University of Essex

Department of Mathematical Sciences

COMBINED COURSEWORK REASSESSMENT FOR
CE802

COMPARATIVE STUDY: INVESTIGATING THE EFFECTIVENESS OF MACHINE LEARNING FOR TARGETED EMAIL MARKETING

WORD COUNT: 792

2202026

August 23, 2023
Colchester

Contents

1 Introduction **3**

2 Data Exploration and Pre–processing **4**

3 Results and Analysis **7**

3.1 Interpretation 8

4 Conclusions **9**

5 Appendix **10**

CHAPTER 1

Introduction

This research delved into the evaluation of various machine learning methodologies aimed at predicting email success in generating sales. Leveraging historical data supplied by the company, this study explored the effectiveness of algorithms including Naive Bayes, Decision Trees, Logistic Regression, and Support Vector Machines (SVM). The overarching objective was to pinpoint the optimal strategy for enhancing email campaign precision and bolstering overall conversion rates.

CHAPTER 2

Data Exploration and Pre–processing

The data set required little to no pre–processing techniques as it consisted of numerical variables and had no missing values.

The plots below were done to identify false predictors. Most of the plots have a pattern which means that most of our independent variables have a relationship with the dependent variable which is the class variable. The only plot that doesn't have a pattern is the F4 plot.

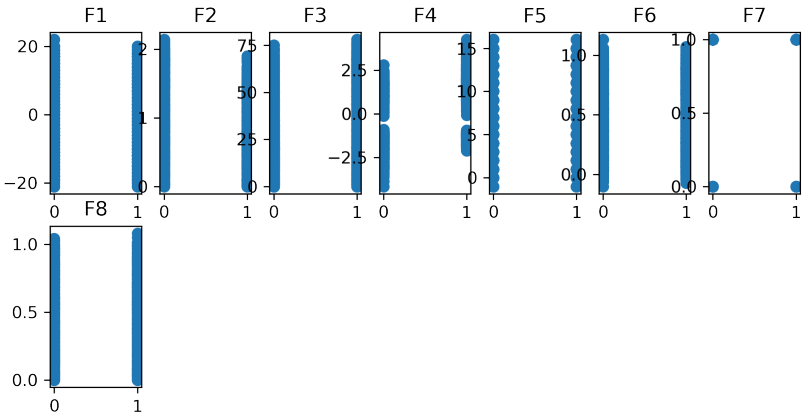

Figure 2.1: Scatter Plot of Features and Labels

The patterns shown in 2.1 also showcase that the features are not identical as they all have their own unique relationship with the labels.

Scaling of each input features was done to ensure there is fair comparison between them.

The figure below showcases how the features used had no significant outliers.

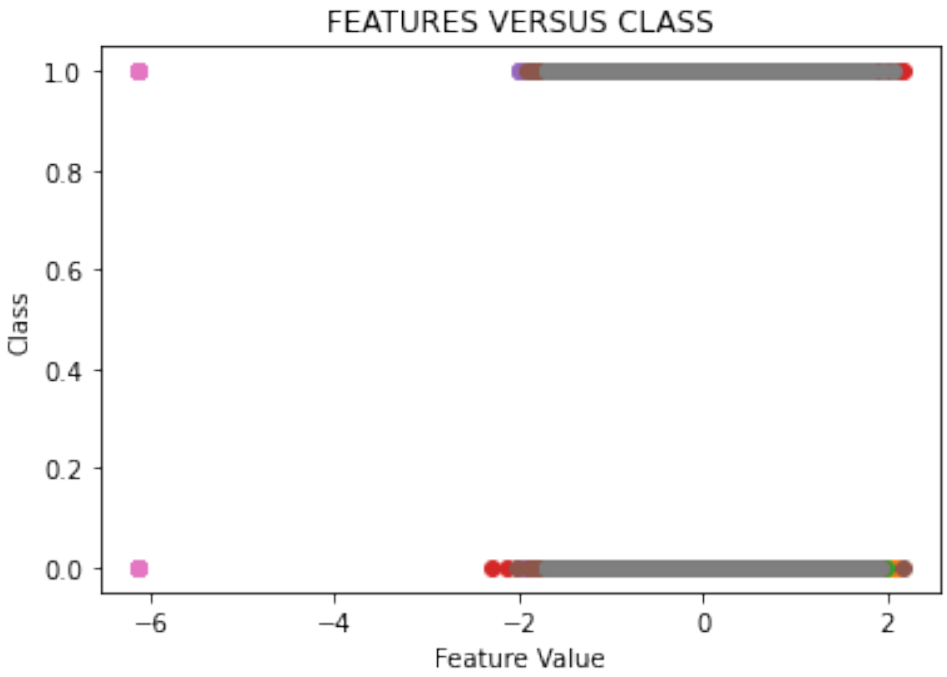

Figure 2.2: Features versus Class

The Scikit-learn library in Python was utilized to implement and evaluate the machine learning models. The dataset consisted of 1000 examples, each with 8 features and a binary label indicating whether the corresponding email led to a sale. The data set was randomly split into a training set (70%) and a testing set (30%) for model evaluation. For each algorithm, the model was trained on the training set and its performance was evaluated on the testing set.

The distribution of the classes is slightly evenly split as depicted in the chart below.

Figure 2.3: Distribution of the classes

CHAPTER 3

Results and Analysis

The performance of each algorithm was measured using the following key performance metrics accuracy, precision, recall, F1-score and Cohen's Kappa statistic.

The performance of these performance metrics on each classification technique is depicted in the chart below.

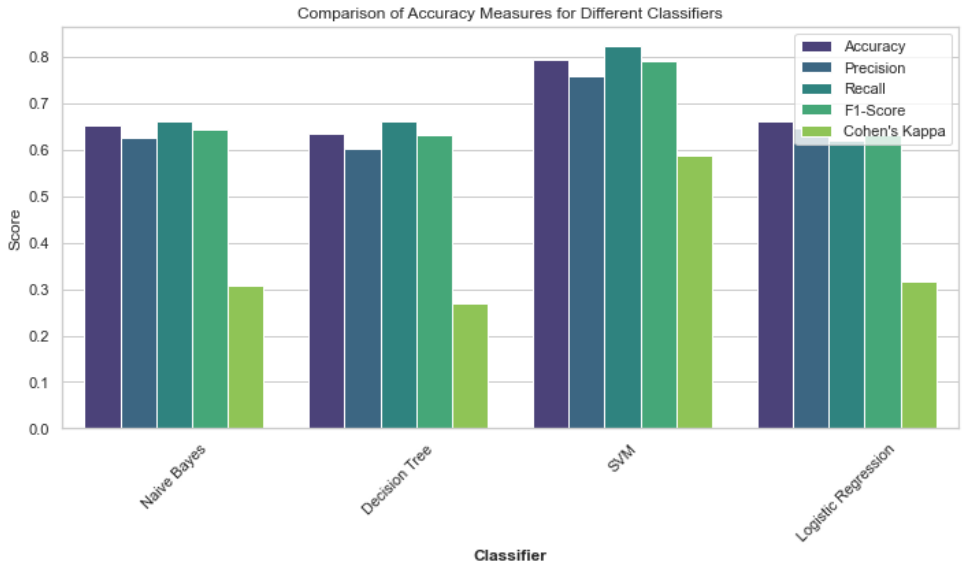

Figure 3.1: Comparison of Accuracy Measures for Different Classifiers

3.1 Interpretation

The performance metrics on 3.1 provide insights into the effectiveness of different classifiers in a classification task. The "Accuracy" measure signifies the proportion of correct predictions overall. "Precision" illustrates the ratio of true positive predictions to all instances predicted as positive, emphasizing the precision of positive predictions. "Recall" quantifies the ratio of true positives to actual positive instances, indicating the classifier's capability to identify all relevant cases. The "F1-Score" strikes a balance between precision and recall, offering a unified value that considers both false positives and false negatives. Additionally, "Cohen's Kappa" statistic evaluates the agreement between the classifier's predictions and the actual classes, while considering random chance.

Among the classifiers examined, the Support Vector Machine (SVM) stands out with notable performance. SVM achieves an accuracy of 0.793, indicating a high proportion of correct predictions. Its precision value of 0.760 showcases accurate positive predictions, while its recall value of 0.824 indicates its proficiency in capturing actual positive instances. The F1-Score, at 0.791, reflects the balance between precision and recall. Furthermore, the Cohen's Kappa score of 0.587 signifies substantial agreement beyond random chance.

In comparison, the Naive Bayes classifier achieves an accuracy of 0.653 with a precision of 0.627, recall of 0.662, F1-Score of 0.644, and Cohen's Kappa of 0.307. The Decision Tree classifier records an accuracy of 0.633, precision of 0.603, recall of 0.662, F1-Score of 0.631, and Cohen's Kappa of 0.268. The Logistic Regression classifier yields an accuracy of 0.660, precision of 0.647, recall of 0.620, F1-Score of 0.633, and Cohen's Kappa of 0.317. These metrics collectively highlight the SVM's superiority in this classification task compared to the other classifiers.

CHAPTER 4

Conclusions

The comparative study highlighted the strengths and weaknesses of different machine learning techniques for predicting email campaign effectiveness. Overall, the study suggests that leveraging Support Vector Machines could significantly enhance the company's targeted email marketing strategy and campaign conversion rates.

Support Vector Machines (SVMs) offer significant advantages for targeted email marketing. They excel in handling both linear and complex non-linear relationships in customer data, making them adept at capturing intricate behaviors and preferences. SVMs' ability to manage high-dimensional data, tackle imbalanced datasets, and resist the influence of outliers is particularly valuable in email marketing scenarios. Their interpretability through support vectors aids in understanding influential features, while parameter tuning controls overfitting and ensures generalization. SVMs accommodate smaller datasets common in marketing campaigns and maintain consistency in predictions. Their adaptability to changing data suits evolving marketing strategies, and diverse kernel functions offer flexibility to model different customer behaviors. While parameter tuning and computational complexity should be considered, SVMs stand as a powerful tool for enhancing targeted email marketing efforts.

The code for this study is available in the Jupyter notebook provided in the appendix section.

CHAPTER 5

Appendix

```
#INVESTIGATE THE PERFORMANCE OF A NUMBER OF MACHINE LEARNING PROCEDURES ON THIS DATASET USING PYTHON/SCIKIT-LEARN.

#DATA PRE-PROCESSING

#### Importing necessary libraries

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import pandas as pd
from sklearn.preprocessing import StandardScaler

data = pd.read_csv('CE802_Ass_Resit.csv')
data

###Checking the data types
data.dtypes


####Transfering the data to two numpy arrays, one containing the variables (features) and another the labels (ground truth).

import numpy as np
```

```
x = data.loc[:, data.columns != 'C'].to_numpy() #features
y = data.loc[:, data.columns == 'C'].to_numpy() #labels

print(data.shape)
print(x.shape, y.shape)

#### Identifying false predictors
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
for i in range(x.shape[1]):
    plt.subplot(3, 7, i+1)
    plt.scatter(y,x[:,i])
    plt.xticks([0, 1])
    plt.title(data.columns[i])
    # Save the plot as an image file
    plt.savefig('false_predictors.png', dpi=300)

np.sum(np.isnan(x), 0)
np.sum(np.isnan(y), 0)

# From the code above we can see that none of features and the labels had
   missing values.

### Convert the label's column vector to a 1d array
y = y.ravel()

###Scaling the data
scaler = StandardScaler()
X = scaler.fit_transform(x)

##Identifying Outliers
# Create the scatter plot
for i in range(8):
    plt.scatter(X[:, i], y)

# Add a title and axis labels
plt.title('FEATURES VERSUS CLASS')
plt.xlabel('Feature Value')
```

```
plt.ylabel('Class')
plt.savefig('featuresversusclass.png', dpi=300)
plt.show()

import pandas as pd

# Convert the NumPy array y into a pandas Series
y_series = pd.Series(y)

# Use value_counts() to count unique values and plot a bar chart
y_series.value_counts().plot.bar()

# Add labels and title
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Distribution of Classes')
plt.show()

##Split the data into training and test sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3,
    random_state=42)

# A NAIVE BAYES CLASSIFIER

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, cohen_kappa_score

# Create a Naive Bayes classifier
clf = GaussianNB()

# Naive Bayes doesn't have hyperparameters to tune with GridSearchCV

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Make predictions using the trained model
naive_bayes_predictions = clf.predict(X_val)
```

```
98
99  # Calculate accuracy
100 naive_bayes_accuracy = accuracy_score(y_val, naive_bayes_predictions)
101 print("Validation accuracy:", naive_bayes_accuracy)
102
103 # Calculate precision
104 naive_bayes_precision = precision_score(y_val, naive_bayes_predictions)
105 print("Precision:", naive_bayes_precision)
106
107 # Calculate recall
108 naive_bayes_recall = recall_score(y_val, naive_bayes_predictions)
109 print("Recall:", naive_bayes_recall)
110
111 # Calculate F1-score
112 naive_bayes_f1 = f1_score(y_val, naive_bayes_predictions)
113 print("F1-score:", naive_bayes_f1)
114
115 # Calculate the Cohen's Kappa statistic
116 naive_bayes_kappa = cohen_kappa_score(y_val, naive_bayes_predictions)
117 print("Cohen's Kappa:", naive_bayes_kappa)
118
119 #DECISION TREES
120
121 from sklearn.tree import DecisionTreeClassifier
122 from sklearn.model_selection import GridSearchCV
123
124 # Create a Decision Tree classifier
125 clf = DecisionTreeClassifier(random_state=0)
126
127 # Define the parameter grid for grid search
128 param_grid = {
129     'criterion': ['gini', 'entropy'],
130     'max_depth': [None, 5, 10, 20],
131     'min_samples_split': [2, 5, 10],
132     'min_samples_leaf': [1, 2, 4]
133 }
134
135 # Create the GridSearchCV object
136 grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
```

```
137
138 # Fit the grid search to the training data
139 grid_search.fit(X_train, y_train)
140
141 # Get the best parameters and the corresponding model
142 best_params = grid_search.best_params_
143 best_model = grid_search.best_estimator_
144
145 # Make predictions using the best model
146 decision_tree_predictions = best_model.predict(X_val)
147 print("Best parameters:", best_params)
148
149 # Calculate accuracy
150 decision_tree_accuracy = accuracy_score(y_val, decision_tree_predictions)
151 print("Validation accuracy:", decision_tree_accuracy)
152
153 # Calculate precision
154 decision_tree_precision = precision_score(y_val, decision_tree_predictions)
155 print("Precision:", decision_tree_precision)
156
157 # Calculate recall
158 decision_tree_recall = recall_score(y_val, decision_tree_predictions)
159 print("Recall:", decision_tree_recall)
160
161 # Calculate F1-score
162 decision_tree_f1 = f1_score(y_val, decision_tree_predictions)
163 print("F1-score:", decision_tree_f1)
164
165 # Calculate the Cohen's Kappa statistic
166 decision_tree_kappa = cohen_kappa_score(y_val, decision_tree_predictions)
167 print("Cohen's Kappa:", decision_tree_kappa)
168
169 ## SUPPORT VECTOR MACHINES
170 from sklearn.svm import SVC
171 from sklearn.model_selection import GridSearchCV
172 from sklearn.metrics import accuracy_score
173
174 # Create an SVM classifier
175 clf = SVC(random_state=0)
```

```
176
177 # Define the parameter grid for grid search
178 param_grid = {
179     'C': [0.1, 1, 10],
180     'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
181     'degree': [2, 3, 4],
182     'gamma': ['scale', 'auto']
183 }
184
185 # Create the GridSearchCV object
186 grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
187
188 # Fit the grid search to the training data
189 grid_search.fit(X_train, y_train)
190
191 # Get the best parameters and the corresponding model
192 best_params = grid_search.best_params_
193 best_model = grid_search.best_estimator_
194
195 # Make predictions using the best model
196 svm_predictions = best_model.predict(X_val)
197 print("Best parameters:", best_params)
198
199 # Calculate accuracy
200 svm_accuracy = accuracy_score(y_val, svm_predictions)
201 print("Validation accuracy:", svm_accuracy)
202
203 # Calculate precision
204 svm_precision = precision_score(y_val, svm_predictions)
205 print("Precision:", svm_precision)
206
207 # Calculate recall
208 svm_recall = recall_score(y_val, svm_predictions)
209 print("Recall:", svm_recall)
210
211 # Calculate F1-score
212 svm_f1 = f1_score(y_val, svm_predictions)
213 print("F1-score:", svm_f1)
214
```

```
# Calculate the Cohen's Kappa statistic
svm_kappa = cohen_kappa_score(y_val, svm_predictions)
print("Cohen's Kappa:", svm_kappa)

#LOGISTIC REGRESSIONS

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Create a Logistic Regression classifier
clf = LogisticRegression(max_iter=10000)

# Define the parameter grid for grid search
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.1, 1, 10],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
}

# Create the GridSearchCV object
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and the corresponding model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Make predictions using the best model
logistic_regression_predictions = best_model.predict(X_val)
print("Best parameters:", best_params)

# Calculate accuracy
logistic_regression_accuracy = accuracy_score(y_val,
    logistic_regression_predictions)
print("Validation accuracy:", logistic_regression_accuracy)

```

```
# Calculate precision
logistic_regression_precision = precision_score(y_val,
    logistic_regression_predictions)
print("Precision:", logistic_regression_precision)

# Calculate recall
logistic_regression_recall = recall_score(y_val,
    logistic_regression_predictions)
print("Recall:", logistic_regression_recall)

# Calculate F1-score
logistic_regression_f1 = f1_score(y_val, logistic_regression_predictions)
print("F1-score:", logistic_regression_f1)

# Calculate the Cohen's Kappa statistic
logistic_regression_kappa = cohen_kappa_score(y_val,
    logistic_regression_predictions)
print("Cohen's Kappa:", logistic_regression_kappa)

import pandas as pd

# Create a dictionary to store the accuracy measures
accuracy_measures = {
    "Classifier": ["Naive Bayes", "Decision Tree", "SVM", "Logistic
    Regression"],
    "Accuracy": [naive_bayes_accuracy, decision_tree_accuracy, svm_accuracy
    , logistic_regression_accuracy],
    "Precision": [naive_bayes_precision, decision_tree_precision,
    svm_precision, logistic_regression_precision],
    "Recall": [naive_bayes_recall, decision_tree_recall, svm_recall,
    logistic_regression_recall],
    "F1-Score": [naive_bayes_f1, decision_tree_f1, svm_f1,
    logistic_regression_f1],
    "Cohen's Kappa": [naive_bayes_kappa, decision_tree_kappa, svm_kappa,
    logistic_regression_kappa]
}

# Create a DataFrame from the dictionary
accuracy_df = pd.DataFrame(accuracy_measures)
```

```
283
284 # Print the DataFrame
285 print(accuracy_df)
286
287 import pandas as pd
288 import seaborn as sns
289 import matplotlib.pyplot as plt
290
291 # Assuming you have a DataFrame named accuracy_df
292
293 # Set the style of the plot
294 sns.set(style="whitegrid")
295
296 # Melt the DataFrame to create a suitable format for the plot
297 melted_df = accuracy_df.melt(id_vars='Classifier', var_name='Metric',
        value_name='Score')
298
299 # Create the grouped bar plot using seaborn
300 plt.figure(figsize=(10, 6))
301 sns.barplot(x='Classifier', y='Score', hue='Metric', data=melted_df,
        palette='viridis')
302
303 # Add labels and title
304 plt.xlabel('Classifier', fontweight='bold')
305 plt.ylabel('Score')
306 plt.title('Comparison of Accuracy Measures for Different Classifiers')
307
308 # Rotate x-axis labels for better visibility
309 plt.xticks(rotation=45)
310
311 # Show the legend
312 plt.legend()
313
314 # Show the plot
315 plt.tight_layout()
316 plt.show()
317
318 ##support vector machines performed the best
319
```

```
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
    random_state=0)

# Create and train an SVM classifier with the best parameters
clf = SVC(C=10, kernel='rbf', degree=2, gamma='scale', probability=True)
clf.fit(X_train, y_train)

# Make predictions on the validation set
predictions = clf.predict(X_val)

import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the confusion matrix
cm = confusion_matrix(y_val, predictions)

# Visualize the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

Listing 5.1: Codes