Case Study IV

Randy Kim, Kati Schuerger, Will Sherman
October 17, 2022

1 Introduction

Objective

The objective of this case study was to build an algorithm to predict future bankruptcy per company from historical data. Because there wasn't a data dictionary, several foundational questions occur. The most pivotal being what class was assigned as bankrupt, $b'0'$ or $b'1'$? We proceeded under the assumption that the $0'$ and $1'$ represent truth state, so $0'$ corresponded to false when querying if a company went bankrupt.

Other important questions would primarily be focused on the attributes (ATTR). Initial email replies would be as follows:

> *To whom it may concern:*
>
> *We need to identify what each of the features labeled Attr1 through Attr64 represent from your historical data. And a pivotal assessment criterion is what classes b'0' and b'1' represent… because that will dictate which way we focus the model.*

An important question of this case study was pre-processing of the data. Because the identities of the ATTRs were not provided, scaling needed to be applied with caution. However, we proceeded with the dataset scaled—treating ATTRs as independent—since some of the models under consideration are sensitive to weighted and skewed distributions.

Datasets

The dataset was provided by the Finance Department of the stakeholders, and the data structure was an attribute-relation file format (ARFF). This is essentially a comma separated file format with much richer header information. However, in this case the header fields were mostly blank.

We were able to determine the relative structure of each of the datasets provided (based on a prior study of the data)[1]. These were informative and would, likely, have been provided after follow-up communication like the above (Appendix: Table 3).

2 Methods

Data pre-processing

The data was provided in the form header-poor ARFF files in five different files that was denoting some level of yearly data. We processed the data using SciPy v1.9.2, which introduced some artifacts in the data structure. We converted the class components (*i.e.*, $b'0'$ and $b'1'$) to binary integers (*i.e.*, 0 and 1) representing *False* and *True*, respectively.

[1] Maciej Zięba, Sebastian K. Tomczak, Jakub M. Tomczak. *Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction*. Expert Systems with Applications, Volume 58, 2016, Pages 93-101, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2016.04.001.

There were a number of missing values in each of ATTRs, and after analyzing the distributions, we proceeded with imputation based on the median. As part of our data cleaning setup, we ended up dropping the thirty-seventh ATTR (Attr37) as it had 43.74% missing values. All other ATTRs had under 15% missing (Table 1).

One other consideration in the initial data was the distribution of companies that went bankrupt. The distribution was such that there were far more companies that didn't go bankrupt than went bankrupt. This is of concern for several classification model when dealing with unequal class numbers; therefore, it required sub-setting any training data with a stratification schema.

Scoring metric

We used Area Under the Receiver Operating Characteristic Curve (ROC AUC) for tuning and model evaluation. ROC AUC was optimized for all models under consideration in combination with 5-fold stratified cross-validation.

We also utilized a confusion matrix and a classification report to evaluate and compare each model. Both the matrix and the report provided measuring of Recall, Precision, and Accuracy. Depending on the models, it could be difficult to compare models with high precision and low recall or vice versa, so the classification report also provides F1-score (the harmonic mean of precision and recall) to measure them at the same time.

Classification Methods

We explored three classification methods for correctly identifying bankruptcy. SKLEARN's Logistic Regression classification was tested using ridge regularization. We also looked at Random Forest Classification which relies on many decision tree classifiers "voting" on sub-sampled data and classifying output on the max votes. The final model we explored tuning and prediction on was XGBoost Classifier's boosted tree method in combination with ridge regularization. All methods were investigated using cross-validation and randomized search (SKLearn's *RandomizedSearchCV*) for hyperparameter tuning with *mlogloss*. Additionally, posterior probability thresholds were investigated as a possible way to boost performance.

Logistic Regression

Logistic regression method was used with ridge regression which adds penalty equal to the square of the magnitude of the model's coefficients. Logistic classification can handle unbalanced classes by tuning the threshold value after predicting probabilities. This method was investigated due to the distributional bias to non-bankrupt companies.

Random Forest

The random forest model is built on decision trees which are sensitive to class imbalance. Therefore, we evaluated the model with a focus on investigating class weight parameters. Posterior probabilities were modeled as a means of optimization.

XGBoost

Extreme Gradient Boosting (XGBoost) is a boosting method which uses many weak-learners, which are decision trees, and iterates over the learned information each provides to attempt to optimize the learning objective and the loss function (L2). This is a gradient boosting method since it includes a gradient descent approach to minimizing loss.

Train/Test/Validate split

Before modeling, we randomized the arrangement of the data samples and separated the data into 2 groups: a training dataset for model fitting and a validation dataset (holdout group of 20%) that was used to measure model performance on unseen data—evaluating whether the model generalizes well. This was done with SKLearn's *train_test_split* with stratification dependent on the classes. Training data class distribution was confirmed post-split.

3 Results

While XGBoost can handle missing data, this cannot be said for Random Forest and Logistic Regression models. We opted for mean imputation on all attributes (ATTRs) and excluded only Attr37 which had 43.74% missing values which would lead to low confidence on both imputation and modeling on this feature (Table 1).

Missing Value Amounts per ATTR

	Attr	Missing Count	Missing Percentage
36	37	18984	43.74
20	21	5854	13.49
26	27	2764	6.37
59	60	2152	4.96
44	45	2147	4.95
...	...	...	...

Table 1. Top 5 features with missing values: counts and percentages.

While some other ATTRs had missingness, only Attr37 showed this level of missing information. A comparative visualization was done to assess missing values across data instances and while there was some correlation with missing values, we proceeded with imputation without discarding any of the data instances (Figure 1).

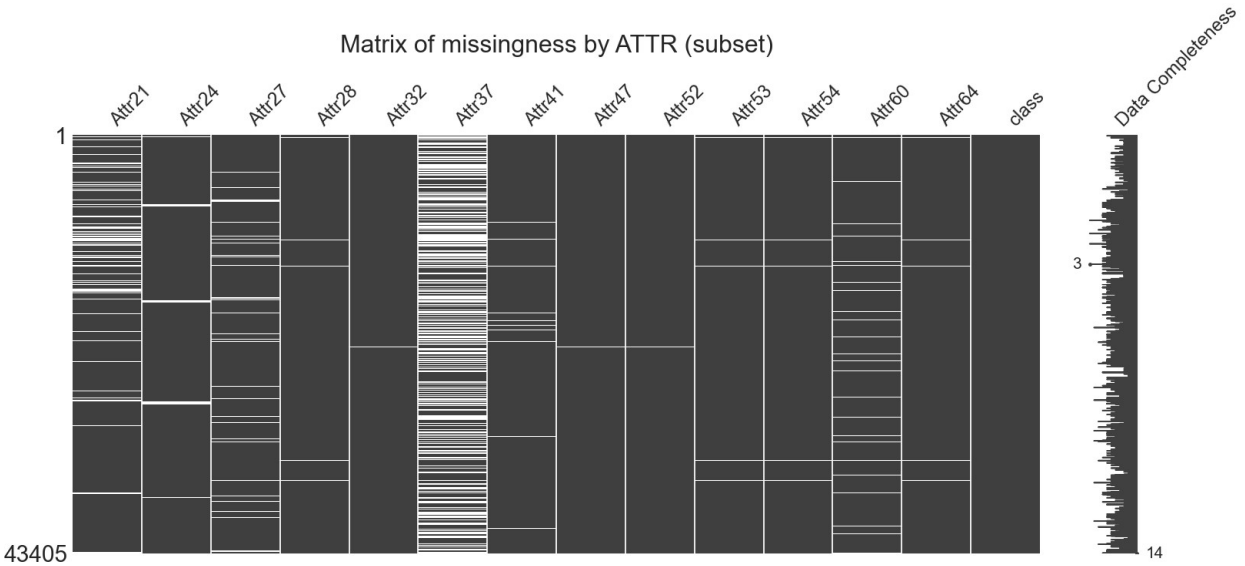

Figure 1. Plot of missingness per sample, evaluating data completeness. White space represents missingness.

Data distribution analysis was done to evaluate train-test-split and cross-validation methods. Bankrupt companies occurred in 5% of the data (Figure 2).

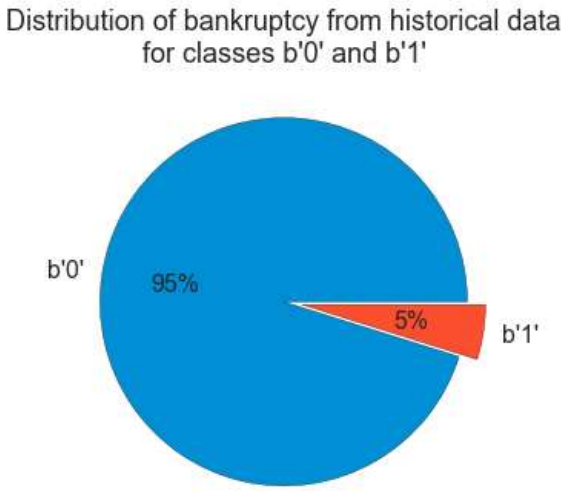

Figure 2. Pie chart distributional analysis of b'0' (not bankrupt) and b'1' (bankrupt) companies.

After splitting the data utilizing the stratify method from SKLearn's *train_test_split*(), we used SKLearn's *RandomizedSearchCV* to perform cross-validation for tuning the hyperparameters of each of the classification schema. The original number of parameter combinations for XGBoost model was 13,650 from which we subset the investigation to 10% of that at 140 values for the random search. The optimal model for XGBoost had a learning rate of 0.3, with max-depth of 3 when utilizing the *multi:softprob* objective while subsampling on a 90-10-split. The number of boosting rounds that were generated was 173 with early stopping rounds set to 10. The tuned ROC AUC on hold-out test set that we achieved was relative to the published model (Maciej, 2016) as can be seen in Table 2.

XGBoost Training Rounds

	train-auc-mean	train-auc-std	test-auc-mean	test-auc-std
0	0.775907	0.011494	0.769267	0.018012
1	0.815764	0.014627	0.804663	0.022177
2	0.831869	0.008506	0.822167	0.017355
3	0.838250	0.010841	0.827264	0.018281
4	0.855912	0.012710	0.845522	0.018306
...	...	...	...	...
169	0.996270	0.000367	0.957128	0.005671
170	0.996343	0.000366	0.957152	0.005600
171	0.996421	0.000351	0.957181	0.005518
172	0.996467	0.000367	0.957128	0.005500
173	0.996516	0.000383	0.957239	0.005419

Table 2. Training rounds iterations for XGBoost model. Training AUC mean and standard deviation can be seen in the two left-most columns. Test set AUC mean and standard deviation are the two right-most columns.

Post-training threshold values were iterated on from 0.01 through 0.99 with an expected value of 0.05 (Appendix) based on the distribution of the classes. Optimal value was found to be 0.3, which yielded an overall accuracy of 0.97. However, accuracy may be sacrificed to optimizing metrics in a class-dependent fashion.

This method was repeated and post-training thresholds were optimized for both Random Forest and Logistic Regression. Neither produced values as high as the XGBoost model (Appendix: Table 4). Again, depending on the class importance for investment strategies, thresholding may capture the class of interest—where p represents the threshold value (Figure 3).

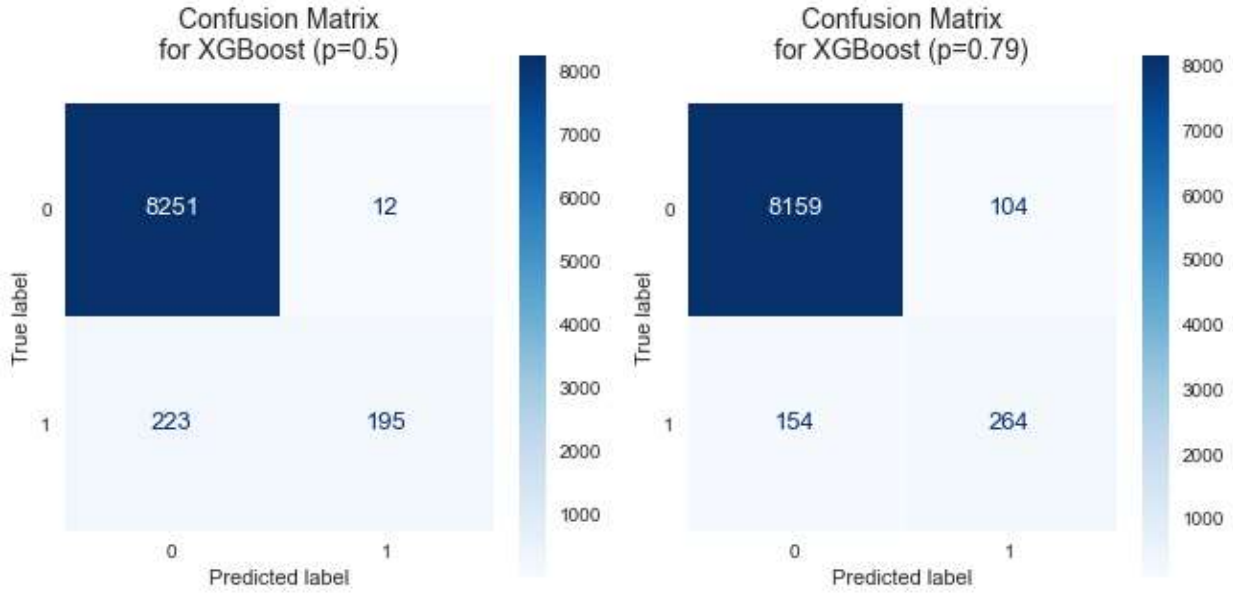

Figure 3. (a) Left. Confusion matrix with posterior threshold (p) at 0.50 shows strong classification of class 0. (b) Right. Confusion matrix with posterior threshold (p) at 0.79 had highest F1 score.

Based on the previous figure, we can sacrifice strong assignment to $b'0'$ to capture increased sensitivity for class $b'1'$.

Regardless, the optimally performing model was the tuned XGBoost model as evidenced by ROC AUC analysis (Figure 4). Final optimization can be tailored specific to risk assessment team for the Finance Department of the stakeholders.

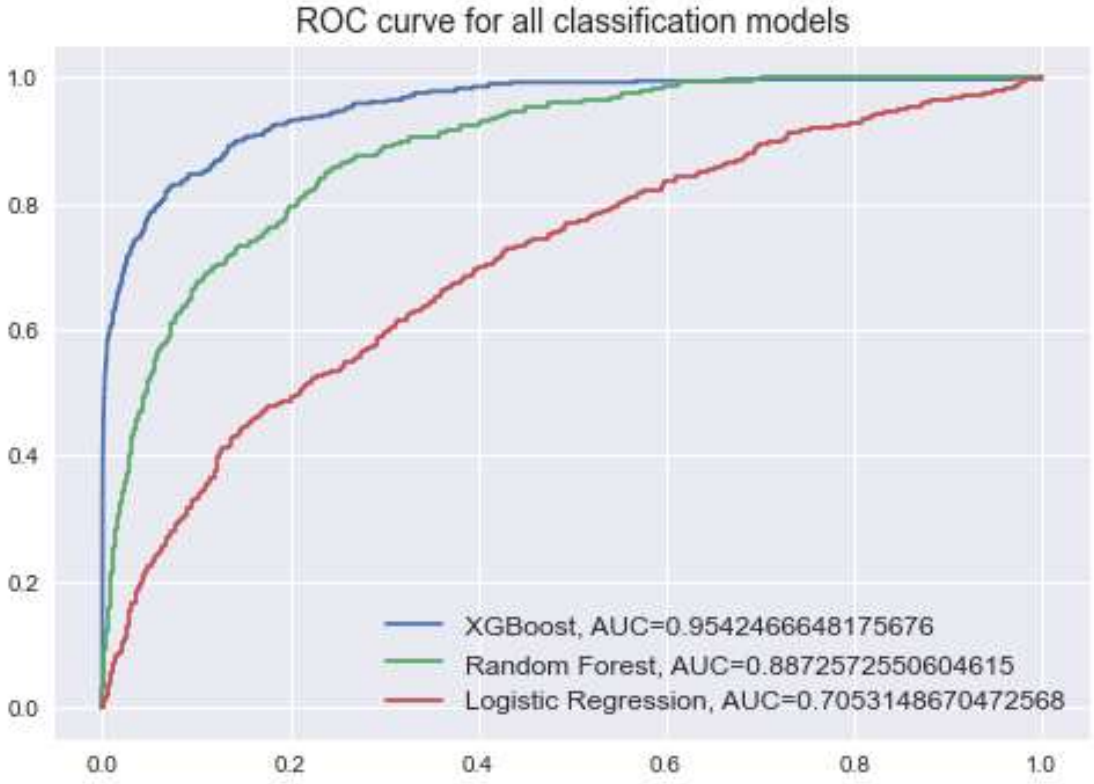

Figure 4. Area Under the Receiver Operating Characteristic Curve (ROC AUC) for all tuned models with minimum performance (Logistic Regression) of 0.705 and best performance (XGBoost) of 0.954—highest possible of 1.000.

4 Conclusion

Based on the models we looked at, we propose further tuning and posterior threshold analysis for our best performing single model: XGBoost. The hyperparameter tuning being the most intensive component, we recommend continuing analysis with our first-pass hyperparameter optimization. Although, revisiting these is an option for identification of whether we achieved a local or global minimum for our particular hyperparameters.

Further inquiry could be done if the number of bankrupt datapoints were increased from 2,091 out of 43,405 datapoints as, again, this represents only 5% of the data. There is a potential for increased model bias with this distribution of classes.

Other methods for balancing classes could include subsampling for equal sizes (information loss) or oversampling (synthetically increasing class $b'1'$).

Appendix

Table 1.

ID	Description	ID	Description
Attr1	net profit / total assets	**Attr33**	operating expenses / short-term liabilities
Attr2	total liabilities / total assets	**Attr34**	operating expenses / total liabilities
Attr3	working capital / total assets	**Attr35**	profit on sales / total assets
Attr4	current assets / short-term liabilities	**Attr36**	total sales / total assets
Attr5	[(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365,	**Attr37**	(current assets - inventories) / long-term liabilities
Attr6	retained earnings / total assets	**Attr38**	constant capital / total assets
Attr7	EBIT / total assets	**Attr39**	profit on sales / sales
Attr8	book value of equity / total liabilities	**Attr40**	(current assets - inventory - receivables) / short-term liabilities
Attr9	sales / total assets	**Attr41**	total liabilities / ((profit on operating activities + depreciation) * (12/365))
Attr10	equity / total assets	**Attr42**	profit on operating activities / sales
Attr11	(gross profit + extraordinary items + financial expenses) / total assets	**Attr43**	rotation receivables + inventory turnover in days
Attr12	gross profit / short-term liabilities	**Attr44**	(receivables * 365) / sales
Attr13	(gross profit + depreciation) / sales	**Attr45**	net profit / inventory
Attr14	(gross profit + interest) / total assets	**Attr46**	(current assets - inventory) / short-term liabilities
Attr15	(total liabilities * 365) / (gross profit + depreciation)	**Attr47**	(inventory * 365) / cost of products sold
Attr16	(gross profit + depreciation) / total liabilities	**Attr48**	EBITDA (profit on operating activities - depreciation) / total assets
Attr17	total assets / total liabilities	**Attr49**	EBITDA (profit on operating activities - depreciation) / sales
Attr18	gross profit / total assets	**Attr50**	current assets / total liabilities
Attr19	gross profit / sales	**Attr51**	short-term liabilities / total assets
Attr20	(inventory * 365) / sales	**Attr52**	(short-term liabilities * 365) / cost of products sold)

ID	Description	ID	Description
Attr21	sales (n) / sales (n-1)	**Attr53**	equity / fixed assets
Attr22	profit on operating activities / total assets	**Attr54**	constant capital / fixed assets
Attr23	net profit / sales	**Attr55**	working capital
Attr24	gross profit (in 3 years) / total assets	**Attr56**	(sales - cost of products sold) / sales
Attr25	(equity - share capital) / total assets	**Attr57**	(current assets - inventory - short-term liabilities) / (sales - gross profit - depreciation)
Attr26	(net profit + depreciation) / total liabilities	**Attr58**	total costs /total sales
Attr27	profit on operating activities / financial expenses	**Attr59**	long-term liabilities / equity
Attr28	working capital / fixed assets	**Attr60**	sales / inventory
Attr29	logarithm of total assets	**Attr61**	sales / receivables
Attr30	(total liabilities - cash) / sales	**Attr62**	(short-term liabilities *365) / sales
Attr31	(gross profit + interest) / sales	**Attr63**	sales / short-term liabilities
Attr32	(current liabilities * 365) / cost of products sold	**Attr64**	sales / fixed assets

Table 3. Depicts the features provided for analysis from the original datasets.

Expected Value:

$$\frac{count\ of\ focus\ class}{count\ of\ class: other} = \frac{2{,}091}{41{,}314} = 0.0506$$

Classification Report for All Models

XGBOOST Classification Report

	precision	recall	f1-score	support
0	0.97	1.00	0.99	8263
1	0.94	0.47	0.62	418
accuracy			0.97	8681
macro avg	0.96	0.73	0.80	8681
weighted avg	0.97	0.97	0.97	8681

Random Forest Classification Report

	precision	recall	f1-score	support
0	0.98	0.88	0.93	8263
1	0.23	0.70	0.34	418
accuracy			0.87	8681
macro avg	0.60	0.79	0.63	8681
weighted avg	0.95	0.87	0.90	8681

Logistic Regression Classification Report

	precision	recall	f1-score	support
0	0.95	1.00	0.97	8263
1	0.21	0.01	0.02	418
accuracy			0.95	8681
macro avg	0.58	0.50	0.50	8681
weighted avg	0.92	0.95	0.93	8681

Table 3. Classification Report for individual models: (a) XGBoost, (b) Random Forest, (c) Logistic Regression.

Code

```
#!/usr/bin/env python
# coding: utf-8

from scipy.io import arff
import time
import pandas as pd
import numpy as np
import os
from os.path import isfile, join
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_recall_curve, plot_precision_recall_curve, accuracy_score, confusion_matrix, average_precision_score
from sklearn.preprocessing import label_binarize, StandardScaler
from sklearn import metrics as mt
import getpass
from sklearn.model_selection import cross_validate, cross_val_score, cross_val_predict, StratifiedKFold, RandomizedSearchCV
import glob

import warnings
warnings.filterwarnings('ignore')

path = r'C:\Users\sherm\Documents\Grad School - Classes\MSDS - 7333 - Quantifying the World\Case Study 4\data'
files = [i for i in os.listdir(path)]

df_concat = pd.DataFrame()

for i in files:
    df, meta = arff.loadarff(path + '/' + i)
    df = pd.DataFrame(df)
    df_concat = df_concat.append(df, ignore_index=True)

df_concat

df_concat.describe()


warnings.filterwarnings('ignore')

for i in df_concat.columns:
    sns.distplot(df_concat.loc[:,i]).set(title=i)
    plt.show()


df.info()

df_concat.isnull().values.any()

df_concat.isnull().any()


colname = list(df_concat.columns)
rank = {}

for i in range(len(colname)):
    count = df_concat[df_concat.columns[i]].isna().sum()
    rank[i] = count
    print("Column '{col}' has {ct} NAs".format(col = colname[i], ct = count))


import missingno as msno
```

```
list_most_missing_plus_class = ['Attr21','Attr24','Attr27','Attr28','Attr32','Attr37','Attr41','Attr47','Attr52',
                                'Attr53','Attr54','Attr60','Attr64','class']

msno.matrix(df_concat[list_most_missing_plus_class], labels=True, fontsize=24)
plt.title('Matrix of missingness by ATTR (subset)\n', fontsize=32)
plt.show()


# dict(sorted(rank.items(), key=lambda item: item[1]))

column_rank = pd.DataFrame(rank.items(), columns = ['Attr', 'Missing Count'])
column_rank["Missing Percentage"] = round(column_rank["Missing Count"]/len(df_concat)*100,2)
column_rank.sort_values("Missing Count", ascending=False)


filledDF = df_concat.apply(lambda x: x.fillna(x.median()), axis = 0)
filledDF

cleanedDF = filledDF.drop(columns='Attr37')

print(cleanedDF['class'])
print(cleanedDF['class'].unique())
class_labels = cleanedDF['class'].unique()


cleanedDF['class'].value_counts()


2091/41314


plt.style.use('seaborn')
plt.rcParams.update({'font.size': 14})

fig, ax = plt.subplots()

ax.pie(cleanedDF['class'].value_counts(),
       labels=['b\'0\'','b\'1\''],
       colors=['#008fd5', '#fc4f30'],
       wedgeprops={'edgecolor': 'black'},
       autopct='%1.f%%',
       explode = (0, 0.1),
       textprops={'fontsize': 16})
ax.set_title('Distribution of bankruptcy from historical data\nfor classes b\'0\' and b\'1\'', fontsize=18)

plt.show()


cleanedDF['class'] = cleanedDF['class'].astype('str')

i = 0
for line in cleanedDF['class']:
    cleanedDF['class'][i] = line.split('\'')[1]
    i += 1

cleanedDF
```

```
# Starting Prediction Algorithms



scaler = StandardScaler()

x_data = cleanedDF.drop(columns='class')
x_scaled = scaler.fit_transform(x_data)
y_data = cleanedDF['class']


from sklearn.model_selection import train_test_split

# x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2 , stratify=y_data, random_state=63)
x_s_train, x_s_test, y_train, y_test = train_test_split(x_scaled, y_data, test_size=0.2 , stratify=y_data, random_state=63)



sns.distplot(y_train)


from sklearn.model_selection import StratifiedShuffleSplit

shuffler = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=47)


# XGBoost

y_train_vals = pd.to_numeric(y_train)
y_test_vals = pd.to_numeric(y_test)



# state_list = [1,5,10,24,48]

# dxTrain = xgb.DMatrix(x_train)
# dxTest = xgb.DMatrix(x_test)
dTrain_scaled = xgb.DMatrix(x_s_train, label=y_train_vals)
dTest_scaled = xgb.DMatrix(x_s_test, label=y_test_vals)

# xg_params = {'booster':['gbtree'],
#              'eta': [0.01,0.07,0.15,0.3,0.5,1],
#              'max_depth':[1,2,3,4,6],
#              'objective': ['multi:softprob'],
#              'min_child_weight':[1,2,3,4,6],
#              'max_delta_step':[1,2,4,6,10,15,20],
#              'subsample': [0.95],
#              'num_class': [2],
#              'scale_pos_weight':[0.04,0.06,0.08,0.1,0.46,0.48,0.50,0.52,0.54,0.9,0.92,0.94,0.96],
#              'n_estimators':[1000]}



xgb_clf = xgb.XGBClassifier()
```

```
# from sklearn.model_selection import RandomizedSearchCV
score_m = ['roc_auc']

# grid_xgb = RandomizedSearchCV(xgb_clf, param_distributions=xg_params,
#                               n_iter=120, cv=shuffler, scoring='roc_auc',
#                               n_jobs=10)

# xgb_fit = grid_xgb.fit(x_s_train, y_train)


from xgboost import DMatrix
params = {'eta':0.3,
          'max_depth':3,
          'objective':'multi:softprob',
          'min_child_weight':2,
          'max_delta_step':1,
          'subsample':0.90,
          'num_class':2,
          'eval_metric':'auc'}

from sklearn.model_selection import KFold
folder = KFold(n_splits=5, shuffle=True, random_state=47)

xgb.cv(params, dTrain_scaled, num_boost_round=1000, folds=folder, early_stopping_rounds=10)



# xgb_model = xgb_fit.best_estimator_
# print(xgb_model)
# print(xgb_fit.best_params_)
# print(xgb_fit.cv_results_)

xgb_model = xgb.XGBClassifier(**params)


xgb_model.fit(x_s_train, y_train, eval_metric='auc')
xgb_preds = xgb_model.predict(x_s_test)
xgb_pred_probs = xgb_model.predict_proba(x_s_test)


from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
cm = confusion_matrix(y_test, xgb_preds)

font = {'size'   : 13}
plt.rc('font', **font)

c_disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(5,5))
ax.grid(False)
plt.title('Confusion Matrix\nfor XGBoost (p=0.5)\n', fontsize=14)
c_disp.plot(cmap=plt.cm.Blues, ax=ax)

threshold_list = np.arange(0.01,0.99,0.01)
```

```
threshold_list


from sklearn.metrics import f1_score

labels = y_test.unique()
f1_dict = {}

for t in threshold_list:
    new_preds = (xgb_pred_probs [:,1] >= t).astype('int')
    f1_dict.update({t: f1_score(y_test_vals, new_preds)})

sorted(f1_dict.items(), key=lambda x: x[1], reverse=True)[0:10]
threshold = 1 - 0.21
predicted_xgb = (xgb_pred_probs [:,0] <= threshold).astype('int')
predicted_xgb = [str(i) for i in predicted_xgb]


from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
cm = confusion_matrix(y_test, predicted_xgb)

font = {'size'   : 13}
plt.rc('font', **font)

c_disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(5,5))
ax.grid(False)
plt.title('Confusion Matrix\nfor XGBoost (p=0.79)\n', fontsize=14)
c_disp.plot(cmap=plt.cm.Blues, ax=ax)



# RandomForest Classifier

rf_params = {'criterion':['gini'],
             'max_depth':range(2,11,2),
             'max_features':['sqrt','log2'],
             'min_impurity_decrease':np.logspace(-10,2,13),
             'class_weight':[None,'balanced','balanced_subsample'],
             'n_jobs':[10]}




rf_clf = RandomForestClassifier(n_estimators=500)
rf_clf




rf_clf.get_params().keys()

get_ipython().run_cell_magic('time', '', "\ngrid_rf = RandomizedSearchCV(rf_clf, rf_params, n_iter=40, cv=shuffler,
scoring=score_m, refit='roc_auc')\nrf_fit = grid_rf.fit(x_s_train, y_train)\n# out_rf = cross_val_predict(rf_clf)")
```

```
grid_rf
```

```
rf_model = rf_fit.best_estimator_
print(rf_model)
print(rf_fit.best_params_)
print(rf_fit.cv_results_)
```

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(rf_model, x_s_train, y_train, cv=shuffler, scoring='roc_auc')
scores
```

```
rf_model.fit(x_s_train, y_train)
preds = rf_model.predict(x_s_test)
pred_probs = rf_model.predict_proba(x_s_test)
```

```
count = 0
for i in pred_probs:
    if i[0] > 0.90:
        count += 1

print(count, 'out of', len(pred_probs))
```

```
y_test_vals = pd.to_numeric(y_test)
```

```
from sklearn.metrics import f1_score

labels = y_test.unique()
f1_dict = {}

for t in threshold_list:
    new_preds = (pred_probs [:,1] >= t).astype('int')
    f1_dict.update({t: f1_score(y_test_vals, new_preds)})

sorted(f1_dict.items(), key=lambda x: x[1], reverse=True)[0:10]
```

```
threshold = 0.52
predicted = (pred_probs [:,0] <= threshold).astype('int')
predicted = [str(i) for i in predicted]
```

```
len(predicted)
```

```
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
cm = confusion_matrix(y_test, predicted)

font = {'size': 13}
plt.rc('font', **font)

c_disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(5,5))
ax.grid(False)
plt.title('Confusion Matrix\nfor Random Forest\n', fontsize=14)
c_disp.plot(cmap=plt.cm.Blues, ax=ax)


# Logistic Regression Classification

from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression(penalty='l2')

lr_params = {'C': np.logspace(-10,10,100),
             'solver': ['newton-cf','lbfgs','sag'],
             'max_iter': range(50,200,20)}

grid_lr = RandomizedSearchCV(lr_clf, lr_params, n_iter=60, cv=shuffler, scoring=score_m, refit='roc_auc')
lr_fit = grid_lr.fit(x_s_train, y_train)

lr_model = lr_fit.best_estimator_
print(lr_model)
print(lr_fit.best_params_)
print(lr_fit.cv_results_)

lr_model = lr_model.set_params(**lr_fit.best_params_)
lr_model.fit(x_s_train, y_train)
lr_preds = lr_model.predict(x_s_test)
lr_pred_probs = lr_model.predict_proba(x_s_test)
lr_model.get_params()


f1_dict = {}

for t in threshold_list:
    new_preds = (lr_pred_probs[:,1] >= t).astype('int')
    f1_dict.update({t: f1_score(y_test_vals, new_preds)})

sorted(f1_dict.items(), key=lambda x: x[1], reverse=True)[0:10]


threshold = 0.62
lr_predicted = (lr_pred_probs [:,0] <= threshold).astype('int')
lr_predicted = [str(i) for i in predicted]


cm = confusion_matrix(y_test, lr_predicted)

font = {'size': 13}
plt.rc('font', **font)
```

```
c_disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(5,5))
ax.grid(False)
plt.title('Confusion Matrix\nfor Logistic Regression\n', fontsize=14)
c_disp.plot(cmap=plt.cm.Blues, ax=ax)



lr_pred_probs


# Graphing Composite Scores

from sklearn.metrics import roc_curve, roc_auc_score
# XGBoost
fpr, tpr, _ = roc_curve(y_test.ravel(),  xgb_pred_probs[:,1], pos_label='1')
auc = roc_auc_score(y_test, xgb_pred_probs[:,1], labels=labels)
plt.plot(fpr,tpr,label="XGBoost, AUC="+str(auc))
plt.legend(loc=4)

# Random Forest
fpr, tpr, _ = roc_curve(y_test.ravel(),  pred_probs[:,1], pos_label='1')
auc = roc_auc_score(y_test, pred_probs[:,1], labels=labels)
plt.plot(fpr,tpr,label="Random Forest, AUC="+str(auc))
plt.legend(loc=4)

# Logistic Regression
fpr, tpr, _ = roc_curve(y_test.ravel(),  lr_pred_probs[:,1], pos_label='1')
auc = roc_auc_score(y_test, lr_pred_probs[:,1], labels=labels)
plt.plot(fpr,tpr,label="Logistic Regression, AUC="+str(auc))
plt.legend(loc=4, fontsize=12)

# Plot All
plt.title('ROC curve for all classification models', fontsize=14)
plt.show()


# In[109]:


from sklearn.metrics import classification_report

print("\tXGBOOST Classification Report\n----------------------------------------------------------")
print(classification_report(y_test.tolist(), xgb_preds))
print("\tRandom Forest Classification Report\n----------------------------------------------------------")
print(classification_report(y_test.tolist(), predicted))
print("\tLogistic Regression Classification Report\n----------------------------------------------------------")
print(classification_report(y_test.tolist(), lr_preds))
```