Case Study 5: SVM / SGD

Tim Cabaza

July 8, 2024

**Abstract**

Cybe security, big company with firewall interactions.

build a program about whether or nor to accept or deny access – automate it, takes up a lot of manpower and resources. they have historical data about how they have chosen to accept and deny the requests. auto accept or auto deny the request, accurate and function at speech, 60k rows with data to play with – accuracy and performance to make decision about whether or not this can be implemented.

Action is the column we are trying to predict. a port is an address that allows different points of connection, no value other than an address, like last 4 digits of your telephone number.

The case study is an exercise in use RandomizedSearchCV to compare and beat the results of a tuned-random forest model versus a tuned XGboost Model. The goal of the classifier is to predict a binary target (1/0) of whether a business will go bankrupt to inform business decisions on divesting from said companies. The data provided is a 5 year period from 5 different business financial data and the stakeholder believes that a classifier model can accurately provide insight into future bankruptcies. Note this is not Time Series data. The following is a list of steps that are covered in the case study: Exploratory Data Analysis, Data Cleansing, Label Encoding, Normalizing, RandomForestClassifer, XGBOOST, Randomized Classifier Cross validation.

**1 Introduction**

The financial data provided is over a 5 year period from 5 different businesses.

**Table 1: Data**

|  |  |
| --- | --- |
| **Rows** | 43,405 |
| **Columns** | 65 |

**Table 2:** Missing Data

Abbreviated to show the only the first 10 columns

|  |  |
| --- | --- |
| **Column #** | **Number of Missing Values** |
| **0** | **8** |
| **1** | **8** |
| **2** | **8** |
| **3** | **134** |
| **4** | **89** |
| **5** | **8** |
| **6** | **8** |
| **7** | **94** |
| **8** | **9** |
| **9** | **8** |
| **10** | **44** |

**Table 3:** Columns with >1% Missing Values

|  |  |  |
| --- | --- | --- |
| **Column #** | **# of Missing Values** | **% Missing** |
| **20** | **5854** | **13%** |
| **23** | **922** | **2%** |
| **26** | **2764** | **6%** |
| **27** | **812** | **2%** |
| **36** | **18984** | **44%** |
| **40** | **754** | **2%** |
| **44** | **2147** | **5%** |
| **53** | **812** | **2%** |
| **59** | **2,152** | **5%** |
| **63** | **812** | **2%** |

**Table 4: Column Information**

Source: https://www.sciencedirect.com/science/article/pii/S0957417416301592

**2 Methods**

**Exploratory Data Analysis:** The data has

0 Source Port: Categorical in nature but has 22k distinct values and vast majority were an "other" category and everything else was 1% or less. To prevent the data from getting to wide the column was categorized into Rare and Common so that we can create a binary 0/1 hot encoding and not widen our dataset unnessarily.

A screenshot of a computer

Description automatically generated A screenshot of a white background

Description automatically generatedRare and (OTHER) imputed to SourceData

1 Destination Port: Categorical in nature but has ~3k distinct values. The sprad seemed to indicate 6 categories were manageable with the 5 highest having their own categories and a ‘not\_common’ category all below 5% of the data.

A white background with black text and numbers

Description automatically generated A screenshot of a graph

Description automatically generated 6 categories

2 NAT Source Port (NATS) : Categorical in nature but has ~29k distinct values. 2 categories selected

A screen shot of a graph

Description automatically generated A close-up of a white card

Description automatically generated 0, Other imputed to NATS

NAT Destination Port

Action

The rest of the variables are contious varaibles and are scaled to prevent outliers from skewing the models.

Bytes

Bytes Sent

Bytes Received

Packets

Elapsed Time

Packet Sent

Packet Received

finish going through data and target putting them into manageable categories that oyu can work with.

then you will need to one hot encode or you can try label encoding if it takes to much time

**Table 5:** Imputation

Highlighting Columns with >1% Missing Values

|  |  |
| --- | --- |
| **Imputed Column Number** | **Mean Value** |
| **20** | **3.88** |
| **23** | **0.27** |
| **26** | **1107.90** |
| **27** | **6.00** |
| **36** | **105.09** |
| **40** | **7.12** |
| **44** | **14.83** |
| **53** | **24.65** |
| **59** | **448.09** |
| **63** | **72.79** |

The target variable was dropped and the features were normalized using Scikit Learns Standard Scaler.

**Table 6:** Data Shape

|  |  |
| --- | --- |
| **Data** | **Shape** |
| Target (y) | 43405, 1 |
| Features (X) | 43405, 64 |

Two models were built and tuned to predict the target variable (bankruptcy/no bankruptcy) represented by a binary 1/0. SciKitLearn’s Random Forest Classifier and xgboost (no from SciKit Learn) were utilized to create these models. The basic process was to search over a reasonable and logical space over each models hyperparameters using RandomizedSearchCV (also from Scikit Learn). The 5 fold cross validation returned scores for Accuracy, Precision, Recall, F1 Score, and an ROC-AUC. The models were optimized for ROC-AUC to get the smoothest curve and accuracy was used to judge the overall performance.

accuracy: {'n\_estimators': 100, 'min\_weight\_fraction\_leaf': 0.0, 'min\_samples\_split': 4, 'min\_samples\_leaf': 2, 'max\_features': None, 'max\_depth': 8, 'criterion': 'gini'}

**Table 6:** Random Forest Classifier Model

|  |  |
| --- | --- |
| **Hyperparameters Tuned** | **Values/Selection** |
| n\_estimators | Testing [50], Final Model [100] (NOTE: 500 and 1000 were tried but took an excessive amount of time) |
| criterion | [‘gini’, ‘entropy’] |
| max\_depth | [None, 3, 5, 8, 10] |
| min\_samples\_split | [2, 4, 6, 8, 10] |
| min\_weight\_fraction\_leaf | [0.0, 0.1, 0.2, 0.3, 0.4] |
| min\_samples\_leaf | [2, 4, 5] |
| max\_features | [ 1, 2, 'sqrt', None] |
| min\_impurity\_decrease | 0.0 |

**Table 6:** XGBoost

|  |  |
| --- | --- |
| **Parameters** | **Values/Selection** |
| **Booster** | Gbtree |
| **Objective** | Binary:logistic |
| **Eta** | [0.1] – others were tried |
| **Min\_child\_weight** | 1 |
| **Subsample** | [0.8, 0.9] |
| **Col\_sample\_bytree** | [0.5, 0.8] |
| **Reg\_lambda** | [1,2,3] |
| **Max\_depth** | [50, 100, 500] (could have increased but the score was quite high and would most likely slightly improve) |
| **Num\_boost\_round** | 5000 |
| **Early\_stopping\_rounds** | 5 |
| **Metrics** | [logloss, auc] |

**Table 7:** Randomized Grid Search Configuration

|  |  |
| --- | --- |
| **Parameter** | **Value/Selection** |
| Estimator | Model (rf, xg) |
| Param\_distribution | Params (relative to model) |
| Scoring | [accuracy, precision, recall, roc\_auc, average\_precision] |
| Refit | roc\_auc |
| n\_jobs | -1 (all processors) |
| cv | 5 |
| random\_state | 42 |

Next Modeling was done. Random Forest is able to use cross validation and therefore it was utilized to using the Randomizedgridsearchcv from Sklearn. XGboost on the other hand does not have that capability and thus a grid search and manual changes were done to get the best hyperparameters out of the XGboost model using xgb.cv (made sure to use early stopping and a high number of boosted rounds). Once the hyperparameters and scores were saved the a regular XGBoost model was built using the hyperparameters and the scores, classification report, and ROC/AUC curve were extracted.

**3 Results**

The results were interesting as both the random forest and the xgboost model put out nearly the same AUC score with the Random Forest outperforming the XGBoost model by a few tenths (0.97 versus 0.9695). The regular xgboost model using the best hyperparameters proved to have the best accuracy in classifying bankruptcy or no bankruptcy with an accuracy score of 0.99 (not CV).

**Table 8**

**Random Forest**

|  |  |
| --- | --- |
| Best Hyperparameters for accuracy as the top metric | accuracy: {'n\_estimators': 100, 'min\_weight\_fraction\_leaf': 0.0, 'min\_samples\_split': 4, 'min\_samples\_leaf': 2, 'max\_features': None, 'max\_depth': 8, 'criterion': 'gini'} |
| Best AUC Score | 0.91 |
| Classification Report |  |
| ROC/AUC Curve |  |
| Confusion Matrix |  |

**Table 9**

**XGBoost.cv from Random Search**

|  |  |
| --- | --- |
| Best Hyperparameters with CV and random search  Best Iteration: 99  AUC Mean: 0.9695 | {'booster': 'gbtree', 'objective': 'binary:logistic', 'eta': 0.1, 'subsample': 0.5726045094068731, 'colsample\_bytree': 0.8142795758435973, 'max\_depth': 100, 'reg\_lambda': 2} |
| Regular Model with best hyperparameters Accuracy : | 0.9695 |
| LogLoss over rounds on xgb.cv |  |
| Regular Model with best hyperparameters Accuracy : | 0.9730 |
| Regular Model with best hyperparameters F1 Score : | 0.6528 |
| Regular Model with best hyperparameters Precision : | 0.9442 |
| Regular Model with best hyperparameters Recall : | 0.4989 |
| ROC/AUC Curve | A line graph with blue and orange lines  Description automatically generated |
| Classification Report: |  |
| Confusion Matrix |  |

**4 Conclusions**

The case study assessed a financial datasets of five different countries over a five year period. A Random Forest model and XGboost models were created with the tuned XGboost model outperforming the rest with an AUC score 0.99.

Overall both the Random Forest Model and the XGBoost model outperformed the models cited in the “Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction” article (<https://www.sciencedirect.com/science/article/pii/S0957417416301592>).

A table with numbers and a number of numbers

Description automatically generated

**Appendix: Code**

# Start Here.

import pandas as pd

# R-File data

# site to look at columns and their description

# https://www.sciencedirect.com/science/article/pii/S0957417416301592

# https://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_det.html#sphx-glr-auto-example-model-selection-plot-det-py

# the data starts after the first 69 rows. Hence we skip the first 69 rows when loading the data with pandas.

#

file\_names = [

'1year.arff',

'2year.arff',

'3year.arff',

'4year.arff',

'5year.arff'

]

dataframes = []

file\_path = '/Users/tmc/Desktop/MS\_SMU\_Admin/05\_2024Summer/QTW/Data\_Science/Case\_Study4/data/'

for filename in file\_names:

data = pd.read\_csv(file\_path + filename, skiprows=69, header=None)

dataframes.append(data)

dataframes[0].head()

# concatenate all the dataframes into one dataframe

all\_data = pd.concat(dataframes)

all\_data.head()

all\_data.info()

all\_data.shape

# EDA

# Transform objects to objects to numeric datatypes

# Select columns of type 'object'

object\_columns = all\_data.select\_dtypes(include=['object']).columns

# Convert selected columns to numeric

all\_data[object\_columns] = all\_data[object\_columns].apply(pd.to\_numeric, errors='coerce')

# Seperating original data from data we will impute and use for modeling

# rename all\_data to original data

data = all\_data.copy()

data.head()

# There are missing values in the dataset, we are going to have to impute and justify how we imputed the missing values

pd.set\_option('display.max\_rows', None)

if any(data.isnull().sum()):

print('There are missing values in the data')

else:

print('There are no missing values in the data')

data.isnull().sum()

#make copy of data frame to new variable all\_data

# Get all columns with missing values

columns\_with\_missing = data.columns[data.isnull().any()].tolist()

# columns\_with\_missing

for column in columns\_with\_missing:

mean\_value\_feature = data[column].mean()

data[column].fillna(mean\_value\_feature, inplace=True)

# Check if imputation was successful

if data[column].isnull().sum() == 0:

print(f"Imputed {column} with mean value {mean\_value\_feature:.2f}") # Format the mean to 2 decimal places for better readability

else:

print(f"Warning: {column} still has {all\_data[column].isnull().sum()} missing values after imputation")

data.info()

# convert target variable column 64 to name target and already one and zero so we dont need to encode it, we can just convert it to categorical

data.rename(columns={64: 'target'}, inplace=True)

# covert target variable to categorical dtype

data['target'] = data['target'].astype('category')

# check if target variable is categorical

data['target'].dtype

y = data['target']

y.head()

# Drop the 'target' column from all\_data\_scaled and assign the result to X

X = data.drop('target', axis=1)

# X

X.shape

X.columns

y.shape

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_scaled = pd.DataFrame(scaler.fit\_transform(X), columns=X.columns)

X\_scaled.head()

y.head()

# Generate the report using SweetViz package, it takes a few minutes to run but provides a comprehensive report on the data

import sweetviz as sv

report = sv.analyze(X\_scaled)

report.show\_notebook()

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import RandomizedSearchCV

import numpy as np

from sklearn.metrics import classification\_report

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, roc\_auc\_score, average\_precision\_score, make\_scorer

from sklearn.exceptions import NotFittedError

# Define the parameter distributions to sample from (Your original code)

params = {

'n\_estimators': [100],# 'n\_estimators': [50],

'criterion': ['gini', 'entropy'],

'max\_depth': [None, 3, 5, 8, 10],

'min\_samples\_split': [2, 4, 6, 8, 10],

'min\_weight\_fraction\_leaf': [0.0, 0.1, 0.2, 0.3, 0.4],

'min\_samples\_leaf': [2, 4, 5],

'max\_features': [ 1, 2, 'sqrt', None],

}

rf = RandomForestClassifier(random\_state=42)

scoring = {

'accuracy': make\_scorer(accuracy\_score),

'precision': make\_scorer(precision\_score),

'recall': make\_scorer(recall\_score),

'roc\_auc': make\_scorer(roc\_auc\_score),

'average\_precision': make\_scorer(average\_precision\_score)

}

random\_search = RandomizedSearchCV(

estimator=rf,

param\_distributions=params,

n\_iter=10,

scoring=scoring,

refit='accuracy',

n\_jobs=-1,

cv=5,

random\_state=42,

error\_score='raise',

return\_train\_score=True

)

try:

random\_search.fit(X\_scaled, y)

except NotFittedError:

print("Warning: Some hyperparameter combinations resulted in errors--skipped.")

# Get the best parameters for each metric

best\_params\_per\_metric = {}

for metric in scoring:

best\_index = np.argmax(random\_search.cv\_results\_['mean\_test\_' + metric])

best\_params\_per\_metric[metric] = random\_search.cv\_results\_['params'][best\_index]

print("\nBest Parameters per Metric:")

for metric, params in best\_params\_per\_metric.items():

print(f"- {metric}: {params}")

results\_df = pd.DataFrame(random\_search.cv\_results\_)

for metric in scoring:

mean\_score = results\_df['mean\_test\_' + metric].mean()

std\_score = results\_df['std\_test\_' + metric].mean()

print(f"\n{metric}: {mean\_score:.4f} +/- {std\_score:.4f}")

# Can we create a classification report/ confusion matrix from this not sure but try if not dont worry cause he didnt say anything about it.

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

# Get the best model based on accuracy

best\_index = np.argmax(random\_search.cv\_results\_['mean\_test\_accuracy'])

best\_params = random\_search.cv\_results\_['params'][best\_index]

# Retrain the best model on the full training set

# - accuracy: {'n\_estimators': 100, 'min\_weight\_fraction\_leaf': 0.0, 'min\_samples\_split': 4, 'min\_samples\_leaf': 2, 'max\_features': None, 'max\_depth': 8, 'criterion': 'gini'}

best\_rf\_model = RandomForestClassifier(\*\*best\_params, random\_state=42)

best\_rf\_model.fit(X\_scaled, y)

# Cross-validate the final model

from sklearn.model\_selection import cross\_val\_predict

from sklearn.metrics import classification\_report, confusion\_matrix

y\_pred = cross\_val\_predict(best\_rf\_model, X\_scaled, y, cv=5)

print("Classification Report:\n", classification\_report(y, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y, y\_pred))

y\_scores = cross\_val\_predict(best\_rf\_model, X\_scaled, y, cv=5, method="predict\_proba")[:, 1]

fpr, tpr, thresholds = roc\_curve(y, y\_scores)

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve for Best Model (Accuracy)')

plt.legend(loc="lower right")

plt.show()

ConfusionMatrixDisplay.from\_predictions(y, y\_pred, cmap='Greens')

plt.show()

## XGBOOST Random Grid Search

## Best model crossvalidated 50 estimators and 500 estimators for final comparison

import xgboost as xgb

from xgboost import XGBClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import RandomizedSearchCV

import numpy as np

from sklearn.metrics import classification\_report

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, roc\_auc\_score, average\_precision\_score, make\_scorer

from sklearn.exceptions import NotFittedError

y.shape

X\_scaled.shape

dmat\_data = xgb.DMatrix(X\_scaled, label = y.ravel()) # convert to a 1D array so that we can select the number of classes to be 2 - hence bianaary classification

# 1. Grid Search for XGBoost works !

import xgboost as xgb

import numpy as np

params = {

'booster': 'gbtree',

'objective': 'binary:logistic',

"eta": 0.1,

"subsample": None, # Placeholder to be filled in the loop

"colsample\_bytree": None, # Placeholder to be filled in the loop

"max\_depth": None, # Placeholder to be filled in the loop

'reg\_lambda': 2,

}

max\_depth = [50, 100, 500]

sub\_s = np.random.random(10)

cols = np.random.random(10)

md = np.random.randint(0, 3, 10)

results = []

for i in range(10):

temp\_params = params.copy()

temp\_params['subsample'] = sub\_s[i]

temp\_params['colsample\_bytree'] = cols[i]

temp\_params['max\_depth'] = max\_depth[md[i]]

cv\_result = xgb.cv(

temp\_params,

dmat\_data,

num\_boost\_round=1000,

nfold=5,

stratified=False,

metrics=['auc'],

early\_stopping\_rounds=5,

verbose\_eval=False,

seed=0

)

best\_iteration = cv\_result['test-auc-mean'].idxmax()

best\_auc = cv\_result['test-auc-mean'][best\_iteration]

results.append({

'params': temp\_params,

'best\_iteration': best\_iteration,

'best\_auc': best\_auc

})

best\_params = max(results, key=lambda x: x['best\_auc'])

print("\nXGBoost Grid Search Results:")

for result in results:

print("\nHyperparameters:", result['params'])

print("Best Iteration:", result['best\_iteration'])

print(f"Test AUC Mean: {result['best\_auc']:.4f}")

print("\nBest Hyperparameters:", best\_params['params'])

print(f"Best Test AUC Mean: {best\_params['best\_auc']:.4f}")

final\_cv = xgb.cv(

best\_params['params'],

dmat\_data,

num\_boost\_round=5000,

nfold=5,

stratified=False,

metrics=['auc', 'logloss'],

verbose\_eval=True,

early\_stopping\_rounds=5,

seed=0,

)

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

plt.plot(final\_cv['train-logloss-mean'], label='Train Log Loss')

plt.plot(final\_cv['test-logloss-mean'], label='Test Log Loss') #

plt.xlabel('Boosting Rounds')

plt.ylabel('Log Loss')

plt.title('Log Loss Over Boosting Rounds')

plt.legend()

plt.show()

# Model using best hyperparameters from xgboost.cv grid search

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import roc\_auc\_score, accuracy\_score, f1\_score, precision\_score, recall\_score

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# best tuned hyperparamter

model = xgb.XGBClassifier(

booster='gbtree',

objective='binary:logistic',

eta=0.1,

subsample=0.5726045094068731,

colsample\_bytree=0.8142795758435973,

max\_depth=100,

reg\_lambda=2,

n\_estimators= 99,

use\_label\_encoder=False,

eval\_metric='logloss'

)

# train

model.fit(X\_train, y\_train)

# preds

y\_pred = model.predict(X\_test)

y\_pred\_proba = model.predict\_proba(X\_test)[:, 1] # Probabilities for the positive class

roc\_auc = roc\_auc\_score(y\_test, y\_pred\_proba)

accuracy = accuracy\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

print(f"ROC-AUC: {roc\_auc:.4f}")

print(f"Accuracy: {accuracy:.4f}")

print(f"F1 Score: {f1:.4f}")

print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")

from sklearn.metrics import roc\_curve, auc

# Calculate FPR, TPR, and thresholds

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

roc\_auc = auc(fpr, tpr)

# USE PROBABILITIES FOR ROC CURVE !!!

# plot

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

cls\_report = classification\_report(y\_test, y\_pred)

print(cls\_report)

# get classificaiotn confusion matrix

from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from\_predictions(y\_test, y\_pred, cmap='Greens')

plt.show()