# Data Analysis 3: Assignment 3: Technical Report

Mohammad Ahmed

Jo Kudo

Git hub repository: https://github.com/Jk33033/DA3_hw3/tree/main

## Introduction

In this report, the main task is to build the best possible model to predict defaulted firms in the 'Manufacture of computer, electronic and optical products' industry, 2015. The prediction is only for small or medium enterprise (SME) with certain situation; they existed in 2014 but did not exists in 2015. Holdout dataset with that firms are tested in view of how this prediction is well-modeled.

## Data Preparation

The dataset was gained from the OSF website with the following link:

https://osf.io/b2ft9/?view_only=

The dataset nemed "cs_bisnode_panel" is used to make prediction. A company-year long format xt panel data table, 2005-2016 . The number of observations is 287,829, which includes 46,412 firms.

## Feature Engineering

The original data has many variables the basic characteristcs such as company ID, year, sales of a certain year, and the other specific columns such as exit year, inventories, started year etc. A few columns were dropped from the dataset after we went through the data transformation and understanding phase because they either had too many NULL values or we believed that the variables won't be useful in our analysis. The best practice while building a good model is to not have any unwanted variables and to make sure there's no memory being wasted on things that aren't being used. COGS column didn't have much in it so that was dropped and even the others. Either they weren't being used so no point of keeping them.

```
# 'columns_to_drop' is a list containing the names of the columns to be dropped
columns_to_drop = ['COGS', 'finished_prod', 'net_dom_sales','net_exp_sales','wages','D','exit_year','exit_date','begin','end','birth_year','founded_year','founded_date']
# Dropping multiple columns from the DataFrame
data.drop(columns=columns_to_drop, inplace=True)
```

For gender, origin, and region, the mode was filled in the missing value bacause they are uncountable value.

```
# 'columns_to_fill' is a list containing the names of the columns to fill missing values with mode
columns_to_fill = ['gender','origin','region_m']

# Fill missing values in specified columns with mode
for column in columns_to_fill:
    mode_value = data[column].mode()[0]  # Calculate mode for the column
    data[column].fillna(mode_value, inplace=True)    "fillna": Unknown word.
```

On the contrary, the other countable variables are filled with the mean value when it is missing.

```
# Assuming 'columns_to_fill' is a list containing the names of the columns you want to fill with the mean
columns_to_fill = ['amort', 'material_exp', 'personnel_exp','ceo_count','foreign','female','inoffice_days','labor_avg','curr_assets','curr_liab','extra_exp', 'extra_inc',
      'extra_profit_loss', 'fixed_assets', 'inc_bef_tax', 'intang_assets', 'inventories', 'liq_assets','profit_loss_year', 'sales', 'share_eq', 'subscribed_cap',    "intang"
      'tang_assets', 'nace_main', 'ind2', 'ind']    "nace": Unknown word.

# Filling missing values in multiple columns with the mean
data[columns_to_fill] = data[columns_to_fill].fillna(data[columns_to_fill].mean())    "fillna": Unknown word.
holdout_data[columns_to_fill] = holdout_data[columns_to_fill].fillna(holdout_data[columns_to_fill].mean())    "fillna": Unknown word.
```

In terms of making a holdout dataset, it is limited to a selected industry, which means ind2 == 26, and the only companies with sales between 1000 and 10,000,000 EUR in 2014. In this report, the definition of defaulted companies is the firms which had positive sales in 2014, but not in 2015.

```
# Make hold out data before fill the null with the mean value
# Filter data based on 'ind2 == 26'
# Filter data based on conditions for hold-out sample creation
holdout_data = data[data['ind2'] == 26]
holdout_data_id = holdout_data[(holdout_data['year'] == 2014) &
                               (holdout_data['sales'] >= 1000) &
                               (holdout_data['sales'] <= 10000000)]['comp_id'].unique()# group by id
holdout_data = holdout_data[holdout_data['comp_id'].isin(holdout_data_id)]#data_holdout has all years of the id
```

In order to make prediction, uncountable variables have to be eocoded.

```
from sklearn.preprocessing import LabelEncoder

# 'columns_to_encode' is a list containing the names of the columns to encode label
columns_to_encode = ['gender', 'origin', 'region_m','default']

# Iterate through each column and perform label encoding
for col in columns_to_encode:
    label_encoder = LabelEncoder()
    data[col] = label_encoder.fit_transform(data[col])
    holdout_data[col] = label_encoder.fit_transform(holdout_data[col])
```

At the end, the total number of the holdout data was 1037 and the number of defaulted firms and live firms are 56 and 981 respectedly, which is the same as the known numbers in instruction.

```
#check if there are 981 firms that stayed alive
#already filtered for existence in 2014 above
holdout_data_alive_id = holdout_data[(holdout_data['year'] == 2015)
                                     & (holdout_data['sales'] > 0 )]['comp_id'].unique()
holdout_data_default_id = holdout_data[(holdout_data['comp_id'].isin(holdout_data_alive_id)) == False]['comp_id'].unique()
```

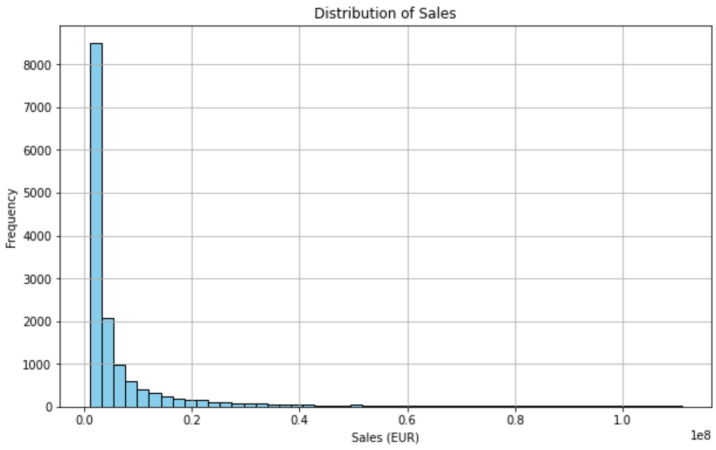
```python
total_firms = len(holdout_data_id)
defaulted_firms = len(holdout_data_default_id)
alive_firms = len(holdout_data_alive_id)

# Calculate average sales of firms
average_sales = holdout_data[(holdout_data['comp_id'].isin(holdout_data_id)) & (holdout_data['year'] == 2014)]['sales'].mean()
min_sales = holdout_data[(holdout_data['comp_id'].isin(holdout_data_id)) & (holdout_data['year'] == 2014)]['sales'].min()
max_sales = holdout_data[(holdout_data['comp_id'].isin(holdout_data_id)) & (holdout_data['year'] == 2014)]['sales'].max()
# Print the summary statistics
print(f"Total firms: {total_firms}")
print(f"Defaulted firms: {defaulted_firms}")
print(f"Alive firms: {alive_firms}")
print(f"Average sales (EUR): {average_sales:.6f}")
print(f"Minimum sales (EUR): {min_sales:.6f}")
print(f"Maximum sales (EUR): {max_sales:.6f}")
```

✓ 0.3s

```
Total firms: 1037
Defaulted firms: 56
Alive firms: 981
Average sales (EUR): 490202.217927
Minimum sales (EUR): 1070.370361
Maximum sales (EUR): 9576485.000000
```

## Exploratory Data Analysis

In order to understand the dataset dealt with in this report, the histgram of the cleaned dataset is made. According to this graph, the distribution is quite left skewed; the number of high sales immediiately drop down. Most of the sales are below 10,000,000 EUR, so it seems to be valid to make a limitaion for the holdoutset to be under that score to make sure there is less outliers.




## Modelling

Before making models, the dataset without holdout data is normalised. After that, three models are made: logistics regression, random forest, gradient boosting. In each modelling, scikit learn library was mainly used to make a fair comparison of the indicator related to the quality of prediction.

## Evaluating the Models

With the models above, following  measures are obtained:

- Brier-score, ROC curve, AUC

- Accuracy, sensitivity, specificity (for optimal threshold)
- Expected loss and optimal threshold

For brier score, Logistic Regression, Random Forest Classifier and Gradient Boosting Classifier got 0.00242, 0.00264, and 0.00252 respectively.

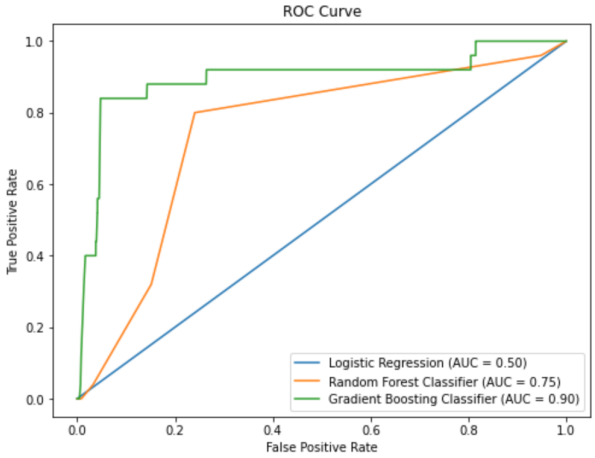
```
# Calculate Brier score for each model
logreg_brier_score = brier_score_loss(holdout_data['default'], logreg_model.predict_proba(holdout_data.drop('default', axis=1))[:,1])
rf_brier_score = brier_score_loss(holdout_data['default'], rf_model.predict_proba(holdout_data.drop('default', axis=1))[:,1])    "proba"
gb_brier_score = brier_score_loss(holdout_data['default'], gb_model.predict_proba(holdout_data.drop('default', axis=1))[:,1])    "proba"
holdout_data
print("Brier Score:")
print("Logistic Regression:", logreg_brier_score)    "logreg": Unknown word.
print("Random Forest Classifier:", rf_brier_score)
print("Gradient Boosting Classifier:", gb_brier_score)
```

✓ 0.5s

```
Brier Score:
Logistic Regression: 0.002422245906404418
Random Forest Classifier: 0.0026454413332041467
Gradient Boosting Classifier: 0.002525221987536404
```

The ROC curve and AUC is shown in the graph below.

```
# Plot ROC curves and calculate AUC for each model
def plot_roc_curve(model, X_test, y_test, name):
    fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:,1])    "proba": Unknown word.
    auc = roc_auc_score(y_test, model.predict_proba(X_test)[:,1])    "proba": Unknown word.
    plt.plot(fpr, tpr, label=name + f' (AUC = {auc:.2f})')
    plt.xlabel('False Positive Rate')    "xlabel": Unknown word.
    plt.ylabel('True Positive Rate')    "ylabel": Unknown word.
    plt.title('ROC Curve')
    plt.legend()

plt.figure(figsize=(8, 6))    "figsize": Unknown word.
plot_roc_curve(logreg_model, holdout_data.drop('default', axis=1), holdout_data['default'], 'Logistic Regression')    "
plot_roc_curve(rf_model, holdout_data.drop('default', axis=1), holdout_data['default'], 'Random Forest Classifier')
plot_roc_curve(gb_model, holdout_data.drop('default', axis=1), holdout_data['default'], 'Gradient Boosting Classifier')
plt.show()
```




Besides, Accuracy, sensitivity, specificity for optimal threshold are calculated like this.

```
# Calculate accuracy, sensitivity, and specificity for each model using optimal threshold
def calculate_metrics(y_true, y_pred):
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    sensitivity = tp / (tp + fn)
    specificity = tn / (tn + fp)
    return accuracy, sensitivity, specificity
```

✓ 0.0s

```
def calculate_optimal_threshold(model, X_test, y_test):
    thresholds = np.arange(0, 1.05, 0.05)    "arange": Unknown word.
    optimal_threshold = None
    optimal_metrics = None
    for threshold in thresholds:
        y_pred = (model.predict_proba(X_test)[:,1] >= threshold).astype(int)    "proba": Unknown word.
        accuracy, sensitivity, specificity = calculate_metrics(y_test, y_pred)
        if optimal_metrics is None or accuracy + sensitivity + specificity > sum(optimal_metrics):
            optimal_threshold = threshold
            optimal_metrics = (accuracy, sensitivity, specificity)
    return optimal_threshold, optimal_metrics
```

✓ 0.0s

```
logreg_optimal_threshold, logreg_metrics = calculate_optimal_threshold(logreg_model, holdout_data.drop('default', axis=1), holdout_data['default'])
rf_optimal_threshold, rf_metrics = calculate_optimal_threshold(rf_model, holdout_data.drop('default', axis=1), holdout_data['default'])
gb_optimal_threshold, gb_metrics = calculate_optimal_threshold(gb_model, holdout_data.drop('default', axis=1), holdout_data['default'])
```

```
Optimal Threshold and Metrics:
Logistic Regression: Threshold = 0.05 , Accuracy = 0.9975777540935956 , Sensitivity = 0.0 , Specificity = 1.0
Random Forest Classifier: Threshold = 0.35000000000000003 , Accuracy = 0.9975777540935956 , Sensitivity = 0.0 , Specificity = 1.0
Gradient Boosting Classifier: Threshold = 1.0 , Accuracy = 0.9975777540935956 , Sensitivity = 0.0 , Specificity = 1.0
```

Expected loss of the three models is **375** by chance. The expected loss is calculated using a solid equation with loss_FN and loss_FP.

```
def calculate_expected_loss(y_true, y_pred, loss_FN, loss_FP):
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    expected_loss = loss_FN * fn + loss_FP * fp
    return expected_loss
```

## Diagnostics

In terms of brier scores, they are indicative of all three models performing well, with Logistic Regression slightly outperforming the others. However, the differences in scores are minimal, suggesting that each model provides a high level of precision in its predictions. In simpler terms, the Gradient Boosting Classifier was the best at telling the difference between two groups, better than the Random Forest and much better than the Logistic Regression. This is shown by how much its ROC curve bends towards the top-left corner, meaning it has a higher chance of correctly identifying the true cases. Logistic Regression was the least accurate, as its curve was closer to a straight line, showing it struggles more to tell the groups apart. In the other indicator such as accuracy, sennsitivity, specificity, and even expected loss, the large differences are not seen, which means the quality of thes prediction can be compared mainly based on ROC curve or AUC this time.

## Conclusion

For the Brier scores, Logistic Regression does a slightly better job than the others. The scores are all very close, though, which means each model is pretty accurate. For telling

two groups apart, Gradient Boosting Classifier is the best. It's better than Random Forest and a lot better than Logistic Regression because the curve for Gradient Boosting goes more towards the top-left corner of the graph, which tells us it's good at picking out the right cases. Overall, it is possible to say that Gradient Boosting made the best contribution to prediction in this report.