Ensemble Learning

Initial Imports

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import numpy as np
import pandas as pd
from pathlib import Path
from collections import Counter
```

In [3]:
```
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import confusion_matrix
from imblearn.metrics import classification_report_imbalanced
```

Read the CSV and Perform Basic Data Cleaning

In [4]:
```
# Load the data
#file_path = Path('Resources/LoanStats_2019Q1.csv')
file_path="C:/Users/CS_Knit_tinK_SC/Documents/GitHub/HW_8_ML_Conf_Imb_Inputs_U
df = pd.read_csv(file_path)

# Preview the data
df.head()
```

Out[4]:

	loan_amnt	int_rate	installment	home_ownership	annual_inc	verification_status	issue_d	loan_s
0	10500.0	0.1719	375.35	RENT	66000.0	Source Verified	Mar-2019	lo
1	25000.0	0.2000	929.09	MORTGAGE	105000.0	Verified	Mar-2019	lo
2	20000.0	0.2000	529.88	MORTGAGE	56000.0	Verified	Mar-2019	lo
3	10000.0	0.1640	353.55	RENT	92000.0	Verified	Mar-2019	lo
4	22000.0	0.1474	520.39	MORTGAGE	52000.0	Not Verified	Mar-2019	lo

5 rows × 86 columns

Split the Data into Training and Testing

In [5]:
```
# Create our features
X = df.drop(columns=["loan_status", "issue_d", "pymnt_plan", "initial_list_sta

# Create our target
y = df["loan_status"]
```

In [6]:
```
X.describe()
```

Out[6]:

	loan_amnt	int_rate	installment	annual_inc	dti	delinq_2yrs	inq_last_
count	68817.000000	68817.000000	68817.000000	6.881700e+04	68817.000000	68817.000000	68817.0
mean	16677.594562	0.127718	480.652863	8.821371e+04	21.778153	0.217766	0.4
std	10277.348590	0.048130	288.062432	1.155800e+05	20.199244	0.718367	0.7
min	1000.000000	0.060000	30.890000	4.000000e+01	0.000000	0.000000	0.0
25%	9000.000000	0.088100	265.730000	5.000000e+04	13.890000	0.000000	0.0
50%	15000.000000	0.118000	404.560000	7.300000e+04	19.760000	0.000000	0.0
75%	24000.000000	0.155700	648.100000	1.040000e+05	26.660000	0.000000	1.0
max	40000.000000	0.308400	1676.230000	8.797500e+06	999.000000	18.000000	5.0

8 rows × 76 columns

In [7]:
```
# Check the balance of our target values
y.head()
y.value_counts()
```

Out[7]:
```
low_risk     68470
high_risk      347
Name: loan_status, dtype: int64
```

In [8]:
```
# Create X_train, X_test, y_train, y_test
from sklearn.model_selection import train_test_split
# from library.module import class

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    # Controls the shuffling
                                                    # Pass an int for reprodu
                                                    random_state=1,
                                                    # If not None, data is sp
                                                    # https://www.scribbr.com
                                                    stratify=y)
```

Data Pre-Processing

Scale the training and testing data using the `StandardScaler` from `sklearn`. Remember that when scaling the data, you only scale the features data (`X_train` and `X_testing`).

In [9]:
```
# Create the StandardScaler instance to normalize the values individually, be
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
new_X_train=pd.get_dummies(X_train, columns=["home_ownership", "verification_
new_X_test=pd.get_dummies(X_test, columns=["home_ownership", "verification_sta
scaler.fit(new_X_train)
print(f'the scaler mean is {scaler.mean_}')
```

```
the scaler mean is [1.66578136e+04 1.27660017e-01 4.80095682e+02 8.77065606e+0
4
 2.18680284e+01 2.14872510e-01 4.96279935e-01 1.25791095e+01
 1.26404712e-01 1.76322217e+04 2.46571534e+01 1.59988270e+04
 1.59958149e+04 9.70309271e+02 9.70092896e+02 6.58986630e+02
 3.11290121e+02 3.25573510e-02 0.00000000e+00 0.00000000e+00
 5.16651802e+02 1.61009068e-02 1.00000000e+00 0.00000000e+00
 1.84220123e+02 1.64013074e+05 1.01720530e+00 3.29072696e+00
 8.45055413e-01 1.90945904e+00 1.43137449e+01 4.34276057e+04
 6.92161319e+01 1.23519724e+00 2.61613578e+00 6.31146993e+03
 5.57263815e+01 4.24598690e+04 1.38434860e+00 1.69228086e+00
 2.19421840e+00 4.82040998e+00 1.41980994e+04 1.69935858e+04
 4.82648996e+01 6.76199333e-03 2.43218631e+00 1.25529567e+02
 1.75546888e+02 1.43142680e+01 7.20092227e+00 1.44708595e+00
 2.47727660e+01 7.69094397e+00 4.65356894e-01 3.80268155e+00
 5.63680927e+00 5.13572425e+00 7.40938154e+00 9.50621948e+00
 8.62183213e+00 1.34986050e+01 5.64169185e+00 1.25636674e+01
 0.00000000e+00 0.00000000e+00 5.06665117e-02 2.21551190e+00
 9.50818666e+01 3.06032531e+01 1.26365961e-01 0.00000000e+00
 2.10605250e+05 6.15583020e+04 2.97737617e+04 5.58864324e+04
 9.43579013e-03 5.29547392e-01 1.05711850e-01 3.55304968e-01
 4.76846470e-01 3.76365961e-01 1.46787569e-01 1.00000000e+00
 1.00000000e+00 8.59102534e-01 1.40897466e-01]
```

In [10]:
```
# Fit the Standard Scaler with the training data
# When fitting scaling functions, only train on the training dataset
data_scaler = StandardScaler()

data_scaler.fit_transform(new_X_train)
```

Out[10]:
```
array([[-0.64978133, -0.30327574, -0.52669276, ...,  0.        ,
         0.40497582, -0.40497582],
       [ 0.42378352, -0.42616892,  0.71269346, ...,  0.        ,
         0.40497582, -0.40497582],
       [ 1.79013878, -0.82400957,  2.19335392, ...,  0.        ,
         0.40497582, -0.40497582],
       ...,
       [ 0.81417074, -0.36368086,  1.17925683, ...,  0.        ,
         0.40497582, -0.40497582],
       [-0.25939411, -0.42616892, -0.08220905, ...,  0.        ,
         0.40497582, -0.40497582],
       [-0.16179731, -1.196855  , -0.05849257, ...,  0.        ,
         0.40497582, -0.40497582]])
```

In [11]:
```
# Scale the training and testing data
X_train_scaled = data_scaler.transform(new_X_train)
X_test_scaled = data_scaler.transform(new_X_test)
```

Ensemble Learners

In this section, you will compare two ensemble algorithms to determine which algorithm results in the best performance. You will train a Balanced Random Forest Classifier and an Easy Ensemble classifier . For each algorithm, be sure to complete the folliowing steps:

1. Train the model using the training data.
2. Calculate the balanced accuracy score from sklearn.metrics.
3. Display the confusion matrix from sklearn.metrics.
4. Generate a classication report using the `imbalanced_classification_report` from imbalanced-learn.
5. For the Balanced Random Forest Classifier only, print the feature importance sorted in descending order (most important feature to least important) along with the feature score

Note: Use a random state of 1 for each algorithm to ensure consistency between tests

Balanced Random Forest Classifier

In [38]:
```
# Resample the training data with the BalancedRandomForestClassifier
from numpy import mean
from imblearn.ensemble import BalancedRandomForestClassifier
brf = BalancedRandomForestClassifier(n_estimators=1000, random_state=1)
print(brf.fit(X_train_scaled, y_train))
```

```
BalancedRandomForestClassifier(n_estimators=1000, random_state=1)
```

In [39]:
```
brf.feature_importances_
```

Out[39]:
```
array([0.01109375, 0.02921056, 0.01627177, 0.01818152, 0.01921447,
       0.00479078, 0.00631991, 0.0103792 , 0.00201208, 0.01560989,
       0.01203236, 0.01630707, 0.01616746, 0.05717909, 0.05572968,
       0.07272245, 0.06403984, 0.0080715 , 0.        , 0.        ,
       0.0585881 , 0.00034626, 0.        , 0.        , 0.00341615,
       0.01478274, 0.005766  , 0.0065586 , 0.00510869, 0.0078655 ,
       0.0144914 , 0.01350968, 0.01682071, 0.00495961, 0.00693035,
       0.01660416, 0.01328503, 0.01429435, 0.00791591, 0.008074  ,
       0.00982653, 0.00900702, 0.01513932, 0.01507902, 0.01680095,
       0.00021578, 0.        , 0.01635283, 0.01581746, 0.01102797,
       0.0102071 , 0.00677331, 0.01239741, 0.01810684, 0.00332916,
       0.00880962, 0.00919837, 0.00806809, 0.00887901, 0.0099447 ,
       0.00880802, 0.01075197, 0.00954607, 0.00956708, 0.        ,
       0.        , 0.00118174, 0.00706752, 0.00940151, 0.00749028,
       0.00179645, 0.        , 0.01576515, 0.01366145, 0.01537165,
       0.01290981, 0.00040282, 0.00177668, 0.00234347, 0.00186362,
       0.00233897, 0.0021412 , 0.00302859, 0.        , 0.        ,
       0.0015259 , 0.00162895])
```

In [40]:
```
brf.predict(X_train_scaled)
```

Out[40]:
```
array(['low_risk', 'low_risk', 'high_risk', ..., 'low_risk', 'low_risk',
       'low_risk'], dtype=object)
```

In [41]:
```
# Calculated the balanced accuracy score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import balanced_accuracy_score

y_pred = brf.predict(X_test_scaled)
print(f'the random forest classifier balanced accuracy score is: {balanced_ac
```

```
the random forest classifier balanced accuracy score is: 0.7365
```

In [42]:
```
#Display the confusion matrix
y_pred = brf.predict(X_test_scaled)
confusion_matrix(y_test, y_pred)
```

Out[42]:
```
array([[   53,    34],
       [ 2332, 14786]], dtype=int64)
```

In [43]:
```
# Print the imbalanced classification report
y_pred_rf = brf.predict(X_test_scaled)
print(classification_report_imbalanced(y_test, y_pred_rf))
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.02      0.61      0.86      0.04      0.73      0.51
87
   low_risk       1.00      0.86      0.61      0.93      0.73      0.54      1
7118

avg / total       0.99      0.86      0.61      0.92      0.73      0.54      1
7205
```

In [44]:
```
# List the features sorted in descending order by feature importance
feature_importances = pd.DataFrame(brf.feature_importances_,
                                   index = X_train_scaled.columns,
                                   columns=
['importance']).sort_values('importance',
ascending=False)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-44-8fd3b38f8dc9> in <module>
      1 # List the features sorted in descending order by feature importance
      2 feature_importances = pd.DataFrame(brf.feature_importances_,
----> 3                                    index = X_train_scaled.columns,
      4                                    columns=
      5 ['importance']).sort_values('importance',

AttributeError: 'numpy.ndarray' object has no attribute 'columns'
```

Easy Ensemble Classifier

In [32]:
```
# Train the Classifier
from imblearn.ensemble import EasyEnsembleClassifier
eec = EasyEnsembleClassifier(n_estimators=100, random_state=1)
eec.fit(X_train_scaled, y_train)
```

Out[32]:
```
EasyEnsembleClassifier(n_estimators=100, random_state=1)
```

In [33]:
```
# Calculated the balanced accuracy score
from sklearn.metrics import balanced_accuracy_score

y_pred = eec.predict(X_test_scaled)
print(f'the easy ensemble classifier balanced accuracy score is: {balanced_ac
```

```
the easy ensemble classifier balanced accuracy score is: 0.7452
```

In [34]:
```
# Display the confusion matrix
y_pred = eec.predict(X_test_scaled)
confusion_matrix(y_test, y_pred)
```

Out[34]:
```
array([[   61,    26],
       [ 3609, 13509]], dtype=int64)
```

In [36]:
```
# Print the imbalanced classification report
y_pred_rf = eec.predict(X_test_scaled)
print(classification_report_imbalanced(y_test, y_pred_rf))
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.02      0.70      0.79      0.03      0.74      0.55
87
   low_risk       1.00      0.79      0.70      0.88      0.74      0.56     1
7118

avg / total       0.99      0.79      0.70      0.88      0.74      0.56     1
7205
```

Final Questions

1. Which model had the best balanced accuracy score?

   YOUR ANSWER HERE.

2. Which model had the best recall score?

   YOUR ANSWER HERE.

3. Which model had the best geometric mean score?

   YOUR ANSWER HERE.

4. What are the top three features?

YOUR ANSWER HERE.

In [ ]: