Assignment 5

February 1, 2022

1 Assignment 5: Company Bankruptcy Prediction

MSDS 422 | Kay Quiballo | 01/31/2022

1.1 Research Question

The data were collected from the Taiwan Economic Journal for the years 1999 to 2009. Company bankruptcy was defined based on the business regulations of the Taiwan Stock Exchange. The goal is to predict 'Bankrupt?' from the variables given.

1.2 Requirements

[x] Split the training set into an 80% training and 20% validation set and conduct / improve upon previous EDA. [x] Build at least three models: an SVM, a logistic regression model, a Naïve Bayes model. [x] Evaluate each of the models' assumptions. [x] Conduct hyperparameter tuning for the SVM kernel. [x] Evaluate goodness of fit metrics including TPR, FPR, precision, recall, and accuracy on the training and validation sets. [x] Build ROC and Precision / Recall graphs. [x] Evaluate your models' performance on the validation set using the F1-score. [x] Python scikit-learn should be your primary environment for conducting this research.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import StratifiedKFold
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
import imblearn
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
```

```
from matplotlib import pyplot
from numpy import where

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB

import plotly.express as px
from sklearn.metrics import roc_curve, auc, precision_recall_curve
```

Examine the data

[2]:
```
data = pd.read_csv('data.csv')
data.shape
```

[2]:
```
(6819, 96)
```

[3]:
```
#look at the data
data.head()
```

[3]:

	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin
0	1	0.370594	0.424389	0.405750	0.601457	0.601457
1	1	0.464291	0.538214	0.516730	0.610235	0.610235
2	1	0.426071	0.499019	0.472295	0.601450	0.601364
3	1	0.399844	0.451265	0.457733	0.583541	0.583541
4	1	0.465022	0.538432	0.522298	0.598783	0.598783

```
    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   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

   …   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

    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

    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

    Degree of Financial Leverage (DFL)  \
0                             0.026601
1                             0.264577
2                             0.026555
3                             0.026697
4                             0.024752

    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

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

[5 rows x 96 columns]
```

[4]:
```
#column names
data.columns = data.columns.str.strip()
list(data)
```

[4]:
```
['Bankrupt?',
 'ROA(C) before interest and depreciation before interest',
 'ROA(A) before interest and % after tax',
 'ROA(B) before interest and depreciation after tax',
 'Operating Gross Margin',
 'Realized Sales Gross Margin',
 'Operating Profit Rate',
 'Pre-tax net Interest Rate',
 'After-tax net Interest Rate',
 'Non-industry income and expenditure/revenue',
 'Continuous interest rate (after tax)',
 'Operating Expense Rate',
 'Research and development expense rate',
 'Cash flow rate',
 'Interest-bearing debt interest rate',
 'Tax rate (A)',
 'Net Value Per Share (B)',
 'Net Value Per Share (A)',
 'Net Value Per Share (C)',
 'Persistent EPS in the Last Four Seasons',
 'Cash Flow Per Share',
 'Revenue Per Share (Yuan ¥)',
 'Operating Profit Per Share (Yuan ¥)',
 'Per Share Net profit before tax (Yuan ¥)',
 'Realized Sales Gross Profit Growth Rate',
 'Operating Profit Growth Rate',
 'After-tax Net Profit Growth Rate',
 'Regular Net Profit Growth Rate',
 'Continuous Net Profit Growth Rate',
 'Total Asset Growth Rate',
 'Net Value Growth Rate',
```

```
'Total Asset Return Growth Rate Ratio',
'Cash Reinvestment %',
'Current Ratio',
'Quick Ratio',
'Interest Expense Ratio',
'Total debt/Total net worth',
'Debt ratio %',
'Net worth/Assets',
'Long-term fund suitability ratio (A)',
'Borrowing dependency',
'Contingent liabilities/Net worth',
'Operating profit/Paid-in capital',
'Net profit before tax/Paid-in capital',
'Inventory and accounts receivable/Net value',
'Total Asset Turnover',
'Accounts Receivable Turnover',
'Average Collection Days',
'Inventory Turnover Rate (times)',
'Fixed Assets Turnover Frequency',
'Net Worth Turnover Rate (times)',
'Revenue per person',
'Operating profit per person',
'Allocation rate per person',
'Working Capital to Total Assets',
'Quick Assets/Total Assets',
'Current Assets/Total Assets',
'Cash/Total Assets',
'Quick Assets/Current Liability',
'Cash/Current Liability',
'Current Liability to Assets',
'Operating Funds to Liability',
'Inventory/Working Capital',
'Inventory/Current Liability',
'Current Liabilities/Liability',
'Working Capital/Equity',
'Current Liabilities/Equity',
'Long-term Liability to Current Assets',
'Retained Earnings to Total Assets',
'Total income/Total expense',
'Total expense/Assets',
'Current Asset Turnover Rate',
'Quick Asset Turnover Rate',
'Working capitcal Turnover Rate',
'Cash Turnover Rate',
'Cash Flow to Sales',
'Fixed Assets to Assets',
'Current Liability to Liability',
```

```
 'Current Liability to Equity',
 'Equity to Long-term Liability',
 'Cash Flow to Total Assets',
 'Cash Flow to Liability',
 'CFO to Assets',
 'Cash Flow to Equity',
 'Current Liability to Current Assets',
 'Liability-Assets Flag',
 'Net Income to Total Assets',
 'Total assets to GNP price',
 'No-credit Interval',
 'Gross Profit to Sales',
 "Net Income to Stockholder's Equity",
 'Liability to Equity',
 'Degree of Financial Leverage (DFL)',
 'Interest Coverage Ratio (Interest expense to EBIT)',
 'Net Income Flag',
 'Equity to Liability']
```

[5]:
```
#check for missing data
data.isnull().sum().sort_values(ascending=False).head()
```

[5]:
```
Equity to Liability                0
Net Income Flag                    0
Operating Profit Growth Rate       0
After-tax Net Profit Growth Rate   0
Regular Net Profit Growth Rate     0
dtype: int64
```

There are no missing values.

[6]:
```
numeric_features = data.dtypes[data.dtypes != 'int64'].index

positive_corr = data[numeric_features].corrwith(data["Bankrupt?"]).
 ↪sort_values(ascending=False)[:10].index.tolist()
negative_corr = data[numeric_features].corrwith(data["Bankrupt?"]).
 ↪sort_values()[:17].index.tolist()

#positive_corr = data[positive_corr + ["Bankrupt?"]].copy()
#negative_corr = data[negative_corr + ["Bankrupt?"]].copy()

#check for collinearity
positive_corr.remove('Liability to Equity')
positive_corr.remove('Borrowing dependency')
positive_corr.remove('Current Liability to Assets')

negative_corr.remove('Net Value Per Share (B)')
```

```
negative_corr.remove('Net Value Per Share (C)')
negative_corr.remove('ROA(A) before interest and % after tax')
negative_corr.remove('ROA(B) before interest and depreciation after tax')
negative_corr.remove('ROA(C) before interest and depreciation before interest')
negative_corr.remove('Persistent EPS in the Last Four Seasons')
negative_corr.remove('Net profit before tax/Paid-in capital')
negative_corr.remove('Per Share Net profit before tax (Yuan ¥)')

X_num1 = data[positive_corr]
X_num2 = data[negative_corr]

#correlation matrix
X_num1.corr().style.background_gradient(cmap='coolwarm').set_precision(2)
X_num2.corr().style.background_gradient(cmap='coolwarm').set_precision(2)

#variable selection
var = positive_corr + negative_corr + ['Bankrupt?']
data = data[var]
data.head()
```

[6]:

	Debt ratio %	Current Liability to Current Assets	Current Liabilities/Equity	Current Liability to Equity	Total expense/Assets	Equity to Long-term Liability	Cash/Current Liability	Net Income to Total Assets	Net worth/Assets
0	0.207576	0.118250	0.339077	0.339077	0.064856	0.126549	1.473360e-04	0.716845	0.792424
1	0.171176	0.047775	0.329740	0.329740	0.025516	0.120916	1.383910e-03	0.795297	0.828824
2	0.207516	0.025346	0.334777	0.334777	0.021387	0.117922	5.340000e+09	0.774670	0.792484
3	0.151465	0.067250	0.331509	0.331509	0.024161	0.120760	1.010646e-03	0.739555	0.848535
4	0.106509	0.047725	0.330726	0.330726	0.026385	0.110933	6.804636e-04	0.795016	0.893491

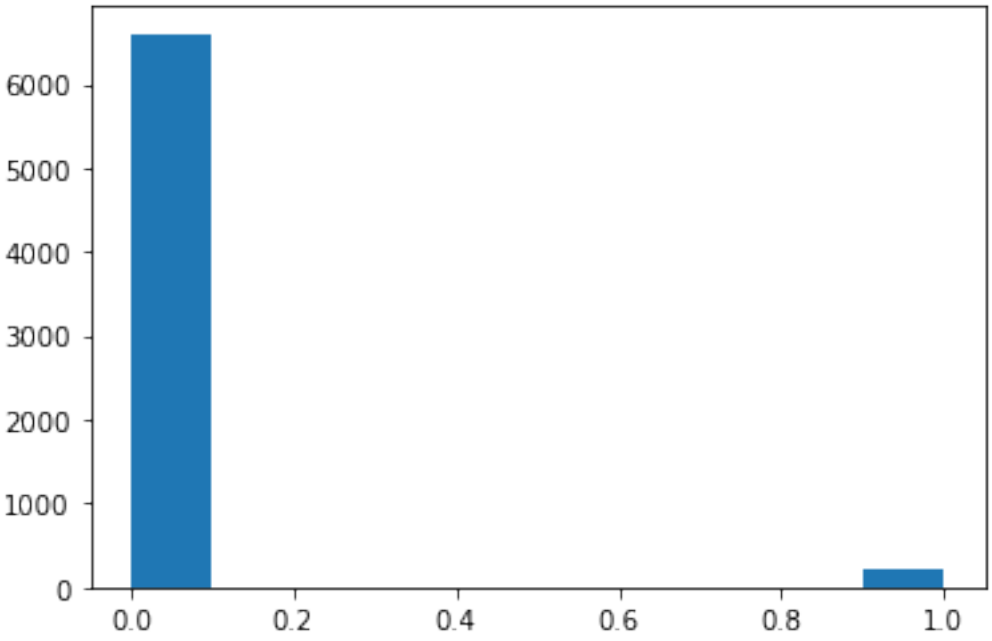
```
   Retained Earnings to Total Assets  Working Capital to Total Assets  \
0                           0.903225                         0.672775
1                           0.931065                         0.751111
2                           0.909903                         0.829502
3                           0.906902                         0.725754
4                           0.913850                         0.751822

   Net Income to Stockholder's Equity  Net Value Per Share (A)  \
0                            0.827890                 0.147950
1                            0.839969                 0.182251
2                            0.836774                 0.177911
3                            0.834697                 0.154187
4                            0.839973                 0.167502

   Working Capital/Equity  Operating Profit Per Share (Yuan ¥)  \
0                0.721275                             0.095921
1                0.731975                             0.093722
2                0.742729                             0.092338
3                0.729825                             0.077762
4                0.732000                             0.096898

   Operating profit/Paid-in capital  Bankrupt?
0                          0.095885          1
1                          0.093743          1
2                          0.092318          1
3                          0.077727          1
4                          0.096927          1
```

For model assumptions, we want to make sure the data is independent and identically distributed. One way to do this is to make sure we choose variables that aren't correlated to remove collinearity. We choose the highly correlated variables (both positive and negative) with 'Bankrupt?' but make sure that they aren't highly correlated with each other.

Split the training set into an 80% training and 20% validation set and conduct / improve upon previous EDA.

[7]:
```
plt.hist(data["Bankrupt?"])
from collections import Counter
Counter(data["Bankrupt?"])
```

[7]:
```
Counter({1: 220, 0: 6599})
```

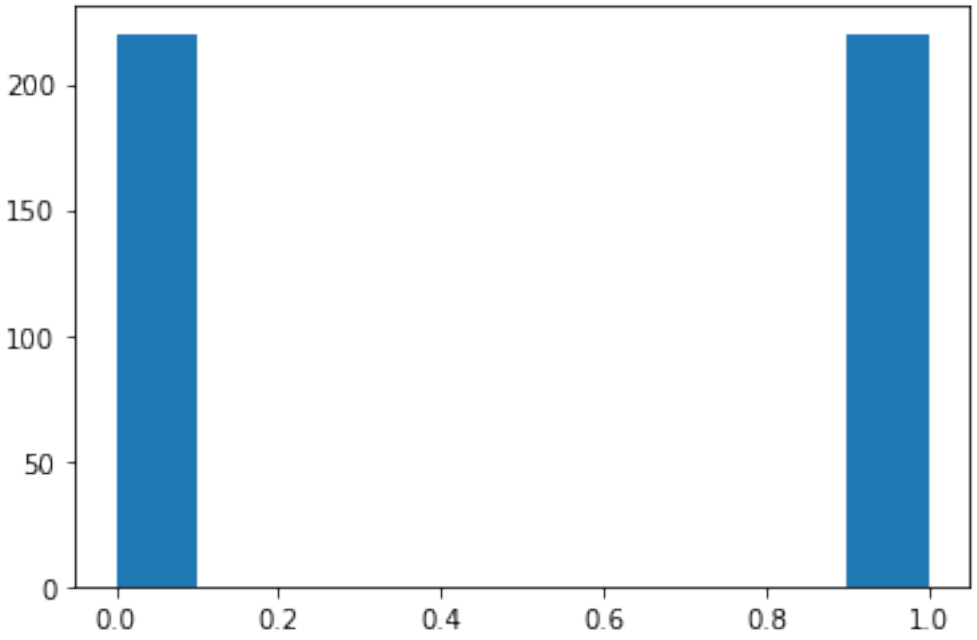


The data mostly indicates that most records do not go bankrupt. We will use stratefied sampling to make sure that this population is not overrepresented in the models. The methodology from this code is taken from Ginelle D'Souza's analysis of this data.

[8]:
```
#adjust for over/undersampling

# Shuffle the Dataset.
shuffled_df = data.sample(frac=1,random_state=4)

# Put all the fraud class in a separate dataset.
y_1 = shuffled_df.loc[shuffled_df['Bankrupt?'] == 1]

#Randomly select 492 observations from the non-fraud (majority class)
y_0 = shuffled_df.loc[shuffled_df['Bankrupt?'] == 0].
 ↪sample(n=220,random_state=42)

# Concatenate both dataframes again
normalized_df = pd.concat([y_1, y_0])
X = normalized_df.drop('Bankrupt?', axis=1)
y = normalized_df['Bankrupt?']
x_train, x_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
 ↪random_state=42, test_size=0.2)

#show new data
plt.hist(normalized_df["Bankrupt?"])
from collections import Counter
```

```
Counter(normalized_df["Bankrupt?"])
```

[8]:
```
Counter({1: 220, 0: 220})
```



Build at least three models: an SVM, a logistic regression model, a Naïve Bayes model.

[9]:
```
results = pd.DataFrame(columns = ['Model', 'TPR', 'FPR', 'precision', 'recall',␣
↪'accuracy', 'f1-score'])
```

[10]:
```
#SVM
from sklearn.metrics import f1_score

#create model
model1 = SVC(probability=True, C=1, kernel='rbf')
sm = SMOTE(sampling_strategy='minority', random_state=42)
Xsm_train, ysm_train = sm.fit_resample(x_train, y_train)
model1 = model1.fit(Xsm_train, ysm_train)
prediction = model1.predict(x_test)
print(classification_report(y_test, prediction))

#=====================================================================
cnf_matrix = confusion_matrix(y_test, model1.predict(x_test))
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_score = round(f1_score(y_test, prediction),2)
row = [Model, TPR, FPR, precision, recall, accuracy, f1_score]
results = results.append(pd.DataFrame([row], columns=results.columns),␣
 ↪ignore_index=True)
results
```

	precision	recall	f1-score	support
0	0.52	1.00	0.68	44
1	1.00	0.07	0.13	44
accuracy			0.53	88
macro avg	0.76	0.53	0.40	88
weighted avg	0.76	0.53	0.40	88

[10]:
```
  Model          TPR          FPR    precision       recall      accuracy  \
0   SVM  [1.0, 0.07]  [0.93, 0.0]  [0.52, 1.0]  [1.0, 0.07]  [0.53, 0.53]

   f1-score
0      0.13
```

[11]:
```
#logistic regression
from sklearn.metrics import f1_score

#create model
model2 = LogisticRegression(C=1.0, penalty='l2', solver='newton-cg')
sm = SMOTE(sampling_strategy='minority', random_state=42)
Xsm_train, ysm_train = sm.fit_resample(x_train, y_train)
model2 = model2.fit(Xsm_train, ysm_train)
prediction = model2.predict(x_test)
print(classification_report(y_test, prediction))

#solvers = ['newton-cg', 'lbfgs', 'liblinear']
#penalty = ['l2']
#c_values = [100, 10, 1.0, 0.1, 0.01]
# define grid search
#grid = dict(solver=solvers,penalty=penalty,C=c_values)
#cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
#grid_search = RandomizedSearchCV(estimator=model, param_grid=grid, n_jobs=-1,␣
 ↪cv=cv, scoring='accuracy',error_score=0)
#grid_result = grid_search.fit(x_train, y_train)
# summarize results
#print("Best: %f using %s" % (grid_result.best_score_, grid_result.
 ↪best_params_))
#means = grid_result.cv_results_['mean_test_score']
#stds = grid_result.cv_results_['std_test_score']
#params = grid_result.cv_results_['params']
#for mean, stdev, param in zip(means, stds, params):
#    print("%f (%f) with: %r" % (mean, stdev, param))
#Best: 0.966759 using {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}

#===========================================================================
cnf_matrix = confusion_matrix(y_test, model2.predict(x_test))
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_score = round(f1_score(y_test, prediction),2)
row = [Model, TPR, FPR, precision, recall, accuracy, f1_score]
results = results.append(pd.DataFrame([row], columns=results.columns),␣
 ↪ignore_index=True)
results
```

```
              precision    recall  f1-score   support

           0       0.89      0.89      0.89        44
           1       0.89      0.89      0.89        44

    accuracy                           0.89        88
   macro avg       0.89      0.89      0.89        88
weighted avg       0.89      0.89      0.89        88
```

[11]:
```
                 Model           TPR           FPR     precision  \
0                  SVM   [1.0, 0.07]   [0.93, 0.0]   [0.52, 1.0]
1  logistic regression  [0.89, 0.89]  [0.11, 0.11]  [0.89, 0.89]

          recall      accuracy  f1-score
0   [1.0, 0.07]  [0.53, 0.53]      0.13
```

```
1   [0.89, 0.89]   [0.89, 0.89]       0.89
```

[12]:
```
#naive bayes
from sklearn.metrics import f1_score
model3 = GaussianNB(var_smoothing=1.0)
sm = SMOTE(sampling_strategy='minority', random_state=42)
Xsm_train, ysm_train = sm.fit_resample(x_train, y_train)
model3 = model3.fit(Xsm_train, ysm_train)
prediction = model3.predict(x_test)
print(classification_report(y_test, prediction))

#params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
#cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
#gs_NB = RandomizedSearchCV(estimator=model,
#                 param_grid=params_NB,
#                 cv=cv,   # use any cross validation technique
#                 verbose=1,
#                 scoring='accuracy')
#grid_result = gs_NB.fit(x_train, y_train)
#print("Best: %f using %s" % (grid_result.best_score_, grid_result.
 ↪best_params_))
#Best: 0.969019 using {'var_smoothing': 1.0}

#============================================================================
cnf_matrix = confusion_matrix(y_test, model3.predict(x_test))
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_score = round(f1_score(y_test, prediction),2)
row3 = [Model, TPR, FPR, precision, recall, accuracy, f1_score]
results = results.append(pd.DataFrame([row3], columns=results.columns),␣
 ↪ignore_index=True)
results
```

```
              precision    recall  f1-score   support

           0       0.52      1.00      0.68        44
           1       1.00      0.07      0.13        44

    accuracy                           0.53        88
   macro avg       0.76      0.53      0.40        88
weighted avg       0.76      0.53      0.40        88
```

[12]:

	Model	TPR	FPR	precision	recall	accuracy	f1-score
0	SVM	[1.0, 0.07]	[0.93, 0.0]	[0.52, 1.0]	[1.0, 0.07]	[0.53, 0.53]	0.13
1	logistic regression	[0.89, 0.89]	[0.11, 0.11]	[0.89, 0.89]	[0.89, 0.89]	[0.89, 0.89]	0.89
2	naive bayes	[1.0, 0.07]	[0.93, 0.0]	[0.52, 1.0]	[1.0, 0.07]	[0.53, 0.53]	0.13

Analysis of F1-score: Based on the 3 models created, logistic regression does significantly better than SVM and naive bayes. With the highest value being 1 and the lowest value being 0, this shows that there is room for improvement when it comes to balancing precision and recall. When first running this analysis, the SMOTE function was not used yielding lopsided data and even lower F1-scores. Rerunning with the SMOTE function yielded higher F1-values. Let us examine the precision and recall curves to see where we can improve the models.

Build ROC and Precision / Recall graphs

[13]:
```
#SVM
y_score = model1.predict_proba(x_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_score)

fig = px.area(
    x=fpr, y=tpr,
    title=f'ROC Curve (AUC={auc(fpr, tpr):.4f})',
    labels=dict(x='False Positive Rate', y='True Positive Rate'),
    width=700, height=500
)
fig.add_shape(
    type='line', line=dict(dash='dash'),
    x0=0, x1=1, y0=0, y1=1
)

fig.update_yaxes(scaleanchor="x", scaleratio=1)
fig.update_xaxes(constrain='domain')
fig.show()

#=====================================

precision, recall, thresholds = precision_recall_curve(y_test, y_score)

fig = px.area(
    x=recall, y=precision,
    title=f'Precision-Recall Curve (AUC={auc(fpr, tpr):.4f})',
    labels=dict(x='Recall', y='Precision'),
    width=700, height=500
)
fig.add_shape(
    type='line', line=dict(dash='dash'),
    x0=0, x1=1, y0=1, y1=0
)
fig.update_yaxes(scaleanchor="x", scaleratio=1)
fig.update_xaxes(constrain='domain')

fig.show()
```

[14]:
```
#logistic regression
y_score = model2.predict_proba(x_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_score)

fig = px.area(
    x=fpr, y=tpr,
    title=f'ROC Curve (AUC={auc(fpr, tpr):.4f})',
    labels=dict(x='False Positive Rate', y='True Positive Rate'),
    width=700, height=500
```

```
)
fig.add_shape(
    type='line', line=dict(dash='dash'),
    x0=0, x1=1, y0=0, y1=1
)

fig.update_yaxes(scaleanchor="x", scaleratio=1)
fig.update_xaxes(constrain='domain')
fig.show()

#====================================

precision, recall, thresholds = precision_recall_curve(y_test, y_score)

fig = px.area(
    x=recall, y=precision,
    title=f'Precision-Recall Curve (AUC={auc(fpr, tpr):.4f})',
    labels=dict(x='Recall', y='Precision'),
    width=700, height=500
)
fig.add_shape(
    type='line', line=dict(dash='dash'),
    x0=0, x1=1, y0=1, y1=0
)
fig.update_yaxes(scaleanchor="x", scaleratio=1)
fig.update_xaxes(constrain='domain')

fig.show()
```

[15]:
```
#naive bayes
#SVM
y_score = model3.predict_proba(x_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_score)

fig = px.area(
    x=fpr, y=tpr,
    title=f'ROC Curve (AUC={auc(fpr, tpr):.4f})',
    labels=dict(x='False Positive Rate', y='True Positive Rate'),
    width=700, height=500
)
fig.add_shape(
    type='line', line=dict(dash='dash'),
    x0=0, x1=1, y0=0, y1=1
)

fig.update_yaxes(scaleanchor="x", scaleratio=1)
fig.update_xaxes(constrain='domain')
```

```
fig.show()

#====================================

precision, recall, thresholds = precision_recall_curve(y_test, y_score)

fig = px.area(
    x=recall, y=precision,
    title=f'Precision-Recall Curve (AUC={auc(fpr, tpr):.4f})',
    labels=dict(x='Recall', y='Precision'),
    width=700, height=500
)
fig.add_shape(
    type='line', line=dict(dash='dash'),
    x0=0, x1=1, y0=1, y1=0
)
fig.update_yaxes(scaleanchor="x", scaleratio=1)
fig.update_xaxes(constrain='domain')

fig.show()
```

Analysis of precision-recall curves: from all SVM and naive bayes graphs, the precision-recall curves indicate that there is high precision with a trade off of very low recall. And vice-versa, there is a high recall with a trade of of low precision. There is not quite a balance, and perhaps tuning the models using a different scoring method will improve the precision-recall graphs. On the other hand, the logistic regression does a great job of having high precision and accuracy without needing to trade off too much of one or the other.

Overall, the logistic regression model does the best job in terms of accuracy and precision to predict if the data will go bankrupt with an f-score of 0.89

1.3 Continued Analysis

[x] Conduct your analysis using a cross-validation design. [x] Conduct / improve upon previous EDA. [x] Build the following models at a minimum. - [x] Random Forest Classifier - [x] Gradient Boosted Trees - [x] Extra Trees

[x] Conduct hyperparameter tuning for the following at a minimum. - [x] n_estimators (number of trees) - [x] max_features (maximum features considered for splitting a node) - [x] max_depth (maximum number of levels in each tree) - [x] splitting criteria (entropy or gini)

[x] Compare your models using the F1-Score on a 20% validation set. Generate predictions from your models, and submit at least two models to Kaggle.com for evaluation. Provide your Kaggle.com user name and a screen snapshot of your scores.

[16]:
```
from sklearn.ensemble import RandomForestClassifier,␣
↪GradientBoostingClassifier, ExtraTreesClassifier
```

[18]:
```
#Random Forest Classifier
from sklearn.metrics import f1_score

#create model
model4 = RandomForestClassifier(n_estimators=1000, min_samples_split=2,␣
↪min_samples_leaf=5, max_features='auto', max_depth=None,␣
↪criterion='entropy', bootstrap=False)
sm = SMOTE(sampling_strategy='minority', random_state=42)
Xsm_train, ysm_train = sm.fit_resample(x_train, y_train)
model4 = model4.fit(Xsm_train, ysm_train)
prediction = model4.predict(x_test)
print(classification_report(y_test, prediction))

#========================================================================
#TUNING
#from sklearn.model_selection import RandomizedSearchCV

# Create the random grid
#random_grid = {'criterion': ["gini", "entropy"],
#               'bootstrap': [True, False],
#               'max_depth': [200, 500, 1000, 1200, 1500, 2000, None],
#               'max_features': ['auto', 'sqrt'],
#               'min_samples_leaf': [3, 4, 5, 6, 7],
#               'min_samples_split': [1, 2, 3, 4],
#               'n_estimators': [900, 1000, 1100, 1200]}

# Use the random grid to search for best hyperparameters
# First create the base model to tune
#model4 = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
#rf_random = RandomizedSearchCV(estimator = model4, param_distributions =␣
↪random_grid, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
#rf_random.fit(Xsm_train, ysm_train)
#print(rf_random.best_params_)
#{'n_estimators': 1000, 'min_samples_split': 2, 'min_samples_leaf': 5,␣
↪'max_features': 'auto', 'max_depth': None, 'criterion': 'entropy',␣
↪'bootstrap': False}

#========================================================================
cnf_matrix = confusion_matrix(y_test, model4.predict(x_test))
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 = 'Random Forest Classifier'
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_score = round(f1_score(y_test, prediction),2)
row4 = [Model, TPR, FPR, precision, recall, accuracy, f1_score]
results = results.append(pd.DataFrame([row4], columns=results.columns),␣
 ↪ignore_index=True)
results
```

```
              precision    recall  f1-score   support

           0       0.95      0.86      0.90        44
           1       0.88      0.95      0.91        44

    accuracy                           0.91        88
   macro avg       0.91      0.91      0.91        88
weighted avg       0.91      0.91      0.91        88
```

[18]:

```
                   Model           TPR           FPR     precision  \
0                    SVM   [1.0, 0.07]   [0.93, 0.0]   [0.52, 1.0]
```

```
1       logistic regression  [0.89, 0.89]  [0.11, 0.11]  [0.89, 0.89]
2               naive bayes   [1.0, 0.07]   [0.93, 0.0]   [0.52, 1.0]
3  Random Forest Classifier  [0.86, 0.95]  [0.05, 0.14]  [0.95, 0.88]

         recall      accuracy  f1-score
0   [1.0, 0.07]  [0.53, 0.53]      0.13
1  [0.89, 0.89]  [0.89, 0.89]      0.89
2   [1.0, 0.07]  [0.53, 0.53]      0.13
3  [0.86, 0.95]  [0.91, 0.91]      0.91
```

[19]:
```
#Gradient Boosted Trees
from sklearn.metrics import f1_score

#create model
model5 = GradientBoostingClassifier()
sm = SMOTE(sampling_strategy='minority', random_state=42)
Xsm_train, ysm_train = sm.fit_resample(x_train, y_train)
model5 = model5.fit(Xsm_train, ysm_train)
prediction = model5.predict(x_test)
print(classification_report(y_test, prediction))

#========================================================================
#TUNING
from sklearn.model_selection import RandomizedSearchCV

# Create the random grid
#random_grid = {'max_depth': [1, 5, 10, 20, 100, None],
#               'max_features': ["auto", "sqrt", "log2", None],
#              'min_samples_leaf': [1,3,5,10],
#               'min_samples_split': [1,3,5,10],
#               'n_estimators': [100,500,1000]}

# Use the random grid to search for best hyperparameters
# First create the base model to tune
#model5 = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
#rf_random = RandomizedSearchCV(estimator = model5, param_distributions =␣
 ↪random_grid, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
#rf_random.fit(Xsm_train, ysm_train)
#print(rf_random.best_params_)
#

#========================================================================
cnf_matrix = confusion_matrix(y_test, model5.predict(x_test))
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_score = round(f1_score(y_test, prediction),2)
row5 = [Model, TPR, FPR, precision, recall, accuracy, f1_score]
results = results.append(pd.DataFrame([row5], columns=results.columns),␣
 ↪ignore_index=True)
results
```

	precision	recall	f1-score	support
0	0.93	0.84	0.88	44
1	0.85	0.93	0.89	44
accuracy			0.89	88
macro avg	0.89	0.89	0.89	88
weighted avg	0.89	0.89	0.89	88

[19]:
```
                     Model           TPR           FPR     precision  \
0                      SVM   [1.0, 0.07]   [0.93, 0.0]   [0.52, 1.0]
1      logistic regression  [0.89, 0.89]  [0.11, 0.11]  [0.89, 0.89]
2              naive bayes   [1.0, 0.07]   [0.93, 0.0]   [0.52, 1.0]
3  Random Forest Classifier  [0.86, 0.95]  [0.05, 0.14]  [0.95, 0.88]
4    Gradient Boosted Trees  [0.84, 0.93]  [0.07, 0.16]  [0.92, 0.85]

         recall      accuracy  f1-score
0   [1.0, 0.07]  [0.53, 0.53]      0.13
1  [0.89, 0.89]  [0.89, 0.89]      0.89
2   [1.0, 0.07]  [0.53, 0.53]      0.13
3  [0.86, 0.95]  [0.91, 0.91]      0.91
4  [0.84, 0.93]  [0.89, 0.89]      0.89
```

[20]:
```
#Extra Trees
from sklearn.metrics import f1_score

#create model
model6 = ExtraTreesClassifier()
sm = SMOTE(sampling_strategy='minority', random_state=42)
Xsm_train, ysm_train = sm.fit_resample(x_train, y_train)
model6 = model6.fit(Xsm_train, ysm_train)
prediction = model6.predict(x_test)
print(classification_report(y_test, prediction))

#========================================================================
cnf_matrix = confusion_matrix(y_test, model6.predict(x_test))
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_score = round(f1_score(y_test, prediction),2)
row6 = [Model, TPR, FPR, precision, recall, accuracy, f1_score]
results = results.append(pd.DataFrame([row6], columns=results.columns),␣
 ↪ignore_index=True)
results
```

```
              precision    recall  f1-score   support

           0       0.95      0.86      0.90        44
           1       0.88      0.95      0.91        44

    accuracy                           0.91        88
   macro avg       0.91      0.91      0.91        88
weighted avg       0.91      0.91      0.91        88
```

[20]:

	Model	TPR	FPR	precision	recall	accuracy	f1-score
0	SVM	[1.0, 0.07]	[0.93, 0.0]	[0.52, 1.0]	[1.0, 0.07]	[0.53, 0.53]	0.13
1	logistic regression	[0.89, 0.89]	[0.11, 0.11]	[0.89, 0.89]	[0.89, 0.89]	[0.89, 0.89]	0.89
2	naive bayes	[1.0, 0.07]	[0.93, 0.0]	[0.52, 1.0]	[1.0, 0.07]	[0.53, 0.53]	0.13
3	Random Forest Classifier	[0.86, 0.95]	[0.05, 0.14]	[0.95, 0.88]	[0.86, 0.95]	[0.91, 0.91]	0.91
4	Gradient Boosted Trees	[0.84, 0.93]	[0.07, 0.16]	[0.92, 0.85]	[0.84, 0.93]	[0.89, 0.89]	0.89
5	Extra Trees	[0.86, 0.95]	[0.05, 0.14]	[0.95, 0.88]	[0.86, 0.95]	[0.91, 0.91]	0.91

As we look at the analysis of the additional 3 models we added this week, we see that the TPR is still ranging between 0.84 and 0.95 which is high. The FPR is also low for all three models ranging from 0.07 to 0.16. These models have a great balance of precision, recall, and accuracy which yields f1-scores of ~0.89-0.91. These scores are a significant improvement. With the help of hyperparameter tuning, these three models yield great prediction value for predciting bankruptcy from the predictors.