MSDS 422-57 | Module 5 Assignment 2 | Group 5
Jul 24, 2022 | Company Bankruptcy Prediction | Scott Jue
Dr. Anil Chaturvedi | | Zach Watson

The first step of our EDA is exploring the shape of our data (6,819, 96). We followed up by looking at all the 96 columns of data and checking for missing values. There are none. We then isolated the columns with two or fewer unique values, which were 'Bankrupt?', 'Liability-Assets Flag,' and 'Net Income Flag.' We determined that 'Bankrupt?' would be the response variable for our models. From the data, we calculated only 3.3% of companies were bankrupted, which shows an imbalance in the response variable. We observed 220 cases of bankruptcy (1) and 6,599 cases of non-bankruptcy (0), making it a relatively rare occurrence. We removed 'Net Income Flag' because it was uniform (every value was 1) and didn't provide any predictive value. We also removed 'Liability -Assets Flag' because only eight rows had values of 1 (the rest had 0), so it also would not provide a good predictive value for our models.

Our data preparation continued with additional feature selection. We used SequentialFeatureSelector from sklearn. We had our feature selector tuned to select features in a step-wise pattern, starting with the best predictor of 'Bankruptcy?' and adding the next best until 15 features were selected. This was a shift from last week's model, which dropped variables based on VIF.

Next, we used an isolation forest to remove 5% of the anomalies within the dataset using IsolationForest from sklearn. The isolation forest looked for anomalies across all potential independent (predictor) variables. After removing 5% of the rows with anomalies, the shape of our dataframe shrunk to include 6,478 values. We split the remaining data into two sets: training (80%; 5,182 values) and testing (20%; 1,296 values). We previously used SMOTE to correct for rare data, but this week we stratified our testing based on the number of bankruptcies. All of our models performed more accurately and faster using the stratified data. We followed by scaling the independent variables for both datasets using StandardScalar from sklearn. We built three new machine learning models to test our dataset.

Our Random Forest Model Classifier model used the preset hyperparameters and 100 estimators; it received an accuracy score of 0.97 and a cross-validation score of 0.97. (We use 5-fold cross-validation in all instances.) We tuned the hyperparameters using a grid search model and found the best parameters for our model: criterion is gini; max_depth is 2; max_features is 15 and n_estimators is 500. These parameters return a cross-validation score of 0.97 and an accuracy score of 0.97, so our model did not improve much with the tuned parameters. We used these parameters for all of our subsequent tree and forest models. Our Random Forest Model Classifier model ultimately received an F1-score of 0.22.

Our Gradient Boosted Trees model performed best and received our highest F1-score of 0.41 and an accuracy score of 0.97. The model did similarly well on the goodness of fit tests and received a TPR (Recall) value of 0.41, an FPR value of 0.02, and a precision value of 0.42. Interestingly, our Gradient Boosting Trees model with early stopping received a slightly higher TPR (Recall) score of 0.49, although its frequency of false positives also increases by about 0.0.8 and the F1-score decreases to 0.39.

Finally, our Extra Trees Model received an F1-score of 0.27 and the highest accuracy score at 0.98. More interestingly, the model has a low TPR of 0.16 and an FPR of 0, meaning it underestimates the number of bankrupt companies. We also included our new F1-scores for models developed last week, where you'll see our F1 values decreased for the SVM and Logistic Regression models but increased for the naïve Bayes model.

Ultimately, our maximum F1-score increased from 0.28 to 0.41 between weeks. This increase could be the result of changing variable selection to a step-wise sequence or replacing SMOTE with a stratified train-test split; however, the increase is most likely caused because tree and forest models are better suited for this classification problem. But, our models could still improve the estimations of bankrupt properties. Additional areas for exploration include further hyperparameter tuning, reintroducing SMOTE, and reevaluating variable selection methods.

Appendix:

	Model	TPR	FPR	precision	recall	accuracy	f1-value
0	Random Forest	[1.0, 0.14]	[0.86, 0.0]	[0.98, 0.56]	[1.0, 0.14]	[0.97, 0.97]	0.22
1	Gradient Boosted Trees	[0.98, 0.41]	[0.59, 0.02]	[0.98, 0.42]	[0.98, 0.41]	[0.97, 0.97]	0.41
2	Gradient Boosted Trees with Early Stopping	[0.97, 0.49]	[0.51, 0.03]	[0.98, 0.33]	[0.97, 0.49]	[0.96, 0.96]	0.39
3	Extra Trees	[1.0, 0.16]	[0.84, 0.0]	[0.98, 0.86]	[1.0, 0.16]	[0.98, 0.98]	0.27
4	SVM	[1.0, 0.0]	[1.0, 0.0]	[0.97, nan]	[1.0, 0.0]	[0.97, 0.97]	0.00
5	Logistic Regression	[1.0, 0.11]	[0.89, 0.0]	[0.97, 0.5]	[1.0, 0.11]	[0.97, 0.97]	0.18
6	naive bayes	[0.92, 0.65]	[0.35, 0.08]	[0.99, 0.19]	[0.92, 0.65]	[0.91, 0.91]	0.29

Module_5_Assignment_2

July 24, 2022

1 Intro

1.1 Links

https://canvas.northwestern.edu/courses/167719/assignments/1078603?module_item_id=2319248

https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction

1.2 Modules

[1]:
```
#For data manipulation and visualization
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from numpy import array
from numpy import arange

#For Isolation Forest from sklearn
from sklearn.ensemble import IsolationForest
from enum import auto

#From sklearn (SVM, Logistic, Bayes)
from sklearn.svm import SVC
from sklearn import svm

from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import ElasticNetCV

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold, StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.feature_selection import SelectKBest, f_classif, RFECV
```

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.naive_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier, GradientBoostingRegressor,␣
↪ExtraTreesClassifier, RandomForestRegressor
from sklearn.svm import LinearSVC
from sklearn.neural_network import MLPClassifier

from sklearn import metrics

from sklearn.metrics import f1_score, classification_report, confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import precision_score, recall_score,␣
↪precision_recall_curve
from sklearn.metrics import mean_squared_error

#Other
from math import sqrt
```

1.3 Import Files

[2]:
```
#Import data.csv from the Kaggle page linked above
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving data.csv to data.csv
```

[3]:
```
df = pd.read_csv("data.csv")
```

2 EDA

2.1 Intro Stats

[4]:
```
df.shape
```

[4]:
```
(6819, 96)
```

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6819 entries, 0 to 6818
Data columns (total 96 columns):
 #   Column                                                    Non-Null Count
Dtype
---  ------                                                    --------------
-----
 0   Bankrupt?                                                 6819 non-null
int64
 1    ROA(C) before interest and depreciation before interest  6819 non-null
float64
 2    ROA(A) before interest and % after tax                   6819 non-null
float64
 3    ROA(B) before interest and depreciation after tax        6819 non-null
float64
 4    Operating Gross Margin                                   6819 non-null
float64
 5    Realized Sales Gross Margin                              6819 non-null
float64
 6    Operating Profit Rate                                    6819 non-null
float64
 7    Pre-tax net Interest Rate                                6819 non-null
float64
 8    After-tax net Interest Rate                              6819 non-null
float64
 9    Non-industry income and expenditure/revenue              6819 non-null
float64
 10   Continuous interest rate (after tax)                     6819 non-null
float64
 11   Operating Expense Rate                                   6819 non-null
float64
 12   Research and development expense rate                    6819 non-null
float64
 13   Cash flow rate                                           6819 non-null
float64
 14   Interest-bearing debt interest rate                      6819 non-null
float64
 15   Tax rate (A)                                             6819 non-null
float64
 16   Net Value Per Share (B)                                  6819 non-null
float64
 17   Net Value Per Share (A)                                  6819 non-null
float64
 18   Net Value Per Share (C)                                  6819 non-null
float64
 19   Persistent EPS in the Last Four Seasons                  6819 non-null
float64
 20   Cash Flow Per Share                                      6819 non-null
```

```
float64
 21   Revenue Per Share (Yuan ¥)                                  6819 non-null
float64
 22   Operating Profit Per Share (Yuan ¥)                         6819 non-null
float64
 23   Per Share Net profit before tax (Yuan ¥)                    6819 non-null
float64
 24   Realized Sales Gross Profit Growth Rate                     6819 non-null
float64
 25   Operating Profit Growth Rate                                6819 non-null
float64
 26   After-tax Net Profit Growth Rate                            6819 non-null
float64
 27   Regular Net Profit Growth Rate                              6819 non-null
float64
 28   Continuous Net Profit Growth Rate                           6819 non-null
float64
 29   Total Asset Growth Rate                                     6819 non-null
float64
 30   Net Value Growth Rate                                       6819 non-null
float64
 31   Total Asset Return Growth Rate Ratio                        6819 non-null
float64
 32   Cash Reinvestment %                                         6819 non-null
float64
 33   Current Ratio                                               6819 non-null
float64
 34   Quick Ratio                                                 6819 non-null
float64
 35   Interest Expense Ratio                                      6819 non-null
float64
 36   Total debt/Total net worth                                  6819 non-null
float64
 37   Debt ratio %                                                6819 non-null
float64
 38   Net worth/Assets                                            6819 non-null
float64
 39   Long-term fund suitability ratio (A)                        6819 non-null
float64
 40   Borrowing dependency                                        6819 non-null
float64
 41   Contingent liabilities/Net worth                            6819 non-null
float64
 42   Operating profit/Paid-in capital                            6819 non-null
float64
 43   Net profit before tax/Paid-in capital                       6819 non-null
float64
 44   Inventory and accounts receivable/Net value                 6819 non-null
```

```
float64
 45   Total Asset Turnover                                    6819 non-null
float64
 46   Accounts Receivable Turnover                            6819 non-null
float64
 47   Average Collection Days                                 6819 non-null
float64
 48   Inventory Turnover Rate (times)                         6819 non-null
float64
 49   Fixed Assets Turnover Frequency                         6819 non-null
float64
 50   Net Worth Turnover Rate (times)                         6819 non-null
float64
 51   Revenue per person                                      6819 non-null
float64
 52   Operating profit per person                             6819 non-null
float64
 53   Allocation rate per person                              6819 non-null
float64
 54   Working Capital to Total Assets                         6819 non-null
float64
 55   Quick Assets/Total Assets                               6819 non-null
float64
 56   Current Assets/Total Assets                             6819 non-null
float64
 57   Cash/Total Assets                                       6819 non-null
float64
 58   Quick Assets/Current Liability                          6819 non-null
float64
 59   Cash/Current Liability                                  6819 non-null
float64
 60   Current Liability to Assets                             6819 non-null
float64
 61   Operating Funds to Liability                            6819 non-null
float64
 62   Inventory/Working Capital                               6819 non-null
float64
 63   Inventory/Current Liability                             6819 non-null
float64
 64   Current Liabilities/Liability                           6819 non-null
float64
 65   Working Capital/Equity                                  6819 non-null
float64
 66   Current Liabilities/Equity                              6819 non-null
float64
 67   Long-term Liability to Current Assets                   6819 non-null
float64
 68   Retained Earnings to Total Assets                       6819 non-null
```

```
float64
 69   Total income/Total expense                                 6819 non-null
float64
 70   Total expense/Assets                                       6819 non-null
float64
 71   Current Asset Turnover Rate                                6819 non-null
float64
 72   Quick Asset Turnover Rate                                  6819 non-null
float64
 73   Working capitcal Turnover Rate                             6819 non-null
float64
 74   Cash Turnover Rate                                         6819 non-null
float64
 75   Cash Flow to Sales                                         6819 non-null
float64
 76   Fixed Assets to Assets                                     6819 non-null
float64
 77   Current Liability to Liability                             6819 non-null
float64
 78   Current Liability to Equity                                6819 non-null
float64
 79   Equity to Long-term Liability                              6819 non-null
float64
 80   Cash Flow to Total Assets                                  6819 non-null
float64
 81   Cash Flow to Liability                                     6819 non-null
float64
 82   CFO to Assets                                              6819 non-null
float64
 83   Cash Flow to Equity                                        6819 non-null
float64
 84   Current Liability to Current Assets                        6819 non-null
float64
 85   Liability-Assets Flag                                      6819 non-null
int64
 86   Net Income to Total Assets                                 6819 non-null
float64
 87   Total assets to GNP price                                  6819 non-null
float64
 88   No-credit Interval                                         6819 non-null
float64
 89   Gross Profit to Sales                                      6819 non-null
float64
 90   Net Income to Stockholder's Equity                         6819 non-null
float64
 91   Liability to Equity                                        6819 non-null
float64
 92   Degree of Financial Leverage (DFL)                         6819 non-null
```

```
float64
 93   Interest Coverage Ratio (Interest expense to EBIT)       6819 non-null
float64
 94   Net Income Flag                                          6819 non-null
int64
 95   Equity to Liability                                      6819 non-null
float64
dtypes: float64(93), int64(3)
memory usage: 5.0 MB
```

[6]:
```
# check for missing values
print(df.isna().sum().sum())
print(np.isnan(df).sum().sum())
print(df.isnull().sum().sum())
```

```
0
0
0
```

[7]:
```
df.head(10)
```

[7]:
```
   Bankrupt?  ROA(C) before interest and depreciation before interest  \
0          1                                           0.370594
1          1                                           0.464291
2          1                                           0.426071
3          1                                           0.399844
4          1                                           0.465022
5          1                                           0.388680
6          0                                           0.390923
7          0                                           0.508361
8          0                                           0.488519
9          0                                           0.495686

    ROA(A) before interest and % after tax  \
0                              0.424389
1                              0.538214
2                              0.499019
3                              0.451265
4                              0.538432
5                              0.415177
6                              0.445704
7                              0.570922
8                              0.545137
9                              0.550916

    ROA(B) before interest and depreciation after tax  \
0                                          0.405750
```

```
1                                            0.516730
2                                            0.472295
3                                            0.457733
4                                            0.522298
5                                            0.419134
6                                            0.436158
7                                            0.559077
8                                            0.543284
9                                            0.542963

   Operating Gross Margin   Realized Sales Gross Margin  \
0                0.601457                      0.601457
1                0.610235                      0.610235
2                0.601450                      0.601364
3                0.583541                      0.583541
4                0.598783                      0.598783
5                0.590171                      0.590251
6                0.619950                      0.619950
7                0.601738                      0.601717
8                0.603612                      0.603612
9                0.599209                      0.599209

   Operating Profit Rate   Pre-tax net Interest Rate  \
0               0.998969                    0.796887
1               0.998946                    0.797380
2               0.998857                    0.796403
3               0.998700                    0.796967
4               0.998973                    0.797366
5               0.998758                    0.796903
6               0.998993                    0.797012
7               0.999009                    0.797449
8               0.998961                    0.797414
9               0.999001                    0.797404

   After-tax net Interest Rate   Non-industry income and expenditure/revenue  \
0                     0.808809                                      0.302646
1                     0.809301                                      0.303556
2                     0.808388                                      0.302035
3                     0.808966                                      0.303350
4                     0.809304                                      0.303475
5                     0.808771                                      0.303116
6                     0.808960                                      0.302814
7                     0.809362                                      0.303545
8                     0.809338                                      0.303584
9                     0.809320                                      0.303483

  …   Net Income to Total Assets   Total assets to GNP price  \
```

```
0  …                    0.716845                    0.009219
1  …                    0.795297                    0.008323
2  …                    0.774670                    0.040003
3  …                    0.739555                    0.003252
4  …                    0.795016                    0.003878
5  …                    0.710420                    0.005278
6  …                    0.736619                    0.018372
7  …                    0.815350                    0.010005
8  …                    0.803647                    0.000824
9  …                    0.804195                    0.005798

    No-credit Interval   Gross Profit to Sales  \
0             0.622879                0.601453
1             0.623652                0.610237
2             0.623841                0.601449
3             0.622929                0.583538
4             0.623521                0.598782
5             0.622605                0.590172
6             0.623655                0.619949
7             0.623843                0.601739
8             0.623977                0.603613
9             0.623865                0.599205

    Net Income to Stockholder's Equity   Liability to Equity  \
0                             0.827890              0.290202
1                             0.839969              0.283846
2                             0.836774              0.290189
3                             0.834697              0.281721
4                             0.839973              0.278514
5                             0.829939              0.285087
6                             0.829980              0.292504
7                             0.841459              0.278607
8                             0.840487              0.276423
9                             0.840688              0.279388

    Degree of Financial Leverage (DFL)  \
0                             0.026601
1                             0.264577
2                             0.026555
3                             0.026697
4                             0.024752
5                             0.026675
6                             0.026622
7                             0.027031
8                             0.026891
9                             0.027243
```

```
   Interest Coverage Ratio (Interest expense to EBIT)   Net Income Flag  \
0                                            0.564050                1
1                                            0.570175                1
2                                            0.563706                1
3                                            0.564663                1
4                                            0.575617                1
5                                            0.564538                1
6                                            0.564200                1
7                                            0.566089                1
8                                            0.565592                1
9                                            0.566668                1

    Equity to Liability
0              0.016469
1              0.020794
2              0.016474
3              0.023982
4              0.035490
5              0.019534
6              0.015663
7              0.034889
8              0.065826
9              0.030801

[10 rows x 96 columns]
```

[8]:
```python
# create a for loop to get the categorical columns with 2 or less than 2 unique
↪values
list_1=[]
for i in df.columns:
  x=df[i].value_counts()
  if len(x)<=2:
    list_1.append(i)
  else:
    continue
```

[9]:
```python
# categorical variables (value_counts <= 2 )
list_1
```

[9]:
```
['Bankrupt?', ' Liability-Assets Flag', ' Net Income Flag']
```

2.1.1 Checking [Bankrupt?]

[10]:
```python
df['Bankrupt?'].unique()
```

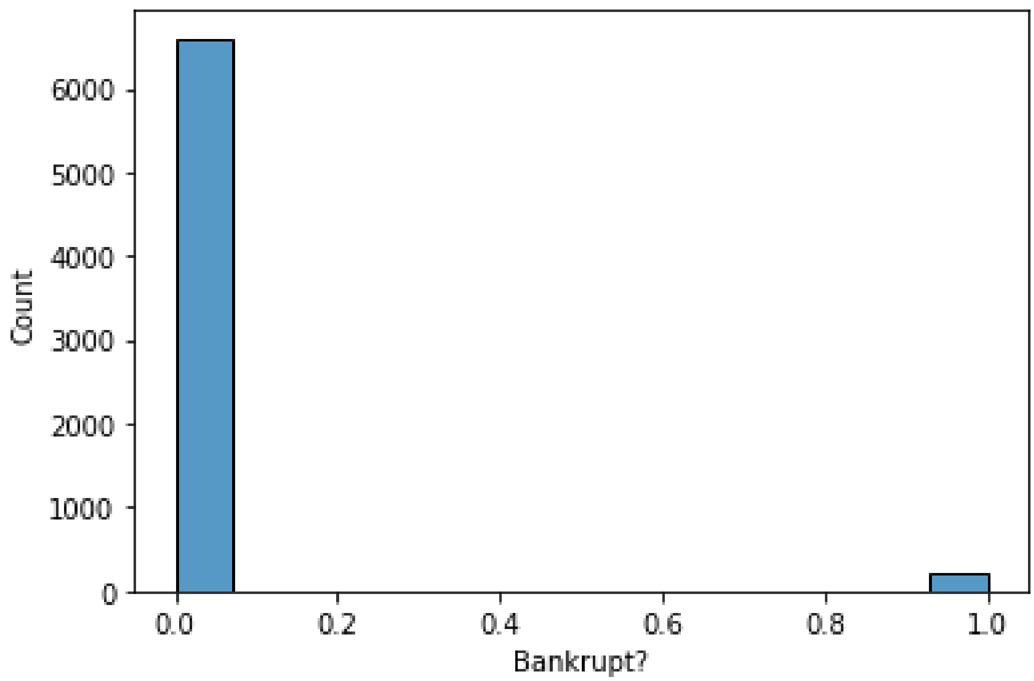
[10]: array([1, 0])

[11]:
```
df['Bankrupt?'].value_counts()
```

[11]:
```
0    6599
1     220
Name: Bankrupt?, dtype: int64
```

[12]:
```
sns.histplot(x="Bankrupt?", data=df)
```

[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff2abe8ab50>



2.1.2 Checking [Net Income Flag]

[13]:
```
df[' Net Income Flag'].unique()
```

[13]: array([1])

[14]:
```
df[' Net Income Flag'].value_counts()
```

[14]:
```
1    6819
Name:  Net Income Flag, dtype: int64
```

Every value is a 1 for this feature, so we can drop this column since it doesn't provide us any predictive value.

2.1.3 Checking [Liability-Assets Flag]

[15]:
```
df[' Liability-Assets Flag'].unique()
```

[15]:
```
array([0, 1])
```

[16]:
```
df[' Liability-Assets Flag'].value_counts()
```

[16]:
```
0    6811
1       8
Name:  Liability-Assets Flag, dtype: int64
```

This feature may not provide good predictive value to the model since it has a large imbalance between the 0 and 1 classes. Therefore, we can also drop this column.

2.2 Feature Selection

[17]:
```
y = df['Bankrupt?']
```

[18]:
```
X = df.drop(columns = ['Bankrupt?',' Liability-Assets Flag',' Net Income Flag'])
```

[19]:
```
# creating KNN classifier
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
```

[20]:
```
# create step-wise model for feature selection
sfs1 = SequentialFeatureSelector(knn, n_features_to_select=15,␣
 ↪direction='forward', cv=5)

sfs1 = sfs1.fit(X, y)
```

[21]:
```
# get list of selected features
features = X.columns
selected_features = np.array(features)[sfs1.get_support()]
print(selected_features)
```

```
[' Operating Profit Rate' ' Pre-tax net Interest Rate'
 ' After-tax net Interest Rate' ' Continuous interest rate (after tax)'
 ' Interest-bearing debt interest rate'
 ' Realized Sales Gross Profit Growth Rate'
 ' Continuous Net Profit Growth Rate' ' Net Value Growth Rate'
```

```
 ' Contingent liabilities/Net worth' ' Total income/Total expense'
 ' Quick Asset Turnover Rate' ' Working capitcal Turnover Rate'
 ' Cash Flow to Sales' ' Degree of Financial Leverage (DFL)'
 ' Equity to Liability']
```

[22]:
```
# reassign X to be only the selected features
X = X[selected_features]
```

[23]:
```
X.head()
```

[23]:
```
    Operating Profit Rate   Pre-tax net Interest Rate  \
0                0.998969                    0.796887
1                0.998946                    0.797380
2                0.998857                    0.796403
3                0.998700                    0.796967
4                0.998973                    0.797366

    After-tax net Interest Rate   Continuous interest rate (after tax)  \
0                      0.808809                               0.780985
1                      0.809301                               0.781506
2                      0.808388                               0.780284
3                      0.808966                               0.781241
4                      0.809304                               0.781550

    Interest-bearing debt interest rate  \
0                              0.000725
1                              0.000647
2                              0.000790
3                              0.000449
4                              0.000686

    Realized Sales Gross Profit Growth Rate  \
0                                  0.022102
1                                  0.022080
2                                  0.022760
3                                  0.022046
4                                  0.022096

    Continuous Net Profit Growth Rate   Net Value Growth Rate  \
0                            0.217535                0.000327
1                            0.217620                0.000443
2                            0.217601                0.000396
3                            0.217568                0.000382
4                            0.217626                0.000439

    Contingent liabilities/Net worth   Total income/Total expense  \
0                           0.006479                     0.002022
```

```
1                        0.005835                      0.002226
2                        0.006562                      0.002060
3                        0.005366                      0.001831
4                        0.006624                      0.002224

    Quick Asset Turnover Rate   Working capitcal Turnover Rate  \
0                6.550000e+09                         0.593831
1                7.700000e+09                         0.593916
2                1.022676e-03                         0.594502
3                6.050000e+09                         0.593889
4                5.050000e+09                         0.593915

    Cash Flow to Sales   Degree of Financial Leverage (DFL)  \
0             0.671568                             0.026601
1             0.671570                             0.264577
2             0.671571                             0.026555
3             0.671519                             0.026697
4             0.671563                             0.024752

    Equity to Liability
0              0.016469
1              0.020794
2              0.016474
3              0.023982
4              0.035490
```

[24]:

```
# create df1 using X
df1 = X
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6819 entries, 0 to 6818
Data columns (total 15 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0    Operating Profit Rate                    6819 non-null   float64
 1    Pre-tax net Interest Rate                6819 non-null   float64
 2    After-tax net Interest Rate              6819 non-null   float64
 3    Continuous interest rate (after tax)     6819 non-null   float64
 4    Interest-bearing debt interest rate      6819 non-null   float64
 5    Realized Sales Gross Profit Growth Rate  6819 non-null   float64
 6    Continuous Net Profit Growth Rate        6819 non-null   float64
 7    Net Value Growth Rate                    6819 non-null   float64
 8    Contingent liabilities/Net worth         6819 non-null   float64
 9    Total income/Total expense               6819 non-null   float64
 10   Quick Asset Turnover Rate                6819 non-null   float64
 11   Working capitcal Turnover Rate           6819 non-null   float64
```

```
 12   Cash Flow to Sales                        6819 non-null   float64
 13   Degree of Financial Leverage (DFL)        6819 non-null   float64
 14   Equity to Liability                       6819 non-null   float64
dtypes: float64(15)
memory usage: 799.2 KB
```

2.3 Removing Anomalies with Isolation Forests

[25]:
```
#Isolation Forest Identifies anomalies
model=IsolationForest(n_estimators=100, contamination=float(.05),␣
↪random_state=42)
model.fit(X)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does
not have valid feature names, but IsolationForest was fitted with feature names
  "X does not have valid feature names, but"
```

[25]:
```
IsolationForest(contamination=0.05, random_state=42)
```

[26]:
```
df1['scores'] = model.decision_function(X)
X = X[selected_features]
```

[27]:
```
#Removing anomolies from the dataset
df1['anomaly_score'] = model.predict(X)

df2 = pd.concat([y, df1], axis=1)

df3 = df2[df2['anomaly_score']!=-1]

df4 = df3.drop(columns = ['scores', 'anomaly_score'])
df4.head(10)
```

[27]:
```
    Bankrupt?  Operating Profit Rate  Pre-tax net Interest Rate  \
0           1               0.998969                   0.796887
1           1               0.998946                   0.797380
3           1               0.998700                   0.796967
4           1               0.998973                   0.797366
5           1               0.998758                   0.796903
6           0               0.998993                   0.797012
7           0               0.999009                   0.797449
8           0               0.998961                   0.797414
9           0               0.999001                   0.797404
10          0               0.998978                   0.797535

    After-tax net Interest Rate  Continuous interest rate (after tax)  \
0                      0.808809                              0.780985
```

```
1                        0.809301                              0.781506
3                        0.808966                              0.781241
4                        0.809304                              0.781550
5                        0.808771                              0.781069
6                        0.808960                              0.781180
7                        0.809362                              0.781621
8                        0.809338                              0.781598
9                        0.809320                              0.781574
10                       0.809460                              0.781629

     Interest-bearing debt interest rate  \
0                               0.000725
1                               0.000647
3                               0.000449
4                               0.000686
5                               0.000716
6                               0.000805
7                               0.000630
8                               0.000737
9                               0.000672
10                              0.000549

     Realized Sales Gross Profit Growth Rate  \
0                                   0.022102
1                                   0.022080
3                                   0.022046
4                                   0.022096
5                                   0.021565
6                                   0.022112
7                                   0.022114
8                                   0.022128
9                                   0.022118
10                                  0.022107

     Continuous Net Profit Growth Rate   Net Value Growth Rate  \
0                             0.217535                0.000327
1                             0.217620                0.000443
3                             0.217568                0.000382
4                             0.217626                0.000439
5                             0.217566                0.000352
6                             0.217604                0.000352
7                             0.217633                0.000451
8                             0.217654                0.000453
9                             0.217700                0.000445
10                            0.217580                0.000449

     Contingent liabilities/Net worth   Total income/Total expense  \
```

```
0                          0.006479                        0.002022
1                          0.005835                        0.002226
3                          0.005366                        0.001831
4                          0.006624                        0.002224
5                          0.005749                        0.001866
6                          0.008044                        0.002121
7                          0.006383                        0.002360
8                          0.005366                        0.002274
9                          0.005819                        0.002269
10                         0.008130                        0.002338

    Quick Asset Turnover Rate   Working capitcal Turnover Rate  \
0                6.550000e+09                         0.593831
1                7.700000e+09                         0.593916
3                6.050000e+09                         0.593889
4                5.050000e+09                         0.593915
5                2.810000e+09                         0.593846
6                9.560000e+09                         0.593893
7                6.180000e+09                         0.593937
8                9.840000e+09                         0.593959
9                3.600000e+09                         0.593936
10               2.920000e+09                         0.593916

    Cash Flow to Sales   Degree of Financial Leverage (DFL)  \
0             0.671568                             0.026601
1             0.671570                             0.264577
3             0.671519                             0.026697
4             0.671563                             0.024752
5             0.671568                             0.026675
6             0.671562                             0.026622
7             0.671572                             0.027031
8             0.671576                             0.026891
9             0.671572                             0.027243
10            0.671572                             0.026971

    Equity to Liability
0              0.016469
1              0.020794
3              0.023982
4              0.035490
5              0.019534
6              0.015663
7              0.034889
8              0.065826
9              0.030801
10             0.036572
```

2.4 Data Prep

2.4.1 Split the Data for Training

[28]:
```
# create X dataframe with dependent variables
X = df4.drop(columns=['Bankrupt?'], axis=1)
X.shape
```

[28]: (6478, 15)

[29]:
```
# create y dataframe for response variable
y = df4['Bankrupt?']
```

[30]:
```
# split data in to training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪stratify = y, random_state=42)
```

2.4.2 Scale the Data

[31]:
```
# use StandardScaler to scale features
scaler = StandardScaler()

#scale the training data
X_train_scaled = scaler.fit_transform(X_train)
X_train = pd.DataFrame(X_train_scaled, columns = X_train.columns)
# scale test data for model testing
X_test_scaled = scaler.fit_transform(X_test)
X_test = pd.DataFrame(X_test_scaled, columns = X_test.columns)
```

3 Models

3.1 Random Forest

[32]:
```
# Random forest classifier model

forest_clf = RandomForestClassifier(n_estimators=100, random_state=42)
forest_clf.fit(X_train, y_train)
```

[32]: RandomForestClassifier(random_state=42)

[33]:
```
# y predictions
y_pred = forest_clf.predict(X_test)
```

[34]:
```python
# cross-validation
forest_scores = cross_val_score(forest_clf, X_train, y_train, cv=5)
forest_scores.mean()
```

[34]: 0.9720188917392389

[35]:
```python
# random forest model accuracy
from sklearn.metrics import accuracy_score

accuracy_score(y_true = y_test, y_pred = y_pred)
```

[35]: 0.9722222222222222

3.1.1 Tuning hyperparmeters

[36]:
```python
# Create the parameter grid based on the results of random search
param_grid = {
    'max_depth': [2, 3, 4],
    'max_features': [5, 10, 15],
    'n_estimators': [100, 500, 1000],
    'criterion': ['gini', 'entropy']
}
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = forest_clf, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)
```

[37]:
```python
# Fit the grid search to the data
rf_grid = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
```

[37]: {'criterion': 'gini', 'max_depth': 2, 'max_features': 15, 'n_estimators': 500}

Best paramaters for Random Forest model:

{'criterion': 'gini', 'max_depth': 2, 'max_features': 15, 'n_estimators': 500}

[38]:
```python
# creating best_parameters variable to be used for other tree models
best_parameters = {
    'max_depth': 2,
    'max_features': 15,
    'n_estimators': 500,
}
```

[39]:
```python
# creating random forest model using best parameters
forest_clf_tuned = RandomForestClassifier(**best_parameters, random_state=42)
forest_clf_tuned.fit(X_train, y_train)
```

[39]:
```
RandomForestClassifier(max_depth=2, max_features=15, n_estimators=500,
                       random_state=42)
```

[40]:
```python
# y predictions
y_pred = forest_clf_tuned.predict(X_test)
```

[41]:
```python
# cross-validation
forest_clf_tuned_scores = cross_val_score(forest_clf_tuned, X_train, y_train,␣
 ↪cv=5)
forest_clf_tuned_scores.mean()
```

[41]:
```
0.972211941932289
```

[42]:
```python
# gradient boosted trees model accuracy
accuracy_score(y_true = y_test, y_pred = y_pred)
```

[42]:
```
0.9722222222222222
```

[43]:
```python
# create confusion matrix and add scores to table
cnf_matrix = confusion_matrix(y_test,y_pred)

FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)

FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
# Specificity or true negative rate
TNR = TN/(TN+FP)
# Precision or positive predictive value
PPV = TP/(TP+FP)
# Negative predictive value
NPV = TN/(TN+FN)
# Fall out or false positive rate
FPR = FP/(FP+TN)
# False negative rate
FNR = FN/(TP+FN)
```

```
# False discovery rate
FDR = FP/(TP+FP)
# Overall accuracy
ACC = (TP+TN)/(TP+FP+FN+TN)

results = pd.DataFrame(columns = ['Model', 'TPR', 'FPR', 'precision', 'recall',␣
 ↪'accuracy', 'f1-value'])

#eval
Model = 'Random Forest'
TPR = [round(num, 2) for num in TPR]
FPR = [round(num, 2) for num in FPR]
precision = [round(num, 2) for num in PPV]
recall = [round(num, 2) for num in TPR]
accuracy = [round(num, 2) for num in ACC]
f1_value = round(f1_score(y_pred, y_test),2)
row = [Model, TPR, FPR, precision, recall, accuracy, f1_value]
results = results.append(pd.DataFrame([row], columns=results.columns),␣
 ↪ignore_index=True)
results
```

[43]:

```
           Model          TPR          FPR     precision       recall  \
0  Random Forest  [1.0, 0.14]  [0.86, 0.0]  [0.98, 0.56]  [1.0, 0.14]

       accuracy  f1-value
0  [0.97, 0.97]      0.22
```

3.2 Gradient Boosted Trees

[44]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

[45]:

```
# creating gradient boosted trees model using best parameters from RF tuning
gbc = GradientBoostingClassifier(**best_parameters, learning_rate=0.01,␣
 ↪random_state=42)
gbc.fit(X_train, y_train)
```

[45]:

```
GradientBoostingClassifier(learning_rate=0.01, max_depth=2, max_features=15,
                           n_estimators=500, random_state=42)
```

[46]:

```
y_pred = gbc.predict(X_test)
```

[47]:

```
accuracy_score(y_true = y_test, y_pred = y_pred)
```

[47]:

```
0.966820987654321
```

[48]:
```python
# create confusion matrix and add scores to table
cnf_matrix = confusion_matrix(y_test,y_pred)

FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)

FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
# Specificity or true negative rate
TNR = TN/(TN+FP)
# Precision or positive predictive value
PPV = TP/(TP+FP)
# Negative predictive value
NPV = TN/(TN+FN)
# Fall out or false positive rate
FPR = FP/(FP+TN)
# False negative rate
FNR = FN/(TP+FN)
# False discovery rate
FDR = FP/(TP+FP)
# Overall accuracy
ACC = (TP+TN)/(TP+FP+FN+TN)

#eval
Model = 'Gradient Boosted Trees'
TPR = [round(num, 2) for num in TPR]
FPR = [round(num, 2) for num in FPR]
precision = [round(num, 2) for num in PPV]
recall = [round(num, 2) for num in TPR]
accuracy = [round(num, 2) for num in ACC]
f1_value = round(f1_score(y_pred, y_test),2)
row = [Model, TPR, FPR, precision, recall, accuracy, f1_value]
results = results.append(pd.DataFrame([row], columns=results.columns),␣
↪ignore_index=True)
results
```

[48]:
```
                    Model          TPR          FPR    precision  \
0           Random Forest   [1.0, 0.14]   [0.86, 0.0]  [0.98, 0.56]
1  Gradient Boosted Trees  [0.98, 0.41]  [0.59, 0.02]  [0.98, 0.42]
```

```
         recall      accuracy  f1-value
0    [1.0, 0.14]  [0.97, 0.97]      0.22
1   [0.98, 0.41]  [0.97, 0.97]      0.41
```

3.2.1 Gradient Boosting with Early stopping

[49]:
```python
# creating gradient boosted tree model with early stopping
gbc = GradientBoostingClassifier(**best_parameters, learning_rate=0.01,␣
↪random_state=42)
gbc.fit(X_train, y_train)

errors = [mean_squared_error(y_test, y_pred)
          for y_pred in gbc.staged_predict(X_test)]
bst_n_estimators = np.argmin(errors) + 1

gbc_best = GradientBoostingClassifier(max_depth=2,␣
↪n_estimators=bst_n_estimators, random_state=42)
gbc_best.fit(X_train, y_train)
```

[49]:
```
GradientBoostingClassifier(max_depth=2, n_estimators=115, random_state=42)
```

[50]:
```python
y_pred = gbc_best.predict(X_test)
```

[51]:
```python
# cross-validation
gbc_best_scores = cross_val_score(gbc_best, X_train, y_train, cv=5)
gbc_best_scores.mean()
```

[51]:
```
0.9720190779014309
```

[52]:
```python
# slow gradient boosted trees with early stopping model accuracy
accuracy_score(y_true = y_test, y_pred = y_pred)
```

[52]:
```
0.9567901234567902
```

[53]:
```python
# create confusion matrix and add scores to table
cnf_matrix = confusion_matrix(y_test,y_pred)

FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)

FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)
```

```
# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
# Specificity or true negative rate
TNR = TN/(TN+FP)
# Precision or positive predictive value
PPV = TP/(TP+FP)
# Negative predictive value
NPV = TN/(TN+FN)
# Fall out or false positive rate
FPR = FP/(FP+TN)
# False negative rate
FNR = FN/(TP+FN)
# False discovery rate
FDR = FP/(TP+FP)
# Overall accuracy
ACC = (TP+TN)/(TP+FP+FN+TN)

#eval
Model = 'Gradient Boosted Trees with Early Stopping'
TPR = [round(num, 2) for num in TPR]
FPR = [round(num, 2) for num in FPR]
precision = [round(num, 2) for num in PPV]
recall = [round(num, 2) for num in TPR]
accuracy = [round(num, 2) for num in ACC]
f1_value = round(f1_score(y_pred, y_test),2)
row = [Model, TPR, FPR, precision, recall, accuracy, f1_value]
results = results.append(pd.DataFrame([row], columns=results.columns),
 ↪ignore_index=True)
results
```

[53]:

```
                                         Model           TPR           FPR  \
0                                Random Forest   [1.0, 0.14]   [0.86, 0.0]
1                       Gradient Boosted Trees  [0.98, 0.41]  [0.59, 0.02]
2   Gradient Boosted Trees with Early Stopping  [0.97, 0.49]  [0.51, 0.03]

      precision        recall      accuracy  f1-value
0  [0.98, 0.56]   [1.0, 0.14]  [0.97, 0.97]      0.22
1  [0.98, 0.42]  [0.98, 0.41]  [0.97, 0.97]      0.41
2  [0.98, 0.33]  [0.97, 0.49]  [0.96, 0.96]      0.39
```

3.3 Extra Trees

[54]:
```
# creating extra trees model, using 500 estimators from RF hyperparameter tuning
extra_trees_clf = ExtraTreesClassifier(n_estimators=500, random_state=42)
extra_trees_clf.fit(X_train, y_train)
```

[54]: ExtraTreesClassifier(n_estimators=500, random_state=42)

[55]:
```
# cross-validation
extra_trees_scores = cross_val_score(extra_trees_clf, X_train, y_train, cv=5)
extra_trees_scores.mean()
```

[55]: 0.9718258415461888

[56]:
```
y_pred = extra_trees_clf.predict(X_test)
```

[57]:
```
# extra trees model accuracy
accuracy_score(y_true = y_test, y_pred = y_pred)
```

[57]: 0.9753086419753086

[58]:
```
# create confusion matrix and add scores to table
cnf_matrix = confusion_matrix(y_test,y_pred)

FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)

FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
# Specificity or true negative rate
TNR = TN/(TN+FP)
# Precision or positive predictive value
PPV = TP/(TP+FP)
# Negative predictive value
NPV = TN/(TN+FN)
# Fall out or false positive rate
FPR = FP/(FP+TN)
# False negative rate
FNR = FN/(TP+FN)
# False discovery rate
```

```
FDR = FP/(TP+FP)
# Overall accuracy
ACC = (TP+TN)/(TP+FP+FN+TN)

#eval
Model = 'Extra Trees'
TPR = [round(num, 2) for num in TPR]
FPR = [round(num, 2) for num in FPR]
precision = [round(num, 2) for num in PPV]
recall = [round(num, 2) for num in TPR]
accuracy = [round(num, 2) for num in ACC]
f1_value = round(f1_score(y_pred, y_test),2)
row = [Model, TPR, FPR, precision, recall, accuracy, f1_value]
results = results.append(pd.DataFrame([row], columns=results.columns),␣
 ↪ignore_index=True)
results
```

[58]:

```
                                   Model           TPR           FPR  \
0                          Random Forest   [1.0, 0.14]   [0.86, 0.0]
1                 Gradient Boosted Trees  [0.98, 0.41]  [0.59, 0.02]
2  Gradient Boosted Trees with Early Stopping  [0.97, 0.49]  [0.51, 0.03]
3                            Extra Trees   [1.0, 0.16]   [0.84, 0.0]

      precision        recall      accuracy  f1-value
0  [0.98, 0.56]   [1.0, 0.14]  [0.97, 0.97]      0.22
1  [0.98, 0.42]  [0.98, 0.41]  [0.97, 0.97]      0.41
2  [0.98, 0.33]  [0.97, 0.49]  [0.96, 0.96]      0.39
3  [0.98, 0.86]   [1.0, 0.16]  [0.98, 0.98]      0.27
```

3.4 Support Vector Machine (SVM)

[59]:

```
# SVM Classifier model
svm_clf = SVC(kernel="rbf", C=1, probability=True)
svm_clf.fit(X_train, y_train)
svm_model = svm_clf.fit(X_train, y_train)
```

[60]:

```
# kfold cross validation
score = cross_val_score(svm_model, X_train, y_train, cv=5, verbose=3)
score.mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] END … score: (test=0.971) total time=   1.0s

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.0s remaining:    0.0s

[CV] END … score: (test=0.971) total time=   0.9s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.9s remaining:    0.0s

[CV] END … score: (test=0.972) total time=   0.9s
[CV] END … score: (test=0.972) total time=   1.0s
[CV] END … score: (test=0.972) total time=   0.9s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    4.8s finished
```

[60]:
```
0.9716327913531385
```

[61]:
```python
# predictions
y_pred = svm_model.predict(X_test)
```

[62]:
```python
# precision and recall scores
print("Precision: {:.2f}%".format(100 * precision_score(y_test, y_pred)))
print("Recall: {:.2f}%".format(100 * recall_score(y_test, y_pred)))
```

```
Precision: 0.00%
Recall: 0.00%

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[63]:
```python
# f1 score
f1_score(y_test, y_pred)
```

[63]:
```
0.0
```

3.4.1 Tuning SVM Model Hyperparameters

[64]:
```python
# create grid search cross validation to tunr hyperparmeters of SVC model
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1,0.1,0.01,0.001]}

grid = GridSearchCV(SVC(probability=True,␣
↪kernel='rbf'),param_grid,refit=True,verbose=2)
grid.fit(X_train,y_train);
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[CV] END …C=0.1, gamma=1; total time=   2.6s
[CV] END …C=0.1, gamma=1; total time=   2.5s
[CV] END …C=0.1, gamma=1; total time=   2.6s
[CV] END …C=0.1, gamma=1; total time=   2.5s
[CV] END …C=0.1, gamma=1; total time=   2.6s
[CV] END …C=0.1, gamma=0.1; total time=   1.0s
[CV] END …C=0.1, gamma=0.1; total time=   0.9s
[CV] END …C=0.1, gamma=0.1; total time=   0.9s
```

```
[CV] END …C=0.1, gamma=0.1; total time=   1.0s
[CV] END …C=0.1, gamma=0.1; total time=   1.0s
[CV] END …C=0.1, gamma=0.01; total time=   0.6s
[CV] END …C=0.1, gamma=0.01; total time=   0.6s
[CV] END …C=0.1, gamma=0.01; total time=   0.6s
[CV] END …C=0.1, gamma=0.01; total time=   0.6s
[CV] END …C=0.1, gamma=0.01; total time=   0.6s
[CV] END …C=0.1, gamma=0.001; total time=   0.4s
[CV] END …C=0.1, gamma=0.001; total time=   0.4s
[CV] END …C=0.1, gamma=0.001; total time=   0.5s
[CV] END …C=0.1, gamma=0.001; total time=   0.5s
[CV] END …C=0.1, gamma=0.001; total time=   0.5s
[CV] END …C=1, gamma=1; total time=   2.8s
[CV] END …C=1, gamma=1; total time=   2.8s
[CV] END …C=1, gamma=1; total time=   2.8s
[CV] END …C=1, gamma=1; total time=   2.8s
[CV] END …C=1, gamma=1; total time=   2.8s
[CV] END …C=1, gamma=0.1; total time=   1.1s
[CV] END …C=1, gamma=0.1; total time=   1.0s
[CV] END …C=1, gamma=0.1; total time=   1.0s
[CV] END …C=1, gamma=0.1; total time=   1.1s
[CV] END …C=1, gamma=0.1; total time=   1.1s
[CV] END …C=1, gamma=0.01; total time=   0.7s
[CV] END …C=1, gamma=0.01; total time=   0.8s
[CV] END …C=1, gamma=0.01; total time=   0.8s
[CV] END …C=1, gamma=0.01; total time=   1.0s
[CV] END …C=1, gamma=0.01; total time=   1.7s
[CV] END …C=1, gamma=0.001; total time=   0.7s
[CV] END …C=1, gamma=0.001; total time=   0.6s
[CV] END …C=1, gamma=0.001; total time=   0.6s
[CV] END …C=1, gamma=0.001; total time=   0.6s
[CV] END …C=1, gamma=0.001; total time=   0.6s
[CV] END …C=10, gamma=1; total time=   2.5s
[CV] END …C=10, gamma=1; total time=   2.6s
[CV] END …C=10, gamma=1; total time=   2.5s
[CV] END …C=10, gamma=1; total time=   2.6s
[CV] END …C=10, gamma=1; total time=   2.5s
[CV] END …C=10, gamma=0.1; total time=   1.1s
[CV] END …C=10, gamma=0.1; total time=   1.0s
[CV] END …C=10, gamma=0.1; total time=   1.0s
[CV] END …C=10, gamma=0.1; total time=   1.1s
[CV] END …C=10, gamma=0.1; total time=   1.0s
[CV] END …C=10, gamma=0.01; total time=   0.8s
[CV] END …C=10, gamma=0.01; total time=   0.8s
[CV] END …C=10, gamma=0.01; total time=   0.8s
[CV] END …C=10, gamma=0.01; total time=   0.9s
[CV] END …C=10, gamma=0.01; total time=   0.8s
[CV] END …C=10, gamma=0.001; total time=   0.8s
```

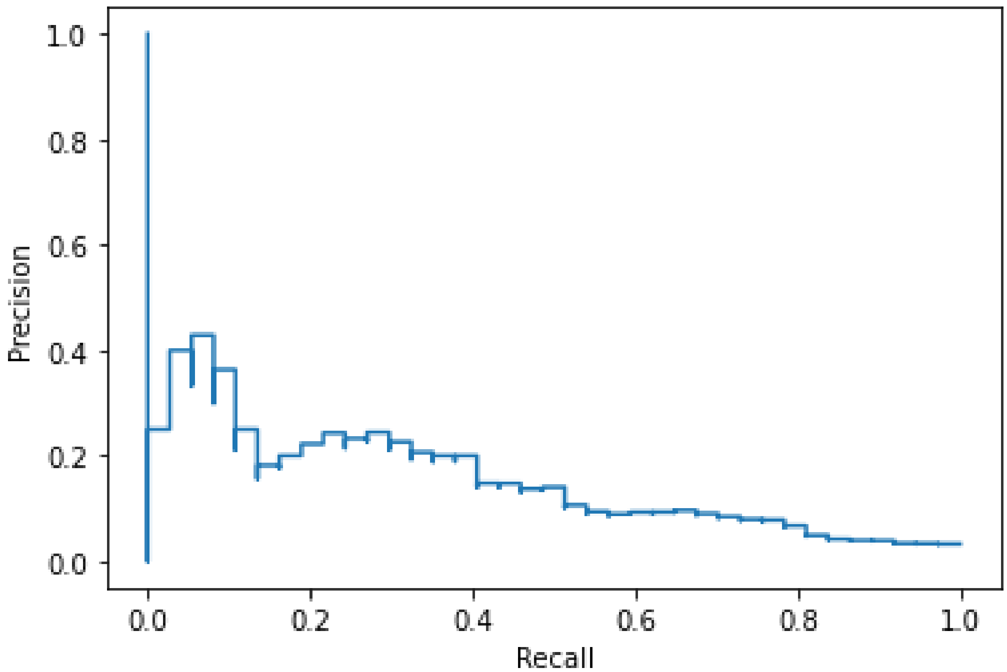
```
[CV] END …C=10, gamma=0.001; total time=   0.8s
[CV] END …C=10, gamma=0.001; total time=   0.9s
[CV] END …C=10, gamma=0.001; total time=   0.8s
[CV] END …C=10, gamma=0.001; total time=   0.8s
[CV] END …C=100, gamma=1; total time=   2.3s
[CV] END …C=100, gamma=1; total time=   2.4s
[CV] END …C=100, gamma=1; total time=   2.3s
[CV] END …C=100, gamma=1; total time=   2.4s
[CV] END …C=100, gamma=1; total time=   2.4s
[CV] END …C=100, gamma=0.1; total time=   1.2s
[CV] END …C=100, gamma=0.1; total time=   1.1s
[CV] END …C=100, gamma=0.1; total time=   1.1s
[CV] END …C=100, gamma=0.1; total time=   1.2s
[CV] END …C=100, gamma=0.1; total time=   1.1s
[CV] END …C=100, gamma=0.01; total time=   1.1s
[CV] END …C=100, gamma=0.01; total time=   1.0s
[CV] END …C=100, gamma=0.01; total time=   1.0s
[CV] END …C=100, gamma=0.01; total time=   1.2s
[CV] END …C=100, gamma=0.01; total time=   1.0s
[CV] END …C=100, gamma=0.001; total time=   0.8s
[CV] END …C=100, gamma=0.001; total time=   0.9s
[CV] END …C=100, gamma=0.001; total time=   1.1s
[CV] END …C=100, gamma=0.001; total time=   0.9s
[CV] END …C=100, gamma=0.001; total time=   0.8s
```

[65]:
```
# print optimal values for C and gamma
print(grid.best_estimator_)
```

```
SVC(C=0.1, gamma=1, probability=True)
```

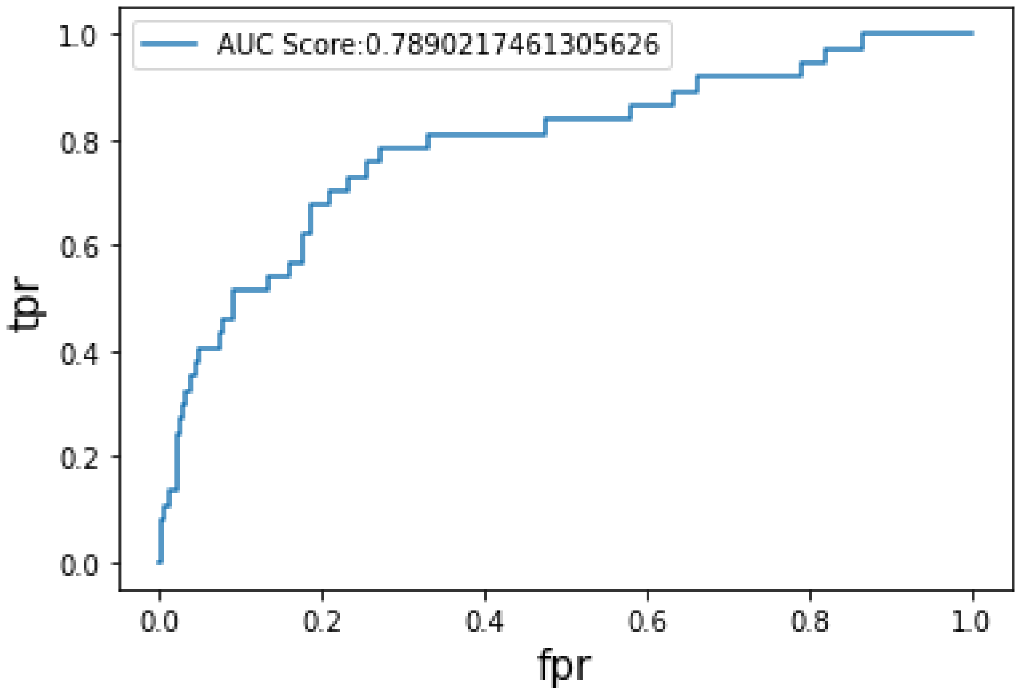
[66]:
```
y_pred = grid.predict(X_test)
y_pred_proba = grid.predict_proba(X_test)
y_score = grid.decision_function(X_test)
```

[67]:
```
#Precision/Recall
prec, recall, _ = precision_recall_curve(y_test, y_score, pos_label=grid.
 ↪classes_[1])
pr_display = PrecisionRecallDisplay(precision=prec, recall=recall).plot()
```



[68]:
```
#Plot the ROC curve
fpr, tpr, ths = roc_curve(y_test, y_pred_proba[:,1])

auc_score = auc(fpr,tpr)
plt.plot(fpr,tpr,label="AUC Score:" + str(auc_score))
plt.xlabel('fpr',fontsize='15')
plt.ylabel('tpr',fontsize='15')
plt.legend(loc='best')
```

[68]: <matplotlib.legend.Legend at 0x7ff2aada9f90>



[69]:
```
# create confusion matrix and add scores to table
cnf_matrix = confusion_matrix(y_test,y_pred)

FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)

FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
# Specificity or true negative rate
TNR = TN/(TN+FP)
# Precision or positive predictive value
PPV = TP/(TP+FP)
# Negative predictive value
NPV = TN/(TN+FN)
# Fall out or false positive rate
FPR = FP/(FP+TN)
# False negative rate
```

```
FNR = FN/(TP+FN)
# False discovery rate
FDR = FP/(TP+FP)
# Overall accuracy
ACC = (TP+TN)/(TP+FP+FN+TN)


#eval
Model = 'SVM'
TPR = [round(num, 2) for num in TPR]
FPR = [round(num, 2) for num in FPR]
precision = [round(num, 2) for num in PPV]
recall = [round(num, 2) for num in TPR]
accuracy = [round(num, 2) for num in ACC]
f1_value = round(f1_score(y_pred, y_test),2)
row = [Model, TPR, FPR, precision, recall, accuracy, f1_value]
results = results.append(pd.DataFrame([row], columns=results.columns),␣
 ↪ignore_index=True)
results
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: RuntimeWarning:
invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:21: RuntimeWarning:
invalid value encountered in true_divide
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning:
invalid value encountered in true_divide
```

[69]:

```
                                      Model           TPR           FPR  \
0                             Random Forest   [1.0, 0.14]   [0.86, 0.0]
1                    Gradient Boosted Trees  [0.98, 0.41]  [0.59, 0.02]
2  Gradient Boosted Trees with Early Stopping  [0.97, 0.49]  [0.51, 0.03]
3                               Extra Trees   [1.0, 0.16]   [0.84, 0.0]
4                                       SVM    [1.0, 0.0]    [1.0, 0.0]

      precision        recall      accuracy  f1-value
0  [0.98, 0.56]   [1.0, 0.14]  [0.97, 0.97]      0.22
1  [0.98, 0.42]  [0.98, 0.41]  [0.97, 0.97]      0.41
2  [0.98, 0.33]  [0.97, 0.49]  [0.96, 0.96]      0.39
3  [0.98, 0.86]   [1.0, 0.16]  [0.98, 0.98]      0.27
4   [0.97, nan]    [1.0, 0.0]  [0.97, 0.97]      0.00
```

3.5 Logistic Regression Model

[70]:
```
# create logistic model
log_clf = LogisticRegression(C=1.0, penalty='l2', solver='newton-cg')

log_model = log_clf.fit(X_train, y_train)
```

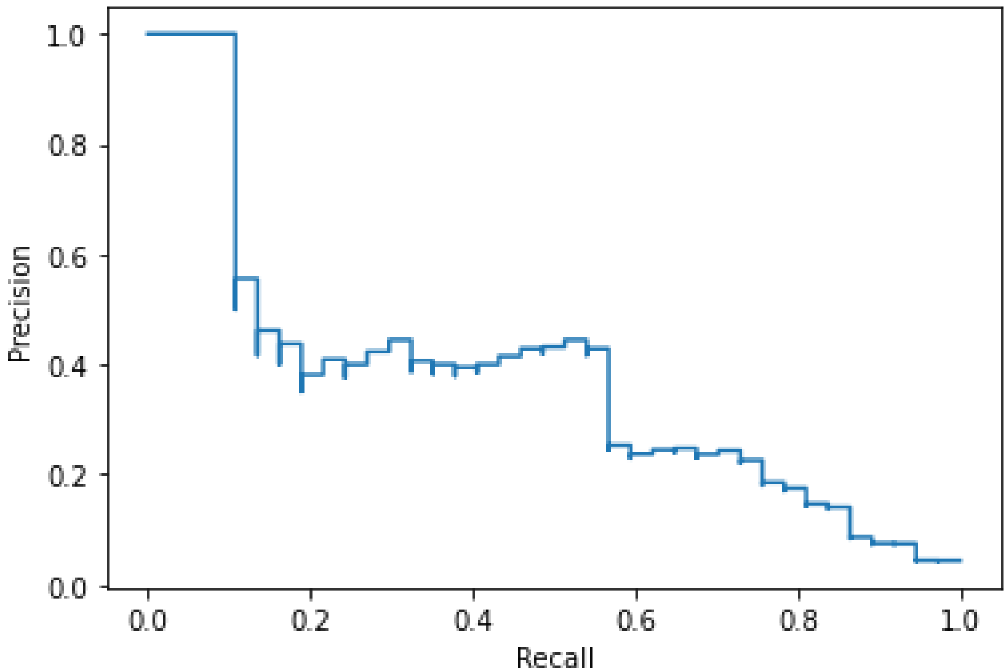
[71]:
```
# kfold validation
score = cross_val_score(log_clf, X_train, y_train, cv=5, verbose=3)
score.mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s

[CV] END … score: (test=0.969) total time=   0.1s
[CV] END … score: (test=0.970) total time=   0.1s
[CV] END … score: (test=0.976) total time=   0.1s
[CV] END … score: (test=0.974) total time=   0.1s
[CV] END … score: (test=0.972) total time=   0.1s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.3s finished
```

[71]: 0.972212500418865

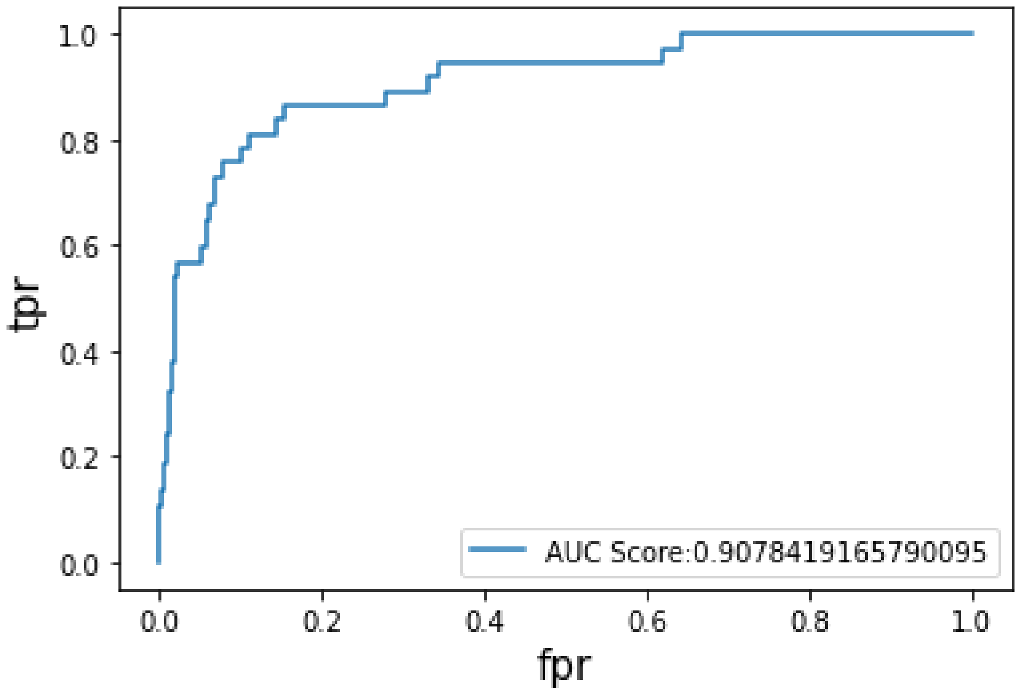
[72]:
```
y_pred = log_model.predict(X_test)
y_pred_proba = log_clf.predict_proba(X_test)
y_score = log_clf.decision_function(X_test)
```

[73]:
```
#Precision/Recall
prec, recall, _ = precision_recall_curve(y_test, y_score, pos_label=log_clf.
 ↪classes_[1])
pr_display = PrecisionRecallDisplay(precision=prec, recall=recall).plot()
```



[74]:
```
# create ROC curve
fpr, tpr, ths = roc_curve(y_test, y_pred_proba[:,1])

auc_score = auc(fpr,tpr)
plt.plot(fpr,tpr,label="AUC Score:" + str(auc_score))
plt.xlabel('fpr',fontsize='15')
plt.ylabel('tpr',fontsize='15')
plt.legend(loc='best')
```

[74]: <matplotlib.legend.Legend at 0x7ff2a84cb5d0>



[75]:
```
# create confusion matrix and add scores to table
cnf_matrix = confusion_matrix(y_test,y_pred)

FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)

FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
# Specificity or true negative rate
TNR = TN/(TN+FP)
# Precision or positive predictive value
PPV = TP/(TP+FP)
# Negative predictive value
NPV = TN/(TN+FN)
# Fall out or false positive rate
FPR = FP/(FP+TN)
# False negative rate
```

```
FNR = FN/(TP+FN)
# False discovery rate
FDR = FP/(TP+FP)
# Overall accuracy
ACC = (TP+TN)/(TP+FP+FN+TN)

#eval
Model = 'Logistic Regression'
TPR = [round(num, 2) for num in TPR]
FPR = [round(num, 2) for num in FPR]
precision = [round(num, 2) for num in PPV]
recall = [round(num, 2) for num in TPR]
accuracy = [round(num, 2) for num in ACC]
f1_value = round(f1_score(y_pred, y_test),2)
row = [Model, TPR, FPR, precision, recall, accuracy, f1_value]
results = results.append(pd.DataFrame([row], columns=results.columns),␣
 ↪ignore_index=True)
results
```

[75]:

```
                                  Model          TPR           FPR  \
0                         Random Forest  [1.0, 0.14]   [0.86, 0.0]
1                Gradient Boosted Trees [0.98, 0.41]  [0.59, 0.02]
2  Gradient Boosted Trees with Early Stopping [0.97, 0.49]  [0.51, 0.03]
3                           Extra Trees  [1.0, 0.16]   [0.84, 0.0]
4                                   SVM   [1.0, 0.0]    [1.0, 0.0]
5                   Logistic Regression  [1.0, 0.11]   [0.89, 0.0]

      precision        recall      accuracy  f1-value
0  [0.98, 0.56]   [1.0, 0.14]  [0.97, 0.97]      0.22
1  [0.98, 0.42]  [0.98, 0.41]  [0.97, 0.97]      0.41
2  [0.98, 0.33]  [0.97, 0.49]  [0.96, 0.96]      0.39
3  [0.98, 0.86]   [1.0, 0.16]  [0.98, 0.98]      0.27
4   [0.97, nan]    [1.0, 0.0]  [0.97, 0.97]      0.00
5   [0.97, 0.5]   [1.0, 0.11]  [0.97, 0.97]      0.18
```

3.6 Naïve Bayes model

[76]:

```
# create naive bayes model
nb_clf = GaussianNB()
nb_model = nb_clf.fit(X_train, np.ravel(y_train))
```

[77]:

```
# kfold cross validation
score = cross_val_score(nb_clf, X_train, y_train, cv=5, verbose=3)
score.mean()
```

```
[CV] END … score: (test=0.516) total time=   0.0s
```

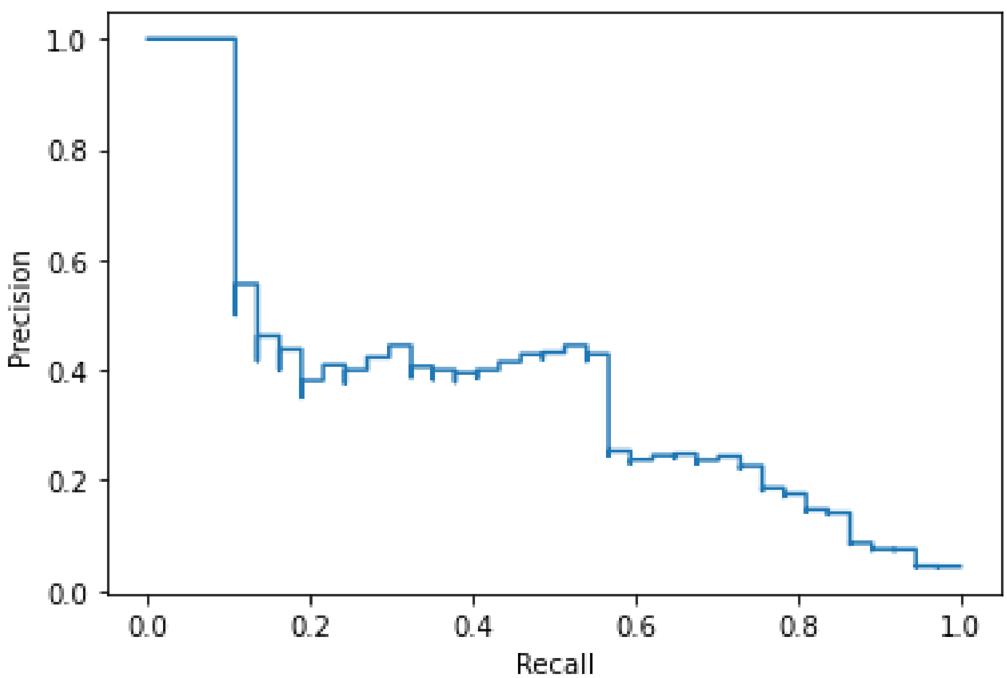
```
[CV] END … score: (test=0.892) total time=   0.0s
[CV] END … score: (test=0.871) total time=   0.0s
[CV] END … score: (test=0.940) total time=   0.0s
[CV] END … score: (test=0.872) total time=   0.0s
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
```

[77]: 0.8180679715395242

[78]:
```python
y_pred = nb_model.predict(X_test)
y_pred_proba = nb_clf.predict_proba(X_test)
```

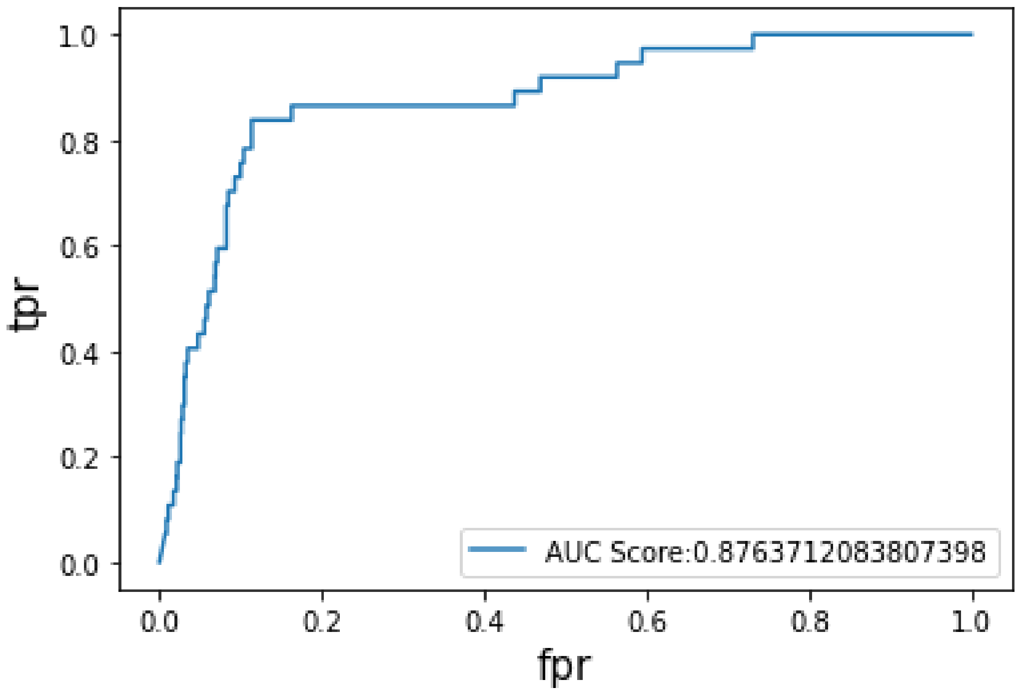
[79]:
```python
#Precision/Recall
prec, recall, _ = precision_recall_curve(y_test, y_score, pos_label=nb_clf.
 ↪classes_[1])
pr_display = PrecisionRecallDisplay(precision=prec, recall=recall).plot()
```



[80]:
```python
# create ROC curve
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score

fpr, tpr, ths = roc_curve(y_test, y_pred_proba[:,1])
```

```
auc_score = auc(fpr,tpr)
plt.plot(fpr,tpr,label="AUC Score:" + str(auc_score))
plt.xlabel('fpr',fontsize='15')
plt.ylabel('tpr',fontsize='15')
plt.legend(loc='best')
```

[80]: <matplotlib.legend.Legend at 0x7ff2a847ce10>



[81]:
```
# create confusion matrix and add scores to table
cnf_matrix = confusion_matrix(y_test,y_pred)
FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)

FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
# Specificity or true negative rate
```

```
TNR = TN/(TN+FP)
# Precision or positive predictive value
PPV = TP/(TP+FP)
# Negative predictive value
NPV = TN/(TN+FN)
# Fall out or false positive rate
FPR = FP/(FP+TN)
# False negative rate
FNR = FN/(TP+FN)
# False discovery rate
FDR = FP/(TP+FP)
# Overall accuracy
ACC = (TP+TN)/(TP+FP+FN+TN)

#eval
Model = 'naive bayes'
TPR = [round(num, 2) for num in TPR]
FPR = [round(num, 2) for num in FPR]
precision = [round(num, 2) for num in PPV]
recall = [round(num, 2) for num in TPR]
accuracy = [round(num, 2) for num in ACC]
f1_value = round(f1_score(y_pred, y_test),2)
row3 = [Model, TPR, FPR, precision, recall, accuracy, f1_value]
results = results.append(pd.DataFrame([row3], columns=results.columns),␣
 ↪ignore_index=True)
results
```

[81]:

```
                                    Model          TPR           FPR  \
0                           Random Forest   [1.0, 0.14]   [0.86, 0.0]
1                   Gradient Boosted Trees  [0.98, 0.41]  [0.59, 0.02]
2  Gradient Boosted Trees with Early Stopping  [0.97, 0.49]  [0.51, 0.03]
3                             Extra Trees   [1.0, 0.16]   [0.84, 0.0]
4                                     SVM    [1.0, 0.0]    [1.0, 0.0]
5                     Logistic Regression   [1.0, 0.11]   [0.89, 0.0]
6                             naive bayes  [0.92, 0.65]  [0.35, 0.08]

      precision        recall      accuracy  f1-value
0  [0.98, 0.56]   [1.0, 0.14]  [0.97, 0.97]      0.22
1  [0.98, 0.42]  [0.98, 0.41]  [0.97, 0.97]      0.41
2  [0.98, 0.33]  [0.97, 0.49]  [0.96, 0.96]      0.39
3  [0.98, 0.86]   [1.0, 0.16]  [0.98, 0.98]      0.27
4   [0.97, nan]    [1.0, 0.0]  [0.97, 0.97]      0.00
5    [0.97, 0.5]   [1.0, 0.11]  [0.97, 0.97]      0.18
6  [0.99, 0.19]  [0.92, 0.65]  [0.91, 0.91]      0.29
```

[ ]:

```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
```

```
colab_pdf('Module_5_Assignment_2.ipynb')
```

```
--2022-07-24 18:42:33--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.111.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: ‘colab_pdf.py’

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2022-07-24 18:42:33 (24.9 MB/s) - ‘colab_pdf.py’ saved [1864/1864]

Mounted at /content/drive/

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```