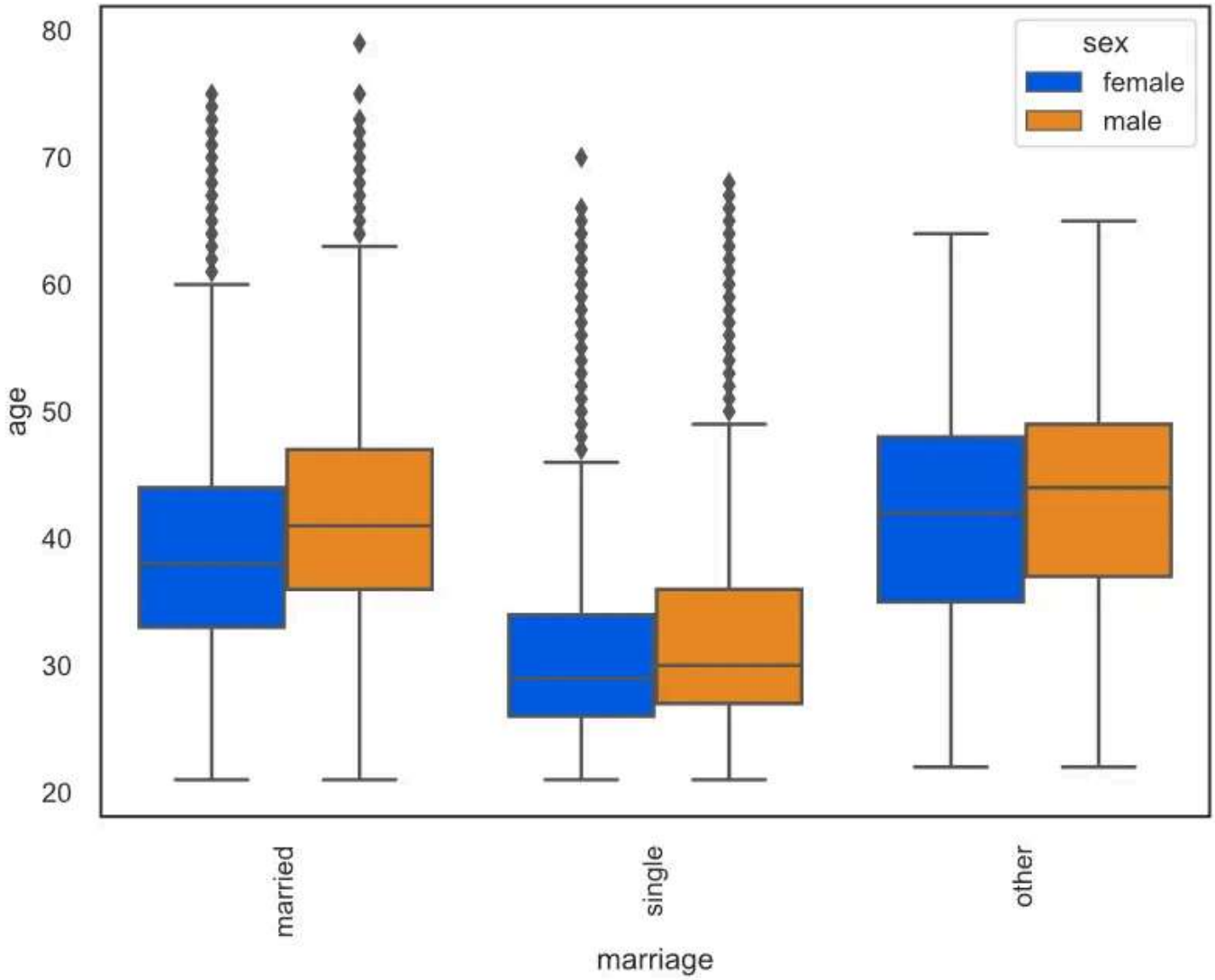
Distribution of credit limit amounts. The three largest credit limit amount groups are $50k, $20k, and $30k, respectively.



Credit Limit by Sex. The data is evenly distributed amongst males and females.



Marriage, age, and sex. The dataset mostly contains couples in their mid-30s to mid-40s and single people in their mid-20s to early-30s.



Baseline Model

Now comes the fun part. Let's start the prediction process by establishing a baseline model that we can build on.

Prepare features and target:

```python
# Establish features and target variables
X = df.loc[:,'limit_bal':'Apr_Pmt']
y = df['Default']

# Encode categorical variables
categoricals = list(X.select_dtypes('object').columns)
numericals = list(X.select_dtypes('int64').columns)

def encode_cats(categoricals, numericals):
    """
    Takes in a list of categorical columns and a list of numerical columns and returns the datafr
    """
    ohe = OneHotEncoder(sparse=False, drop='first')
    cat_matrix = ohe.fit_transform(X.loc[:, categoricals])
    X_ohe = pd.DataFrame(cat_matrix,
                         columns=ohe.get_feature_names(categoricals), #create meaningful column r
                         index=X.index) #keep the same index values

    X = encode_cats(categoricals, numericals)

    return pd.concat([X.loc[:, numericals], X_ohe], axis=1)
```

encode_cats.py hosted with ❤ by GitHub view raw

Scale the data so the model can easily digest information.

```python
def scale_data(X_train, X_val, X_test):
    """
    Input: Features (numpy arrays)
    Output: Scaled data
    """
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)
    X_test_scaled = scaler.transform(X_test)

    return X_train_scaled, X_val_scaled, X_test_scaled

 X_train_scaled, X_val_scaled, X_test_scaled = scale_data(X_train, X_val, X_test)
```

scale_data.py hosted with ❤ by GitHub view raw

Score several models and choose one to improve upon

```python
def model_score(model_name, model, X_train_scaled, X_val_scaled,
                X_test_scaled, y_train, y_val, y_test, test=False):
    """
    Input: Transformed feature and target sets
    Output: Validation scores. If test=True, includes test scores
    """
    print('Calculating validation score...')

    my_model = model
    my_model.fit(X_train_scaled,y_train)
    print(f'{model_name} accuracy score: {my_model.score(X_val_scaled,y_val):.4}\n')

    if test:
        print("Calculating test score...")
        print(f'{model_name} accuracy score: {my_model.score(X_test_scaled,y_test):.4}\n')

    return my_model

knn = model_score('KNN', KNeighborsClassifier(n_neighbors=5),
                      X_train_scaled,X_val_scaled, X_test_scaled,
                      y_train, y_val, y_test,test=False)

lr = model_score('LogReg', LogisticRegression(penalty='none'),
                      X_train_scaled,X_val_scaled, X_test_scaled,
                      y_train, y_val, y_test,test=False)

rf = model_score('RF', RandomForestClassifier(),
                      X_train_scaled,X_val_scaled, X_test_scaled,
                      y_train, y_val, y_test,test=False)

gbm = model_score('XGBoost', xgb.XGBClassifier(),
                      X_train_scaled,X_val_scaled, X_test_scaled,
                      y_train, y_val, y_test,test=False)

svc = model_score('SVM', SVC(probability=True),
                      X_train_scaled,X_val_scaled, X_test_scaled,
                      y_train, y_val, y_test,test=False)
```

model_score.py hosted with ❤ by GitHub view raw

```
Calculating validation score...
KNN accuracy score: 0.7938

Calculating validation score...
LogReg accuracy score: 0.8158

```

```
Calculating validation score...
RF accuracy score: 0.8168

Calculating validation score...
XGBoost accuracy score: 0.8192

Calculating validation score...
SVM accuracy score: 0.8193
```

results.txt hosted with ❤ by GitHub view raw

Support Vector Machine (SVM) performs the best with an accuracy of 0.8193. However, we will move forward with Random Forest because it is just as robust plus it's less computationally expensive.

Optimization

Before moving onto performance metrics, let's discuss optimization. What metric exactly are we optimizing? In this case, we are optimizing recall. Recall is a performance metric which attempts to answer the question: What proportion of actual positives was identified correctly? Mathematically, the formula is:

$$RecallScore = \frac{TP}{TP+FN}$$

TP = True Positive, or a correctly predicted default

FN = False Negative, or an incorrectly predicted non-default

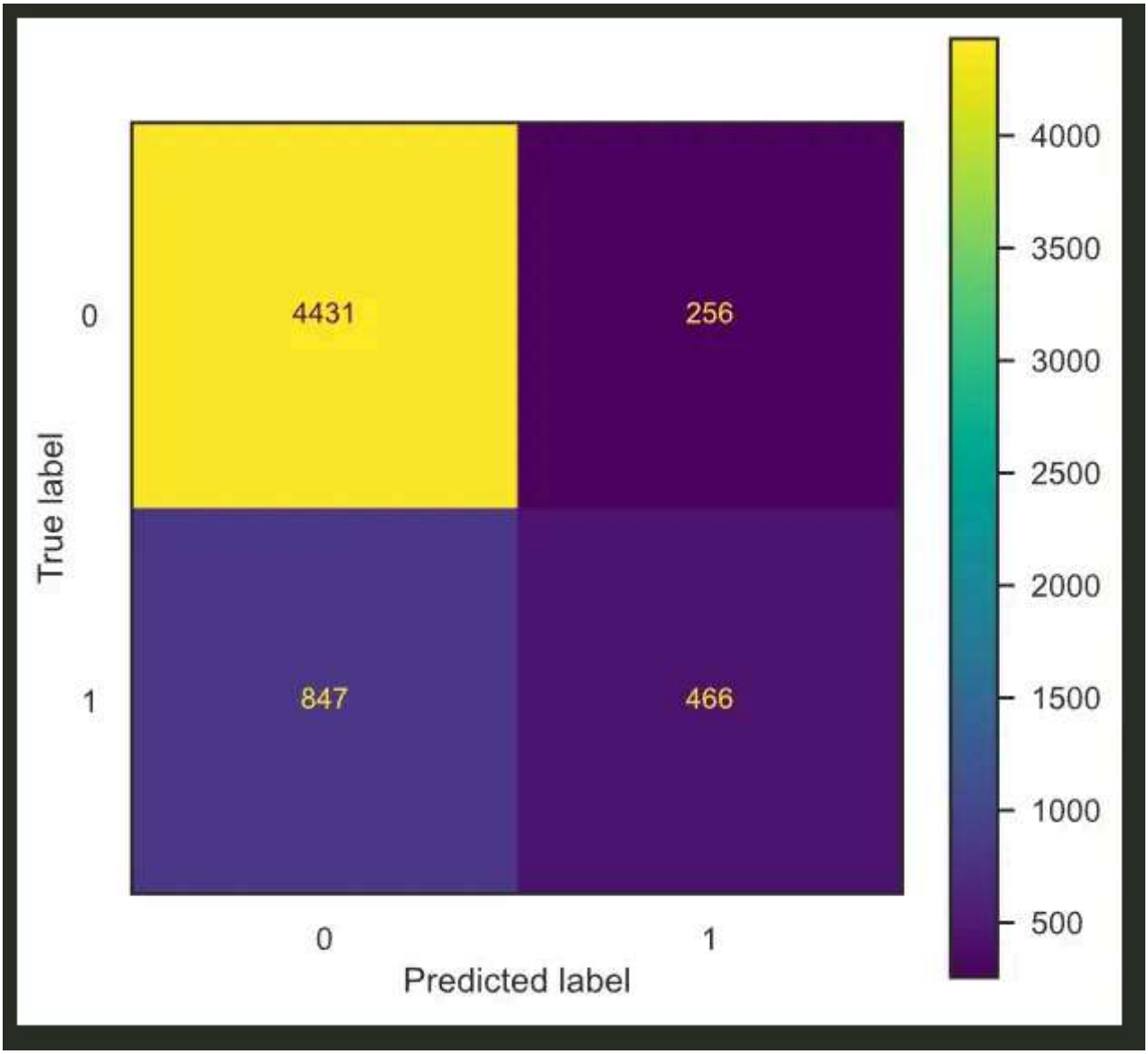
Ideally, we do not want to allow any defaults to fall through the cracks, so our optimal model will minimize False Negatives (So RecallScore is as high as possible).

For a complete discussion of Precision vs Recall scores, check out this article created by the awesome developers at Google:

https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

Performance Metrics

Confusion Matrix



The number we want to minimize is in the bottom-left quadrant. This number represents those customers we predicted would NOT default, but they in fact DID default. We need this number to be as small as possible.

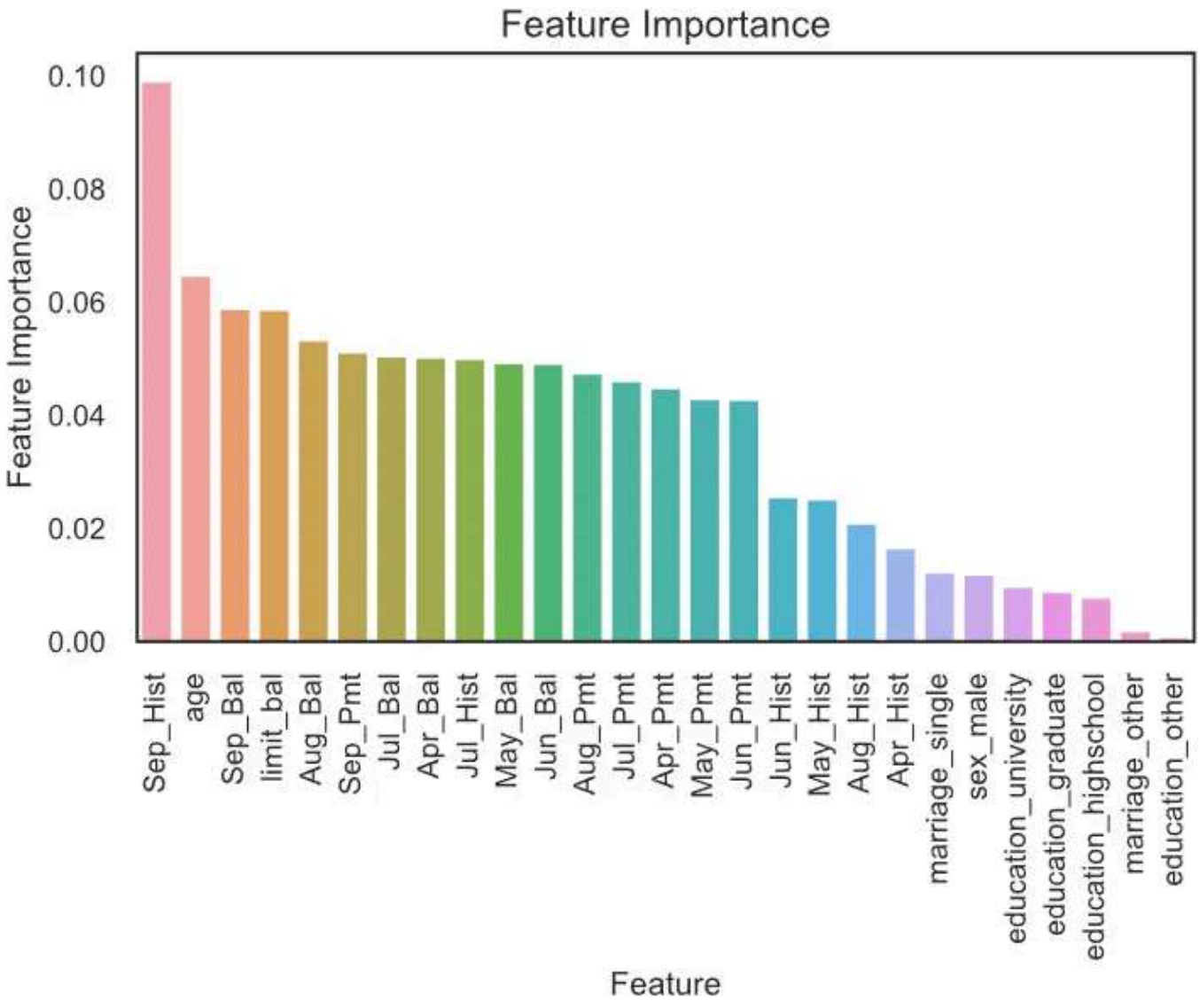
	precision	recall	f1-score	support
0	0.84	0.95	0.89	4687
1	0.65	0.35	0.46	1313
accuracy			0.82	6000
macro avg	0.74	0.65	0.67	6000
weighted avg	0.80	0.82	0.79	6000

Recall of 0.95 is very good. So we're off to a great start. Let's see what improvements we can make.

Feature Selection

There are many feature selection scores one could perform to determine which features are most useful. For this case, we will use Feature Importance.

Generally speaking, Feature Importance is the process of assigning scores to each feature, ranking its usefulness in predicting the target variable.



The top features are shown in the chart above. It's interesting how 'age' is the second most important feature. Let's keep all of them except the last 6, the categorical variables. If we remove those features which are least important and keep the most important ones, this might allow us to better predict our target variable.

```
1   # Update dataframe
2   X = X.iloc[:, np.argsort(rf.feature_importances_,)[7:]]
```

feature_sel.py hosted with ❤ by GitHub view raw

Hyperparameter Tuning

Let’s search for optimal parameters for our model

```python
# Define parameter values that should be searched
bootstrap = [True, False]
max_features = ['auto','sqrt']
min_samples_leaf = [1,2,4]
min_samples_split = [2,5,10]
n_estimators = [50,100,150,200]
max_depth = [4,6,10,12, None]

# Specify "parameter distributions" rather than a "parameter grid"
param_dist = dict(n_estimators=n_estimators, max_depth = max_depth, bootstrap=bootstrap,
                  min_samples_leaf=min_samples_leaf,min_samples_split=min_samples_split,
                  max_features=max_features)

# n_iter controls the number of searches

rand = RandomizedSearchCV(RandomForestClassifier(), param_dist, cv=10, scoring='accuracy',
                          n_iter=10, random_state=42)
rand.fit(X_train_scaled, y_train)

# Uncomment line below to print
# rand.cv_results_

def get_best_model(model_grid):
    best_k = model_grid.cv_results_['rank_test_score'][0]
    print (f"Mean score: {model_grid.cv_results_['mean_test_score'][best_k]}",
           f"Std: {model_grid.cv_results_['std_test_score'][best_k]}")

# Examine the best model
print(rand.best_score_)
print(rand.best_params_)
get_best_model(rand)
```

hyperparameter_tuning.py hosted with ❤ by GitHub view raw

```
score: 0.8209

best_params: {'n_estimators': 100,
              'min_samples_split': 10,
              'min_samples_leaf': 2,
              'max_features': 'auto',
              'max_depth': 6,
              'bootstrap': False}

Mean score: 0.8209 Std: 0.0081
```

output.txt hosted with ❤ by GitHub view raw

Class Imbalance

In the Exploratory Data Analysis section, we established that the target is very imbalanced. “Imbalanced classifications pose a challenge for predictive modeling as most of the machine learning algorithms used for classification were designed around the assumption of an equal number of examples for each class” (Jason Brownlee, Machine Learning Mastery).

There are many ways we can remedy this. For my analysis, I will use random undersampling and oversampling. Random undersampling essentially deletes data from the negative class so the target distribution is even. Random oversampling duplicates information from the positive class so the target distribution is even. We will try both of these and see which one is more effective.

```python
def over_under_sample(X_train, y_train, Under=True, Over=True):
    """
    Input: training features and target
    Output: under/oversampled datasets
    """
    rus = RandomUnderSampler(random_state=42)
    ros = RandomOverSampler(random_state=42)

    if Under and Over:
        X_train_under, y_train_under = rus.fit_sample(X_train, y_train)
        X_train_over, y_train_over = ros.fit_sample(X_train, y_train)
        return X_train_under, y_train_under, X_train_over, y_train_over
    elif Under:
        X_train_under, y_train_under = rus.fit_sample(X_train, y_train)
        return X_train_under, y_train_under
    else:
        X_train_over, y_train_over = ros.fit_sample(X_train, y_train)

# Split data
X_train, X_val, X_test, y_train, y_val, y_test = train_test_val_split(X, y)

# Random under/over sampling
X_train_under, y_train_under, X_train_over, y_train_over = over_under_sample(X_train, y_train, U
```

over_under.py hosted with ❤ by GitHub view raw

Let's see how these models perform

Random Undersampling

```python
# Scale
X_train_scaled_under, X_val_scaled, X_test_scaled = scale_data(X_train_under, X_val, X_test)

# Score
rf_under = model_score('RF', RandomForestClassifier(**rand.best_params_),
                X_train_scaled_under, X_val_scaled, X_test_scaled,
                y_train_under, y_val, y_test, test=False)
```

under_score.py hosted with ❤ by GitHub view raw

RF recall score: 0.79

Random Oversampling

```
# Split data
X_train, X_val, X_test, y_train, y_val, y_test = train_test_val_split(X, y)

# Scale
X_train_scaled_under, X_val_scaled, X_test_scaled = scale_data(X_train_under, X_val, X_test)

# Score
rf_under = model_score('RF', RandomForestClassifier(**rand.best_params_),
                  X_train_scaled_under, X_val_scaled, X_test_scaled,
                  y_train_under, y_val, y_test, test=False)
```

over_score.py hosted with ❤ by GitHub view raw

RF recall score: 0.79

Both recall scores are 0.79. Recall score went down nearly 0.16 from our baseline model. Interesting...

Analyze Results

Sometimes the best model is the simplest. The model with minimal manipulation yielded the highest recall score of 0.95. After feature selection and hyperparameter tuning, recall decreased to 0.79.

Let's check for overfitting. Overfitting means the model is strong at predicting the data on which it was trained, but weak at generalizing to unseen data. Lets test the model on never-before-seen data points and see how it performs

```
# features and target
X = df.loc[:,'limit_bal':'Apr_Pmt']
y = df['Default']

# encode categorical
categoricals = list(X.select_dtypes('object').columns)
numericals = list(X.select_dtypes('int64').columns)

def encode_cats(categoricals, numericals):
    """
    Takes in a list of categorical columns and a list of numerical columns and returns the datafr
    """
    ohe = OneHotEncoder(sparse=False, drop='first')
    cat_matrix = ohe.fit_transform(X.loc[:, categoricals])
    X_ohe = pd.DataFrame(cat_matrix,
                         columns=ohe.get_feature_names(categoricals), #create meaningful column r
                         index=X.index) #keep the same index values

    return pd.concat([X.loc[:, numericals], X_ohe], axis=1)

X = encode_cats(categoricals, numericals)

# Split data
X_train, X_val, X_test, y_train, y_val, y_test = train_test_val_split(X, y)

# Scale
X_train_scaled, X_val_scaled, X_test_scaled = scale_data(X_train, X_val, X_test)

# Model score
rf = model_score('RF', RandomForestClassifier(**rand.best_params_),
                 X_train_scaled, X_val_scaled, X_test_scaled,
                 y_train, y_val, y_test, test=True)
```

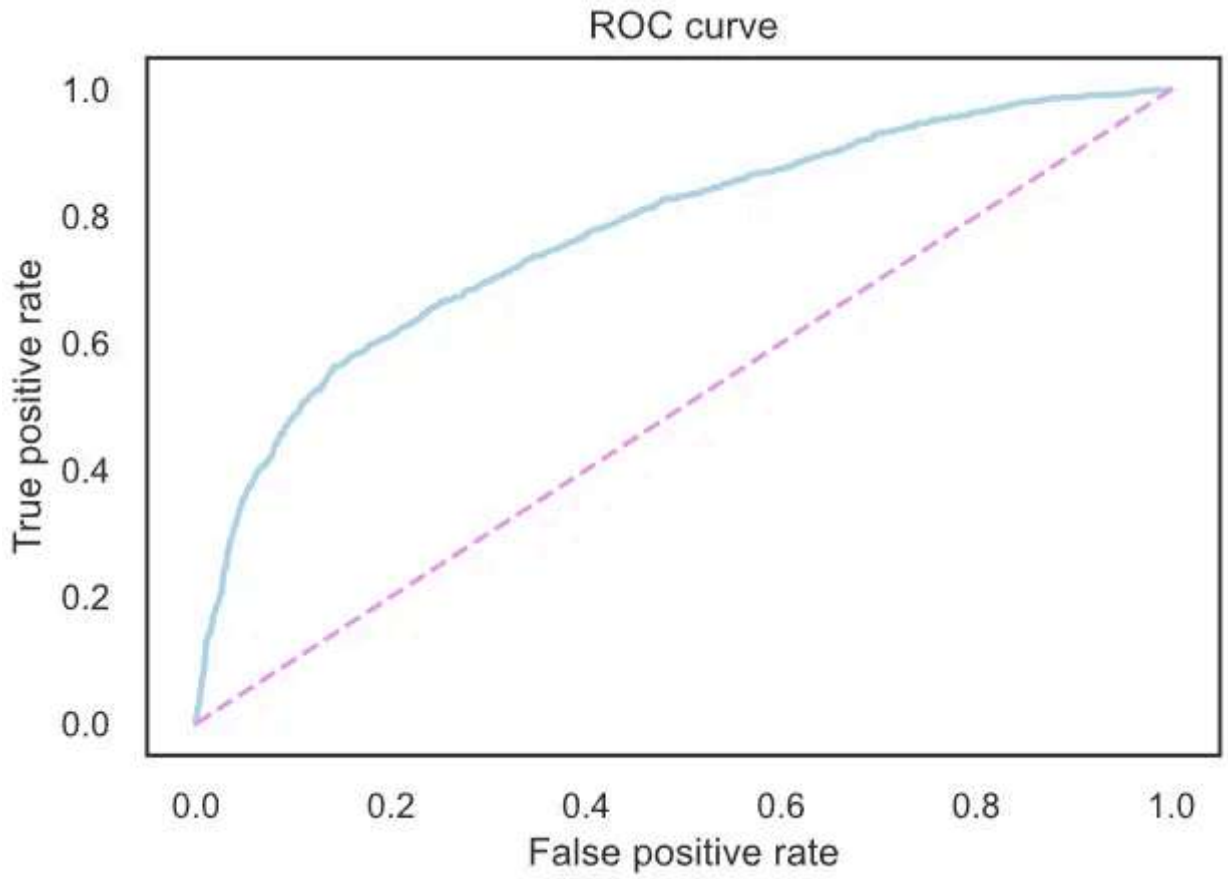
test.py hosted with ❤ by GitHub view raw

RF validation score: 0.8208, test score: 0.8217

The validation score is similar to the test score, so we know it's performing similarly on completely unseen. Therefore, we can conclude there is no overfitting going on.

ROC Curve

The area under the ROC curve tells us how well the model separates the different classes in the dataset. It plots true positive rate against false positive rate



Area under ROC = 0.7747

We're calculating the area between the blue curved line and pink dotted line. This area is a number between 0 and 1, zero meaning the model predicted all of the data incorrectly, and one meaning the model predicted all of the data correctly. Our model is pretty good at 0.7747.

At the end of the day, having the ability to predict 95% (recall score) of potential defaults would save a-lot of money on credit card charge-offs.

Obviously, real-world application is more nuanced, but this modeling process is a step in the right direction.

For full code on this project and others, please visit my Github Repository

mdominguez2010/Classification_Credit_Cards

Github

Social Media

Marcos Dominguez - San Francisco, California, United States | Professional Profile | LinkedIn

I am a Data Scientist with a background in banking and lending

www.link

Contact me

email: md.ghsd@gmail.com

Machine Learning | Data Science | Data Visualization | Credit Cards

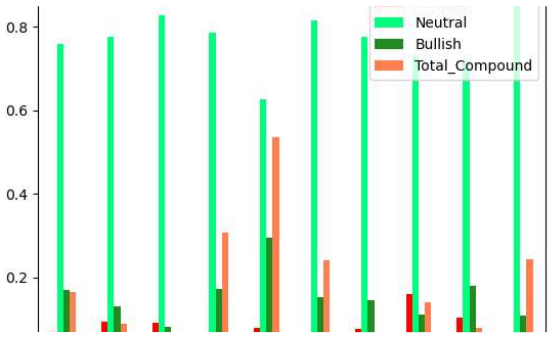
Written by Marcos Dominguez

Follow

84 Followers · Writer for The Startup

Data Scientist with a background in banking and finance. I love statistics, programming, and machine learning.

More from Marcos Dominguez and The Startup



Marcos Dominguez in Nerd For Tech

Wallstreetbets Sentiment Analysis on Stock Prices using Natural Language...

The following is the first part in an ongoing project to analyze sentiment on the famous...

5 min read · Jun 9, 2021

62

Michael Lim in The Startup

What Sam Altman's Prediction About The $1B One-Person Business Mod...

If you're under 39, you've got a massive advantage.

· 5 min read · Feb 25, 2024

4.6K 124

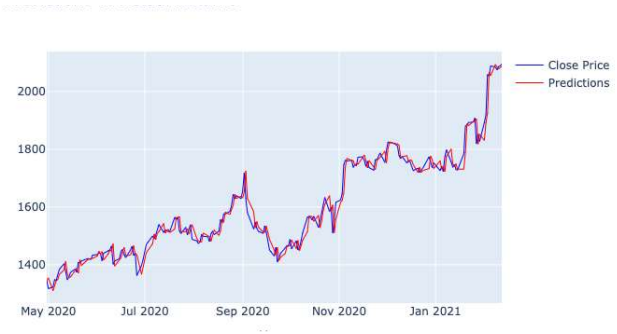
Kurtis Pykes in The Startup

7 Habits That Are Incredibly Hard to Do But Pay Off Forever

Anything Worthwhile Requires a Great Deal of Suffering

✦ · 6 min read · Feb 29, 2024

4.3K 76



Marcos Dominguez in Analytics Vidhya

Forecasting Google's Stock Price with ARIMA Modeling

After the Gamestop fiasco with the subreddit r/wallstreetbets, I became very intrigued wit...

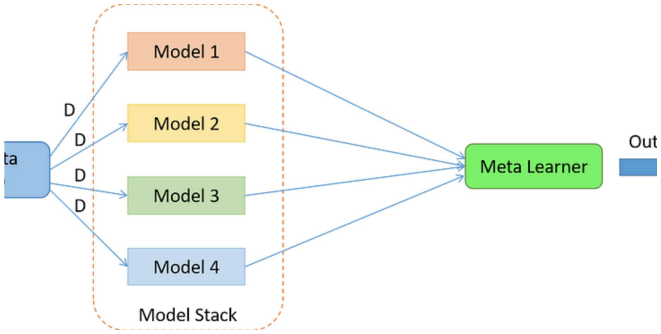
5 min read · Feb 16, 2021

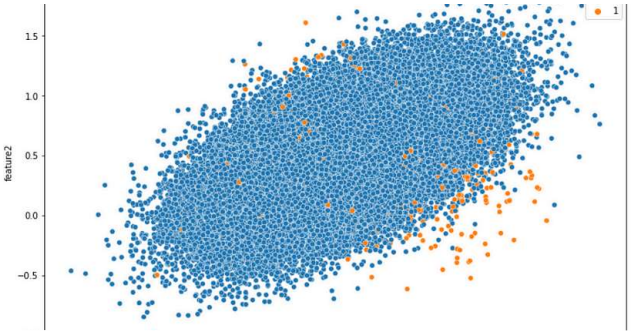
5 1

See all from Marcos Dominguez

See all from The Startup

Recommended from Medium





Ajay Verma in Artificial Intelligence in Plain English

Mastering Complexity: The Comprehensive Guide to Stacking...

In the realm of ensemble learning, stacking emerges as a sophisticated maestro...

7 min read · Nov 28, 2023

1

Manjinder Singh

Sampling Methods For Imbalanced Dataset Classification along with...

In machine learning, dealing with imbalanced data is crucial. This occurs when there are...

8 min read · Nov 16, 2023

27

Lists

Predictive Modeling w/ Python

20 stories · 1053 saves

Practical Guides to Machine Learning

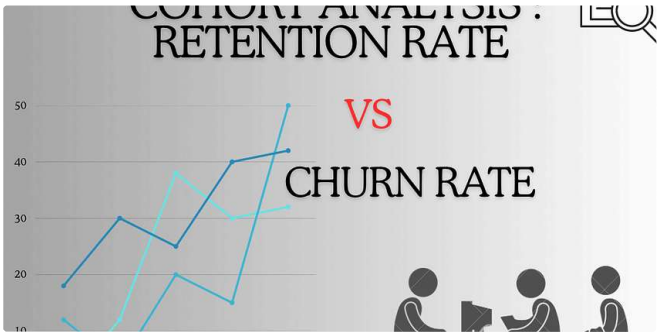
10 stories · 1260 saves

Natural Language Processing

1340 stories · 822 saves

data science and AI

40 stories · 121 saves

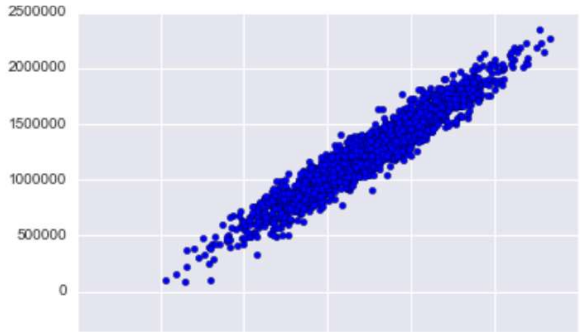


Hafez Afghan

Cohort Analysis in SQL: Retention Rate vs Churn Rate

The algorithm and interpretation are clearly explained!

10 min read · Feb 17, 2024



Simran Kaushik

House Price Prediction: A Simple Guide with Scikit-Learn and Linear...

Navigate the realm of predictive analytics with simplicity

7 min read · Nov 14, 2023

85 69

Unicorn Day

Understanding Decision Trees with the Iris Dataset

Introduction

3 min read · Oct 11, 2023

Muhammad Arief Rachman

Credit Scoring Prediction

Scorecard modeling

15 min read · Nov 14, 2023

21 1

See more recommendations