# MATH 6333 CASE STUDY 2

AKUA SEKYIWAA OSEI-NKWANTABISA
Student ID: 20674356

2024-05-03

## 1 Introduction

In this study, we will concentrate on a classification problem using data from an insurance company. The data we will be working with contains information on previous mail marketing campaign responses from customers. The objective is to achieve this right target and classify clients as accurately as possible using machine learning algorithms and models, primarily Logistic Regression, Random Forest and Support Vector Machine. This will help future marketing campaigns perform better by targeting people who are more likely to accept the company's offer.

## 2 Literature Review

Support Vector Machines (SVMs) have been explored as one promising option for binary classification of companies based on solvency. SVMs work by automatically finding a dividing (hyper-)plane that separates solvent companies from insolvent ones. SVMs can have a linear decision boundary (i.e., a line above which companies are solvent and below which they are insolvent), or they can have an effectively nonlinear boundary by mapping inputs into a higher-dimensional feature space using a "kernel method" or "kernel trick" and finding a dividing hyperplane in the higher-dimensional space (which may correspond to a nonlinear boundary in the lower-dimensional space).

Salcedo-Sanz et al. (2004) used SVMs in combination with simulated annealing and Walsh analysis to perform feature selection on a set of 19 financial variables for predicting insurer insolvency. Their analysis suggests that a limited subset of just five financial ratios are sufficient to evaluate insurer solvency. Tian et al. (2019) rely on a different approach to SVMs, instead using a non-kernel fuzzy quadratic surface SVM applied to financial ratios and macroeconomic factors. This "non-kernel" method attempts to overcome limitations of kernel methods, particularly the fact that results of an SVM may be sensitive to the choice of kernel function, and no universal method exists for selecting the best kernel function.

This method achieves promising results at the cost of higher dimensionality, which may make the problem computationally expensive depending on the number of variables used.

Random forest algorithms have been used to predict insolvency within the insurance industry (Kartasheva et al, 2013) and in more general business settings.(Behr et al, 2016) Compared to SVMs, random forests can handle binary data without additional assumptions, whereas SVMs require the user to specify a notion of "distance" between the binary data points.

Random forests can be designed to provide probability of insurer default rather than a binary classification of "default" or "healthy." Random forests can also be used to automatically rank the importance of variables to determining insurer solvency. They have an advantage over statistical approaches like logistic regression in their ability to automatically detect and model highly non-linear relationships. However, random forests may be challenging to interpret, and may produce more highly variable results, particularly with sparse data sets.

Montserrat Guillen et al. (2002) paper focus on the insurance industry customer. They started by defining a customer as someone having one or several insurance policies in several lines of insurance. They reviewed some of the marketing literature concerning repurchase behavior. A basic logistic regression modeling was presented which was used in insurance fraud detection.

The Logistic Regression,Random Forest and Support Vector machine approaches will also be used in this study, which will use data from the marketing insurance industry with information on customer responses to a direct mail marketing campaign. The goal is to use the best model available to find the target audience most likely to accept the offer.

# 3 Methods

The data was analyzed using three machine learning algorithms namely Logistic Regression,Random Forest and Support Vector machine to help solve the classification problem. Python programming language was the main tool used to analyze the dataset. The data set constitute 68 predictive variables and 20000 records. The data set was split into two parts for modeling and validation purposes. The 10000 was used for training of which was used to estimate model while the other 10000 was used for testing. The model's success was based on its ability to predict the probability that customer will take the offer. Evaluating the performance of a model is one of the crucial steps in its development. So evaluation measures, sometimes referred to as performance metrics, were employed. They are the Confusion Matrix ,Precision,Classification Accuracy,Recall/Sensitivity and F1 –Score/F Measure. The ROC curve was found and used to evaluate the models performance.

Specificity is a metric that calculates the ratio of true negatives to the total number of negatives. It's formula is given as :

$$\text{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$$

Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances.It's formula is given as :

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Accuracy (classification accuracy or overall correctness), defined as proportion of correct predictions (both true positives and true negatives) among all cases examined. It's formula is given as :

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Observations}}$$

Recall (also known as sensitivity or hit rate) is a measure of how many truly positive results were returned out of all possible positives. It quantifies an algorithm's ability to find all the relevant cases within a dataset. It's formula is given as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

The F1 Score is a statistical measure used in assessing binary classification systems and models created based on them. It combines precision and recall into one metric, providing a balance between the two when there are uneven class distributions. It's formula is given as :

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The confusion matrix has 4 components that are explained below :

**True Positives (TP):** Located in the top left corner. These are cases where the model correctly predicts that a customer will accept the offer

**False Negatives (FN):** Located in the top right corner. These occur when the model incorrectly predicts that a customer will reject the offer

**False Positives (FP):** Located in the bottom left corner. These occur when the model incorrectly predicts that a customer will accept the offer

**True Negatives (TN):** Located in the bottom right corner. These are cases where the model correctly predicts that a customer will reject the offer

A ROC Curve is a chart which represents the performance of binary classifier system in different thresholds. It does this by plotting False positive rate (FPR) against True positive rate (TPR). AUC is the area under the ROC curve and it summarizes how good a model is at separating classes. An increased AUC shows improved performance by the model.

# 4 Analysis Findings

The analysis began by downloading the data and uploading it unto colab. The dimensions of the train dataset was 10000 by 69 while the test data set was 10000 by 70. The count of people that accepted the offer per the training data were 2023 and 7977 for those who did not accept the offer.There were no null values

## 4.1 Information Value

Information Value was calculated for each feature to evaluate their predictive power of in relation the binary outcome. In order to get a better model, we had to restrict the features to the ones that are estimated to have a high predictive power.

```
0              M_SNC_MST_RCNT_ACT_OPN
0                TOT_HI_CRDT_CRDT_LMT
0                RATIO_BAL_TO_HI_CRDT
0             AGRGT_BAL_ALL_XCLD_MRTG
0           N_OF_SATISFY_FNC_REV_ACTS
0            AVG_BAL_ALL_FNC_REV_ACTS
0          M_SNCOLDST_BNKINSTL_ACTOPN
0          M_SNC_MSTREC_INSTL_TRD_OPN
0            M_SNC_OLDST_MRTG_ACT_OPN
0                N_BC_ACTS_OPN_IN_24M
0             AVG_BAL_ALL_PRM_BC_ACTS
0          M_SNC_OLDST_RETAIL_ACT_OPN
0            RATIO_RETAIL_BAL2HI_CRDT
0           PRCNT_OF_ACTS_NEVER_DLQNT
0                      N_OPEN_REV_ACTS
0                  HI_RETAIL_CRDT_LMT
0         D_NA_M_SNC_MST_RCNT_ACT_OPN
0       D_NA_AVG_BAL_ALL_FNC_REV_ACTS
0     D_NA_M_SNCOLDST_BNKINSTL_ACTOPN
0       D_NA_M_SNC_OLDST_MRTG_ACT_OPN
Name: Variable, dtype: object
(20,)
```

To achieve this, all variables that had an Information Value less than 0.1 were removed and we had a new training and testing data sets for our analysis. Above is the list of features that had an Information Value greater than 0.1.

## 4.2 Logistic Regression

A baseline Logistic Regression model was built using the new datasets that were generated after filtering out the features with a weak Information Value. After the baseline model was built, we explored different strategies of coming up with an optimal model. In logistic regression, deviance is a measure of how well the model fits the observed data. Lower deviance indicates a better fit. Deviance is calculated by comparing the model's predicted probabilities to the actual outcomes.

The main criteria for the optimal model was one that had the lowest deviance. However, hyperparameter tuning with GridSearchCV was also employed to achieve a better performing model. We started by defining a parameter grid with different values for 'penalty', 'C', and 'solver'. We had to track the deviance so we created a variable for it. We then set it with 5-fold cross-validation in order to allow parallel processing to utilize all available CPU cores.

The model with the lowest deviance score and the highest accuracy emerged the best logistic regression model.

After fitting the model, the learning accuracy of the Logistic Regression Model was **80%** with the deviance at **0.4273**. The 80% Learning Accuracy and deviance of 0.4273 means that the chosen predictor variables can predict the target variable using the training data, which implies these predictors are capturing significant patterns or in the data. Nevertheless, to guard against overfitting we must check it with unseen information before any such assertion is confirmed.

The model was then evaluated on the test data and the Specificity and Deviance calculated:

**Specificity:** 0.17635370094386488

**Deviance :** 0.4102

As well as generating the classification report, confusion matrix, ROC Curve and its AUC as seen below:

Table 1: Classification Report for Logistic Regression

| | **precision** | **recall** | **f1-score** | **support** |
|---|---|---|---|---|
| 0 | 0.82 | 0.98 | 0.89 | 7987 |
| 1 | 0.67 | 0.18 | 0.28 | 2013 |
| accuracy | | | 0.82 | 10000 |
| macro avg | 0.75 | 0.58 | 0.59 | 10000 |
| weighted avg | 0.79 | 0.82 | 0.77 | 10000 |

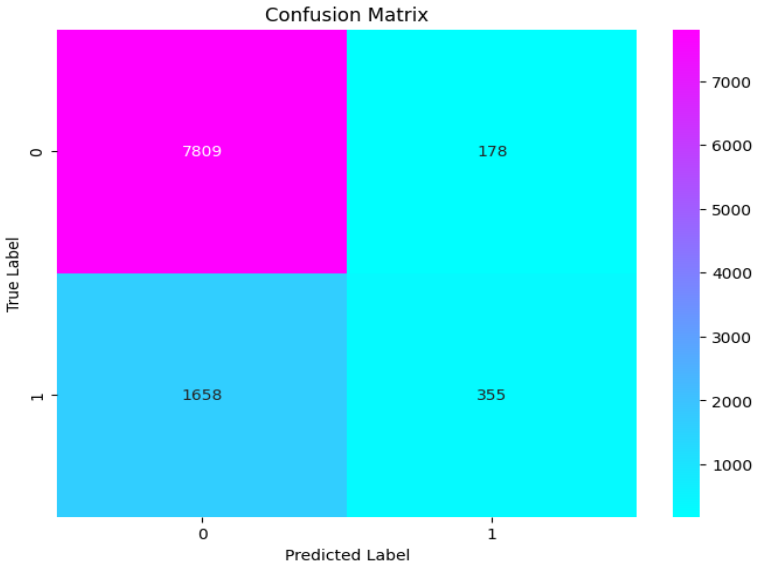


Figure 1: Confusion Matrix for Logistic Regression
This shows that the True Positives were 7809. False Negatives and False Positives were 178 and 1658 respectively. The True Negatives were 355.

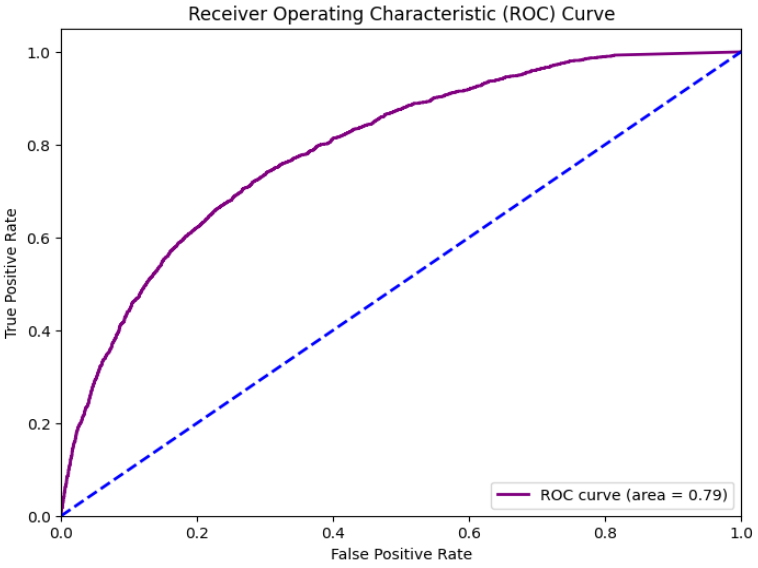


Figure 2: Showing the ROC curve and the Area under Curve.
The Area under the curve for this model is given as 0.79.

## 4.3 Random Forest

The Random Forest model was built using 10001 trees. Predictor variables ranging from 1 to 13 were used in order to find the optimal number of predictor variables for the best performing model.

The main criteria for the selection of the best model was based on the Out of Bag (OOB) Error. The mode with the lowest OOB error was selected as the best model.

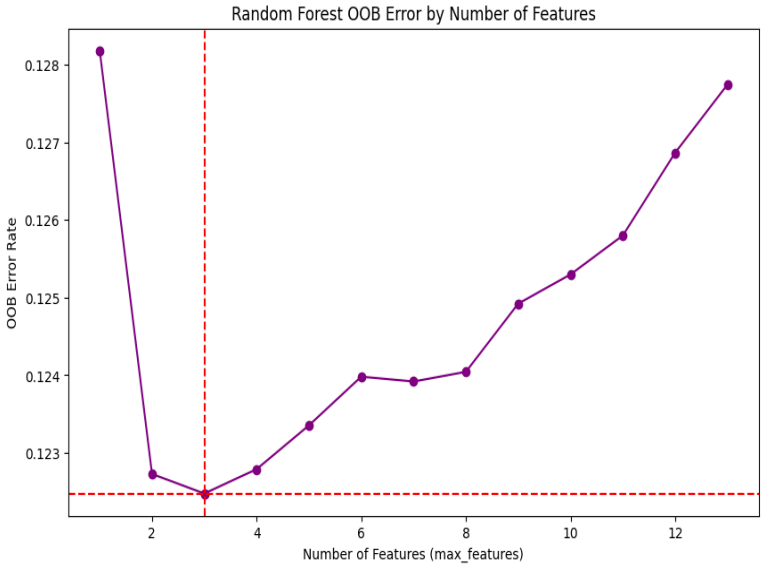


Figure 3: Graph to identify the lowest OOB error

Once again, we evaluated the model by calculating the Specificity:

**Specificity:** 0.5489319423745653

As well as generating the classification report, confusion matrix, ROC Curve and its AUC as seen below:

Table 2: Classification Report for Random Forest

| | **precision** | **recall** | **f1-score** | **support** |
|---|---|---|---|---|
| 0 | 0.87 | 0.77 | 0.82 | 7987 |
| 1 | 0.38 | 0.55 | 0.45 | 2013 |
| accuracy | | | 0.73 | 10000 |
| macro avg | 0.62 | 0.66 | 0.63 | 10000 |
| weighted avg | 0.77 | 0.73 | 0.74 | 10000 |

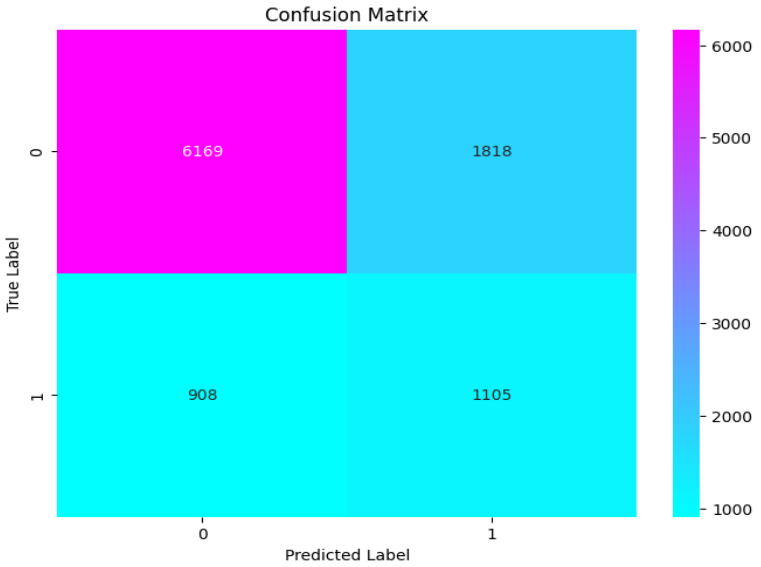


Figure 4: Confusion Matrix for Random Forest
This shows that the True Positives were 6169. False Negatives and False Positives were 1818 and 908 respectively. The True Negatives were 1105.

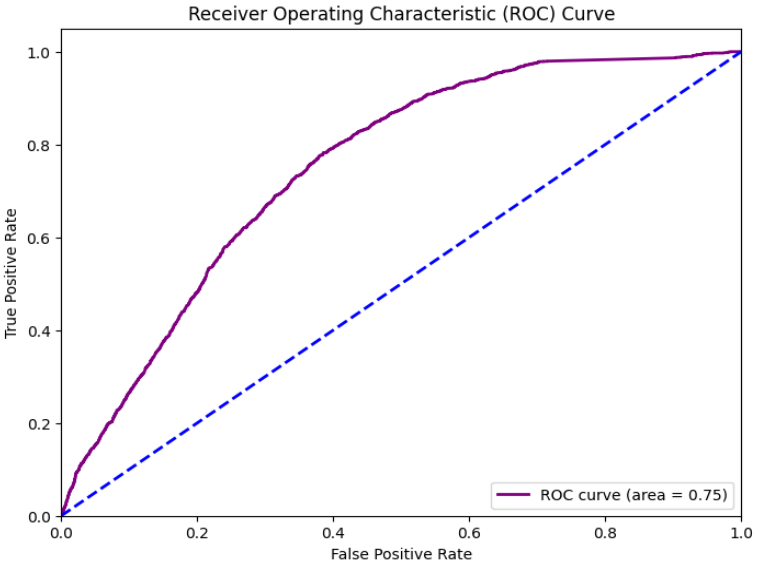


Figure 5: ROC Curve for Random Forest
The Area under the curve for this model is given as 0.75

## 4.4 Support Vector Machine

When it came to the Support vector Machine (SVM), two models were built. One using a polynomial kernel with degree 3 and a cost value of 0.01. The other model was built using the Gaussian radial kernel also with a cost value of 0.01 and gamma as 0.000001.

While developing our classification model, we encountered a warning for the recall, F1-score, and precision metrics.The warning showed that these metrics were set to 0.0 for labels with no predicted samples. This implies a class imbalance problem where some classes are not represented sufficiently, resulting in them not being predicted at all.

To solve the issues of class imbalance in the dataset, we used SMOTE (Synthetic Minority Over-sampling Technique). It is an advanced over-sampling technique that creates artificial instances from the under-represented category to equalize it with others. This can greatly improve classification algorithms such as SVMs by giving them a balanced dataset – this way there won't be any prejudice towards the majority class and models will be better able to identify patterns from less populated classes in general.

### 4.4.1 Support Vector Machine Baseline Models

Below is the classification report for the SVM using the Polynomial Kernel and Gaussian Radial(RBF) Kernel before we proceeded to use SMOTE. As you can see from the classfication reports, there were imbalances in the dataset that triggered the warning for our SVM models.

Table 3: Classification Report for Polynomial Kernel

| | **precision** | **recall** | **f1-score** | **support** |
|---|---|---|---|---|
| 0 | 0.80 | 1.00 | 0.89 | 7987 |
| 1 | 0.73 | 0.02 | 0.03 | 2013 |
| accuracy | | | 0.80 | 10000 |
| macro avg | 0.77 | 0.51 | 0.46 | 10000 |
| weighted avg | 0.79 | 0.80 | 0.72 | 10000 |

Table 4: Classification Report for Gaussian Radial(RBF) Kernel

| | **precision** | **recall** | **f1-score** | **support** |
|---|---|---|---|---|
| 0 | 0.80 | 1.00 | 0.89 | 7987 |
| 1 | 0.00 | 0.00 | 0.00 | 2013 |
| accuracy | | | 0.73 | 10000 |
| macro avg | 0.40 | 0.50 | 0.44 | 10000 |
| weighted avg | 0.64 | 0.80 | 0.71 | 10000 |

### 4.4.2 Support Vector Machine Using A Polynomial Kernel with SMOTE

After SMOTE was used with this model,we had the Specificity for the model to be evaluated to:

**Specificity:** 0.568802781917536

The classification report, confusion matrix, ROC Curve and its AUC have been generated below:

Table 5: Classification Report for Polynomial Kernel with SMOTE

| | **precision** | **recall** | **f1-score** | **support** |
|---|---|---|---|---|
| 0 | 0.88 | 0.82 | 0.85 | 7987 |
| 1 | 0.44 | 0.56 | 0.50 | 2013 |
| accuracy | | | 0.77 | 10000 |
| macro avg | 0.66 | 0.69 | 0.67 | 10000 |
| weighted avg | 0.79 | 0.77 | 0.78 | 10000 |

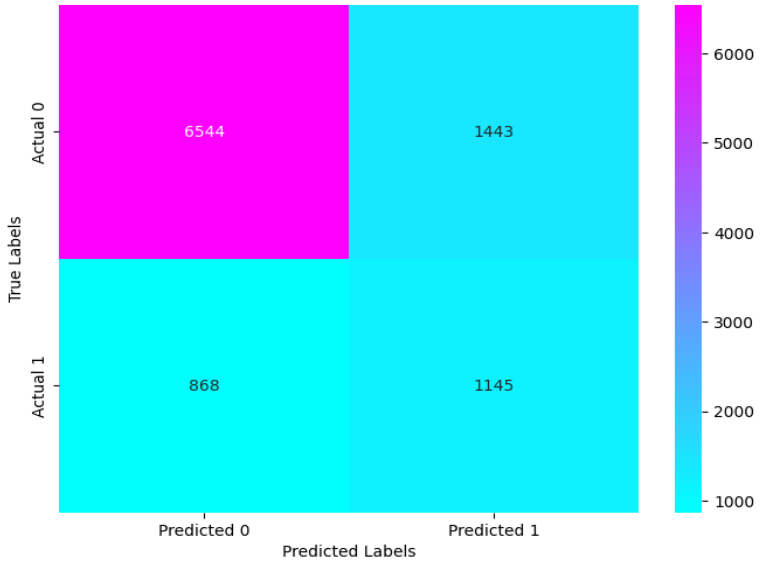


Figure 6: Confusion Matrix for Support Vector Machine Using A Polynomial Kernel with SMOTE

This shows that the True Positives were 6544. False Negatives and False Positives were 1443 and 868 respectively. The True Negatives were 1145.

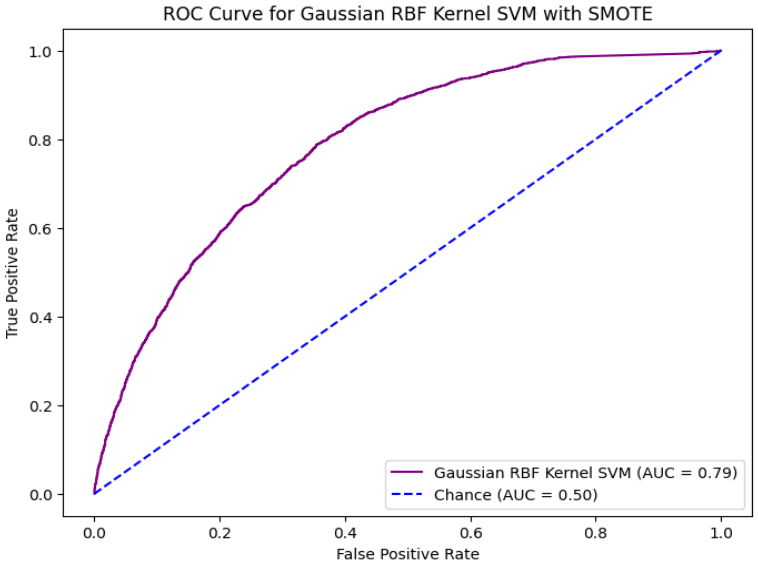


Figure 7: ROC Curve for Support Vector Machine Using Polynomial Kernel with SMOTE The Area under the curve for this model is given as 0.78.

### 4.4.3 Classification Report for Gaussian Radial(RBF) Kernel with SMOTE

After SMOTE was used with this model,we had the Specificity for the model to be evaluated to:

**Specificity:** 0.7208147044212618

The classification report, confusion matrix, ROC Curve and its AUC have been generated below:

Table 6: Classification Report for Gaussian Radial(RBF) Kernel with SMOTE

| | **precision** | **recall** | **f1-score** | **support** |
|---|---|---|---|---|
| 0 | 0.91 | 0.70 | 0.79 | 7987 |
| 1 | 0.38 | 0.72 | 0.49 | 2013 |
| accuracy | | | 0.70 | 10000 |
| macro avg | 0.64 | 0.71 | 0.64 | 10000 |
| weighted avg | 0.80 | 0.70 | 0.73 | 10000 |

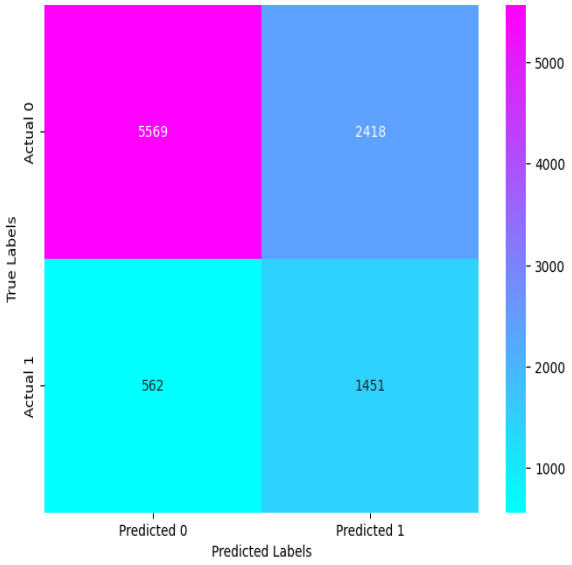


Figure 8: Confusion Matrix for Support Vector Machine Using Gaussian Radial (RBF) Kernel with SMOTE

This shows that the True Positives were 5569. False Negatives and False Positives were 2418 and 562 respectively. The True Negatives were 1451.

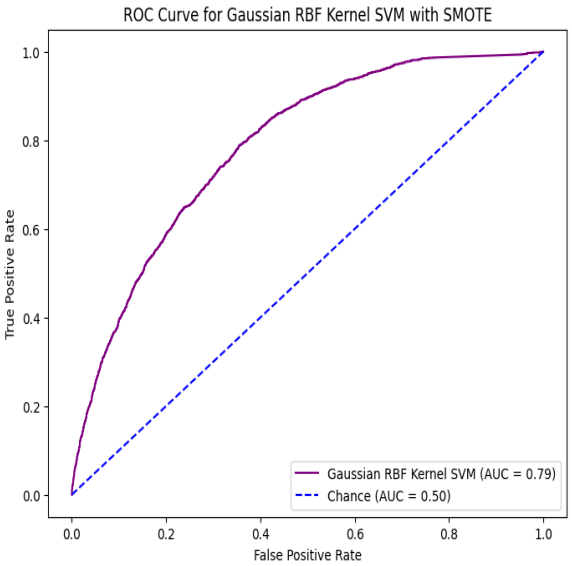


Figure 9: ROC Curve for Support Vector Machine Using A Using Gaussian Radial (RBF) Kernel with SMOTE

The Area under the curve for this model is given as 0.79

# 5 Discussion

This study used Logistic Regression, Random Forest and Support Vector Machine models for classifying customer responses to marketing campaigns using data from an insurance company. Different performance characteristics were observed during evaluation of each modelling approach which were affected by the distribution of the underlying data as well as the features employed.

In terms of the negative class, the Logistic Regression model showed high precision but low recall for the positive class. This means that it can be trusted to predict non-responses but fails to recognize responders. The linearity of logistic regression might not capture intricate details in data; thus this could be one reason why such a model has limitations.

Random Forest showed a more balanced performance between the classes compared to Logistic Regression. This better result might have been achieved by the model's capability of dealing with non-linear relationships and feature interactions without explicit specification. Nevertheless, Random Forest predictions' volatility suggests that there is a problem of overfitting which happens when models pick up noise in training data rather than generalising from true patterns underneath them.

The Support Vector Machines were shown to have improved recall for the positive class especially when used with SMOTE in handling class imbalance. Hence this implies that if the underrepresented group is well represented, SVMs can handle imbalanced data very effectively. Employing various kernels like polynomial and Gaussian RBF enabled mapping of observations into higher dimensions thereby achieving better separation between classes.

Overall, the study emphasizes that machine learning models need to be picked with care in relation to dataset features and analysis goals. What is suggested by the results is a combination method which uses different types of models for better predictive power and practical applicability. Additionally, this investigation brings into focus preprocessing techniques such as SMOTE that deal with class imbalance – one of the most frequent problems encountered while solving binary classification tasks.

Future work could explore more complex ensemble methods that combine predictions from multiple models to improve accuracy and robustness. Furthermore, model performance could be improved by trying out advanced feature selection approaches and hyperparameter optimization.

# References


1. Salcedo-Sanz, S., DePrado-Cumplido, M., Segovia-Vargas, M. J., Pérez-Cruz, F. & Bousoño-Calzón, C. Feature selection methods involving support vector machines for prediction of insolvency in non-life insurance companies. Intelligent Systems in Accounting, Finance & ampManagement 12, 261–281 (2004).

2. Kartasheva, A. V. & Traskin, M. Insurers' insolvency prediction using random forest classification. in (2013).

3. Montserrat, G., Jan, P., Chresten, D. & Ana M, P. Customer Loyalty In The Insurance Industry: A Logistic Regression Approach. Conference in Actuarial Science and Finance on Samos.(2002).

4. Behr, A. & Weinblat, J. Default patterns in seven EU countries: A random forest approach. International Journal of the Economics of Business 24, 181–222 (2016).

# Appendix

## Python Code

```
import pandas as pd
import numpy as np
import io
from sklearn.linear_model import LogisticRegression
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, log_loss
from sklearn.metrics import classification_report
df_train = pd.read_csv('InsuranceData_train.csv')
print(df_train)
print(df_train.shape)
frequency_table = df_train['PURCHASE'].value_counts()

print(frequency_table)
df_valid= pd.read_csv('InsuranceData_valid.csv')
print(df_valid)
print(df_valid.shape)
def iv_woe(data, target, bins=10, show_woe=False):

    #Empty Dataframe
    newDF,woeDF = pd.DataFrame(), pd.DataFrame()

    #Extract Column Names
    cols = data.columns

    #Run WOE and IV on all the independent variables
    for ivars in cols[~cols.isin([target])]:
        if (data[ivars].dtype.kind in 'bifc') and (len(np.unique(data[ivars]))
            binned_x = pd.qcut(data[ivars], bins,  duplicates='drop')
            d0 = pd.DataFrame({'x': binned_x, 'y': data[target]})
        else:
            d0 = pd.DataFrame({'x': data[ivars], 'y': data[target]})


        # Calculate the number of events in each group (bin)
        d = d0.groupby("x", as_index=False).agg({"y": ["count", "sum"]})
        d.columns = ['Cutoff', 'N', 'Events']

        # Calculate % of events in each group.
        d['% of Events'] = np.maximum(d['Events'], 0.5) / d['Events'].sum()
```

```
            # Calculate the non events in each group.
            d['Non-Events'] = d['N'] - d['Events']
            # Calculate % of non events in each group.
            d['% of Non-Events'] = np.maximum(d['Non-Events'], 0.5) / d['Non-Even

            # Calculate WOE by taking natural log of division of % of non-events
            d['WoE'] = np.log(d['% of Events']/d['% of Non-Events'])
            d['IV'] = d['WoE'] * (d['% of Events'] - d['% of Non-Events'])
            d.insert(loc=0, column='Variable', value=ivars)
            print("Information value of " + ivars + " is " + str(round(d['IV'].su
            temp =pd.DataFrame({"Variable" : [ivars], "IV" : [d['IV'].sum()]}, co
            newDF=pd.concat([newDF,temp], axis=0)
            woeDF=pd.concat([woeDF,d], axis=0)

            #Show WOE Table
            if show_woe == True:
                print(d)
        return newDF, woeDF
    #Information Value

import numpy as np
import pandas as pd
np.random.seed(100)

newDF, woeDF = iv_woe(df_train,'PURCHASE', bins=10, show_woe=True)
print(newDF)
# Filtering newDF to find variables with IV less than 0.1
high_iv_variables = newDF[newDF['IV'] >= 0.1]['Variable']
print(high_iv_variables)
print(high_iv_variables.shape)
# Selecting these variables from the training_data DataFrame
selected_traindata = df_train[high_iv_variables.tolist()]

print(selected_traindata)
print(selected_traindata.shape)
selected_traindata.columns
# Selecting these variables from the valid_data DataFrame
selected_validdata = df_valid[high_iv_variables.tolist()]
print(selected_validdata)
print(selected_validdata.shape)
# Instantiate a logistic regression model.
model = LogisticRegression(max_iter=10000)

# Fit the model to the training data.
```

```
model.fit(selected_traindata, df_train['PURCHASE'])

# Make predictions on the validation data.
predictions = model.predict(selected_validdata)

# Evaluate the model's performance.
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(df_valid['PURCHASE'], predictions)
print('Accuracy:', accuracy)
# Define the parameter grid
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.1, 1.0, 10.0],
    'solver': ['liblinear', 'saga'],
    'random_state': [42, 123, 456]
}

# Define a scorer for the deviance (negative log-loss)
deviance_scorer = make_scorer(log_loss, greater_is_better=False, needs_proba=

# Create the GridSearchCV object
logistic_grid = GridSearchCV(
    LogisticRegression(max_iter=100),
    param_grid,
    scoring=deviance_scorer,
    cv=5,
    n_jobs=-1
)
# Fit the GridSearchCV object to the training data
logistic_grid.fit(selected_traindata, df_train['PURCHASE'])

# Get the optimal model
optimal_model = logistic_grid.best_estimator_
print("Best Parameters:", logistic_grid.best_params_)
newDF[newDF['IV'] >= 0.1]
predictions = optimal_model.predict(selected_traindata)

# Evaluate the optimal model's performance
deviance = log_loss(df_train['PURCHASE'], optimal_model.predict_proba(selecte
accuracy = accuracy_score(df_train['PURCHASE'], predictions)
print('Accuracy:', accuracy)
print(f"Deviance (Log Loss) on Validation Data: {deviance:.4f}")
# Make predictions on the validation data
predictions = optimal_model.predict(selected_validdata)
predictions
```

```
print(classification_report(df_valid['PURCHASE'], predictions))
from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt
import seaborn as sns

# Create a confusion matrix
confusion_mat = confusion_matrix(df_valid['PURCHASE'], predictions)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='viridis')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
# Calculate Sensitivity, Specificity, and Accuracy
sensitivity = confusion_mat[0, 0] / (confusion_mat[0, 0] + confusion_mat[0, 1
specificity = confusion_mat[1, 1] / (confusion_mat[1, 0] + confusion_mat[1, 1


# Print the results
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("Accuracy:", accuracy)

# Evaluate the deviance on the validation data
deviance = log_loss(df_valid['PURCHASE'], optimal_model.predict_proba(selecte
print(f"Deviance (Log Loss) on Validation Data: {deviance:.4f}")
from sklearn.metrics import roc_curve, auc

# Calculate the false positive rate (FPR) and true positive rate (TPR)
fpr, tpr, _ = roc_curve(df_valid['PURCHASE'], optimal_model.predict_proba(sel

# Calculate the area under the curve (AUC)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)'
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')
```

```
plt.legend(loc="lower right")
plt.show()

# Print the AUC value
print('Area under the ROC curve:', roc_auc)
X_train = selected_traindata
y_train = df_train['PURCHASE']

# Prepare the validation data
X_valid = selected_validdata
y_valid = df_valid['PURCHASE']
from sklearn.preprocessing import StandardScaler

# Create a scaler object
scaler = StandardScaler()

# Fit on training data and transform both training and validation data
X_train_scaled = scaler.fit_transform(X_train)
X_valid_scaled = scaler.transform(X_valid)
# Applying SMOTE to the training data
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)
from sklearn.ensemble import RandomForestClassifier

oob_errors = []
num_features = range(1, 14)  # Features to consider at each split
smote = SMOTE(random_state=423)

for i in num_features:
    # Apply SMOTE
    X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train

    # Fit Random Forest
    rf = RandomForestClassifier(n_estimators=10001, max_features=i, oob_score
    rf.fit(X_train_smote, y_train_smote)
    oob_error = 1 - rf.oob_score_
    oob_errors.append(oob_error)

# Find the model with the lowest OOB error
min_oob_error = min(oob_errors)
best_features = num_features[oob_errors.index(min_oob_error)]

# Plotting
plt.figure(figsize=(10, 6))
```

```
plt.plot(num_features, oob_errors, marker='o')
plt.scatter(best_features, min_oob_error, color='red')
plt.axhline(y=min_oob_error, linestyle='--', color='red')
plt.axvline(x=best_features, linestyle='--', color='red')
plt.title('Random Forest OOB Error by Number of Features')
plt.xlabel('Number of Features (max_features)')
plt.ylabel('OOB Error Rate')
plt.show()
# Use the best model to make predictions on the validation data
best_model = rf  # Replace with the actual best model from your code
predictions = best_model.predict(X_valid_scaled)

# Print the predictions
print(predictions)
# Create a confusion matrix
confusion_mat = confusion_matrix(y_valid, predictions)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='viridis')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
# Calculate Sensitivity, Specificity, and Accuracy
sensitivity = confusion_mat[0, 0] / (confusion_mat[0, 0] + confusion_mat[0, 1
specificity = confusion_mat[1, 1] / (confusion_mat[1, 0] + confusion_mat[1, 1
accuracy = (confusion_mat[0, 0] + confusion_mat[1, 1]) / (confusion_mat[0, 0]

# Print the results
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("Accuracy:", accuracy)
# Calculate the false positive rate (FPR) and true positive rate (TPR)
fpr, tpr, _ = roc_curve(y_valid, best_model.predict_proba(X_valid_scaled)[:,

# Calculate the area under the curve (AUC)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)'
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')
plt.legend(loc="lower right")
plt.show()

# Print the AUC value
print('Area under the ROC curve:', roc_auc)
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
# Building and training the SVM model using a polynomial kernel
svm_poly = SVC(kernel='poly', degree=3, C=0.01, random_state=423, probability
svm_poly.fit(X_train_scaled, y_train)  # Use the scaled training data
# Building and training the SVM model using a Gaussian radial (RBF) kernel
svm_rbf = SVC(kernel='rbf', gamma=0.000001, C=0.01, random_state=423, probab
svm_rbf.fit(X_train_scaled, y_train)  # Use the same scaled training data
# Making predictions with the polynomial kernel SVM
predictions_poly = svm_poly.predict(X_valid_scaled)

# Making predictions with the Gaussian RBF kernel SVM
predictions_rbf = svm_rbf.predict(X_valid_scaled)
# Evaluating the model
confusion_matrix_poly_smote = confusion_matrix(y_valid, predictions_poly_smot

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='viridis')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
# Training the SVM model without class weights
svm_poly_smote = SVC(kernel='poly', degree=3, C=0.01, random_state=423, proba
svm_poly_smote.fit(X_train_smote, y_train_smote)

# Making predictions on the validation set
predictions_poly_smote = svm_poly_smote.predict(X_valid_scaled)


classification_report_poly_smote = classification_report(y_valid, predictions
print("\nClassification Report with SMOTE:")
print(classification_report_poly_smote)
accuracy_poly_smote = accuracy_score(y_valid, predictions_poly_smote)
print("Accuracy for Polynomial Kernel SVM with SMOTE:", accuracy_poly_smote)

# Confusion Matrix Heatmap
plt.figure(figsize=(8, 6))
```

```
sns.heatmap(confusion_matrix_poly_smote, annot=True, fmt="d", cmap="Blues", x
#plt.title('Confusion Matrix Heatmap for Polynomial Kernel SVM with SMOTE')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
# For Polynomial Kernel SVM with SMOTE
if hasattr(svm_poly_smote, "decision_function"):
    decision_scores_poly = svm_poly_smote.decision_function(X_valid_scaled)
else:
    decision_scores_poly = svm_poly_smote.predict_proba(X_valid_scaled)[:, 1]

fpr_poly, tpr_poly, _ = roc_curve(y_valid, decision_scores_poly)
auc_poly = auc(fpr_poly, tpr_poly)

# Plotting ROC Curve for Polynomial Kernel SVM
plt.figure(figsize=(8, 6))
plt.plot(fpr_poly, tpr_poly, label=f'Polynomial Kernel SVM (AUC = {auc_poly:.
plt.plot([0, 1], [0, 1], 'k--', label='Chance (AUC = 0.50)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Polynomial Kernel SVM with SMOTE')
plt.legend(loc="lower right")
plt.show()
from imblearn.over_sampling import SMOTE

smote = SMOTE()
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)

svm_rbf_smote = SVC(kernel='rbf', gamma=0.000001, C=0.01, class_weight='balan
svm_rbf_smote.fit(X_train_smote, y_train_smote)
predictions_rbf_smote = svm_rbf_smote.predict(X_valid_scaled)

print(confusion_matrix(y_valid, predictions_rbf_smote))
print(classification_report(y_valid, predictions_rbf_smote))
accuracy_rbf_smote = accuracy_score(y_valid, predictions_rbf_smote)
print("Accuracy for Gaussian RBF Kernel SVM with SMOTE:", accuracy_rbf_smote)

# Generate and display the confusion matrix heatmap
confusion_matrix_rbf_smote = confusion_matrix(y_valid, predictions_rbf_smote)

plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix_rbf_smote, annot=True, fmt="d", cmap="Blues", xt
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
if hasattr(svm_rbf_smote, "decision_function"):
    decision_scores_rbf = svm_rbf_smote.decision_function(X_valid_scaled)
else:
    decision_scores_rbf = svm_rbf_smote.predict_proba(X_valid_scaled)[:, 1]

fpr_rbf, tpr_rbf, _ = roc_curve(y_valid, decision_scores_rbf)
auc_rbf = auc(fpr_rbf, tpr_rbf)

# Plotting ROC Curve for Gaussian RBF Kernel SVM
plt.figure(figsize=(8, 6))
plt.plot(fpr_rbf, tpr_rbf, label=f'Gaussian RBF Kernel SVM (AUC = {auc_rbf:.2
plt.plot([0, 1], [0, 1], 'k--', label='Chance (AUC = 0.50)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Gaussian RBF Kernel SVM with SMOTE')
plt.legend(loc="lower right")
plt.show()
```