CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1. Implementation

This project using python language and on google colab platform. For the dataset, is needed to normalize it first in order to make sure the data is ready to be proceeds. First, import the data and change the data from ".csv" file into a table called dataframe. Once turned into a dataframe, normalize the dataframe by fill the missing value by using mean value of the dataframe, change any string datatype into integer number so it can be processed, and renaming the column of dataframe.

```
1. import pandas as pd
2. import numpy as np
3. import io
4.
5. from google.colab import files
6. uploaded = files.upload()
7.
8. shop_df = pd.read_csv(io.BytesIO(uploaded['ecommerce_data.csv']))
9. shop_df.head()
10.
11. shop_df.shape
12.
13. shop_df.info()
```

In line 1 until 13, import the required library then upload the data file which will then be used as a dataframe for processing on the goggle colab platform. After that, check all the attributes in the data and all info related to data types, number of rows and columns.

```
14. print(shop_df['Month'].value_counts(), "\n")
15. print(shop_df['VisitorType'].value_counts())
16.
17. ubah1 = {'Jan':1, 'Feb':2, 'Mar':3, 'Apr':4, 'May':5, 'Jun':6, 'Jul':7,
    'Aug':8, 'Sep':9, 'Oct':10, 'Nov':11, 'Des':12}
18. shop_df['Month'] = shop_df['Month'].map(ubah1).fillna(0).astype(int)
19.
20. ubah2 = {'Returning_Visitor':0, 'New_Visitor':1, 'Other':2}
21. shop_df['VisitorType']                                              =
    shop_df['VisitorType'].map(ubah2).fillna(0).astype(int)
```

After check all datatypes, there are 4 column identified as string and boolean values, so is needed to change the datatypes into interger value so it can be processed. In line 14 until 21, check

the contents of the "Month" and "VisitorType" data. Then the variable input will be changed according to the contents of each data and converted into an integer data type

```
22. from sklearn.preprocessing import LabelEncoder
23. le = LabelEncoder()
24.
25. # labeling revenue
26. shop_df['Revenue'] = le.fit_transform(shop_df['Revenue'])
27. shop_df['Revenue'].value_counts()
28.
29. # labeling weekend
30. shop_df['Weekend'] = le.fit_transform(shop_df['Weekend'])
31. shop_df['Weekend'].value_counts()
```

In line 22 until 31, also change the boolean data type contained in the data, namely the "Revenue" and "Weekend" columns. Then the data type is changed to numbers 1 and 0, which means True = 1 and False = 0.

```
32. shop_df.info()
33.
34. shop_df.head()
```

Line 32 until 34 only to re-check the data type as a whole whether everything has been correctly changed to numeric or not so that it can be processed.

```
35. X = shop_df.iloc[:,:-1]
36. X.head()
37.
38. y = shop_df["Revenue"]
39. y.head()
```

Line 35 until 39, the data is separated into two parts, **X** and **y**. **X** is the data that becomes the parameter for **y** which will be the prediction target.

```
40. # cek
41. print("X : ", X.shape)
42. print("y : ", y.shape)
```

Line 40 until 42 to check whether the data has been divided correctly according to the columns and rows that will be processed later.

```
43. from sklearn.preprocessing import StandardScaler
44. scaler = StandardScaler()
45. X_scaled = scaler.fit_transform(X)
46.
47. from sklearn.decomposition import PCA
48. pca = PCA(n_components=2)
```

```
49. X_pca = pca.fit_transform(X_scaled)
```

Line 43 until 49, only for linear kernels the data is scaled and dimensions are reduced first so that the linear kernel can work. This is done because the value of feature X is very large, so the linear kernel must be adjusted first.

```
50. import matplotlib.pyplot as plt
51. plt.scatter(X_pca[y==0, 0], X_pca[y==0, 1], c='r', label='class 0')
52. plt.scatter(X_pca[y==1, 0], X_pca[y==1, 1], c='b', label='class 1')
53. plt.xlabel('First Principal Component')
54. plt.ylabel('Second Principal Component')
55. plt.legend()
56. plt.show()
```

Line 50 until 56, the results of the PCA are then visualized to find out the value of X after processing.

```
57. from sklearn.model_selection import train_test_split
58. X_train, X_test, y_train, y_test = train_test_split(X_pca, y,
    test_size=0.2)
59.
60. from sklearn.svm import SVC
61. svm = SVC(kernel='linear')
62. svm.fit(X_train, y_train)
63.
64. from sklearn.metrics import classification_report
65. y_pred = svm.predict(X_test)
66. print(classification_report(y_test, y_pred))
```

Line 57 until 66, only for linear kernel data is divided into training data and testing data according to the PCA process that has been carried out. Then the results are evaluated to find out how accurate the process is.

```
67. from sklearn.model_selection import train_test_split
68. from imblearn.over_sampling import ADASYN
69.
70. adasyn = ADASYN()
71. X_adasyn, y_adasyn = adasyn.fit_resample(X, y)
72.
73. X_train, X_test, y_train, y_test = train_test_split(X_adasyn, y_adasyn,
    test_size = 0.2)
74.
75. print("X_train :", X_train.shape)
76. print("y_train :", y_train.shape)
77. print("X_test  :", X_test.shape)
78. print("y_test  :", y_test.shape)
```

In contrast to the previous linear kernel, the process now applies to all subsequent calculations. Line 67 until 78, divide training data and data for the test into 80% training and 20%

test. This is also done for 60%, 40%, 20% training data and also for test data. Balancing data by oversampling is processed using ADASYN.

```
79. from sklearn import svm, metrics
80. from sklearn.svm import SVC
81. from sklearn.metrics import classification_report
82.
83. model = svm.SVC()
84. model.fit(X_train, y_train)
85.
86. y_pred = model.predict(X_test)
87. print(classification_report(y_test, y_pred))
88.
89. print("Akurasi Training: ", model.score(X_train, y_train))
90. print("Akurasi Testing: ", model.score(X_test, y_test))
```

Line 79 until 90, load the SVM algorithm to perform the prediction calculation process. It then prints out the values that show how accurately this algorithm works.

```
91. from sklearn.model_selection import RandomizedSearchCV
92.
93. parametersRS1 = {'kernel': ['linear', 'rbf', 'sigmoid'],
94.                 'C': [0.5, 1, 10, 100],
95.                 'gamma': [1, 0.1, 0.01, 0.001]
96.                 }
97.
98. random_searchSVM1 = RandomizedSearchCV(estimator=SVC(),
99.                                        param_distributions=parametersRS1,
100.                                       n_jobs=6,
101.                                       verbose=1,
102.                                       scoring='accuracy'
103.                                       )
104.
105.random_searchSVM1.fit(X_train, y_train)
```

Line 91 until 105, perform hyperparameter tuning using RandomizedSearchCV. The parameters used are in accordance with the SVM algorithm provided. This tuning process is carried out randomly by trying all the parameters of the algorithm to find which parameter is the best for the calculation process.

```
106.print(f'Best Score: {random_search.best_score_}')
107.
108.best_params = random_search.best_estimator_.get_params()
109.print(f'Best Parameters:')
110.for param in parameters:
111.    print(f'\t{param}: {best_params[param]}')
```

Line 106 until 111, to check the results of RandomizedSearchCV which parameters are found best for this algorithm.

```
# evaluasi
y_pred = random_search.predict(X_test)
print(classification_report(y_test, y_pred))
```

Line 112 until 114, re-evaluate the score results after tuning using RandomizedSearchCV

```
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler() # Stochastic Gradient Descent sangat sensitif terhadap feature scaling, maka data harus diskalakan dahulu
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

model2 = SGDClassifier(loss="hinge", alpha=0.01, max_iter=200)
model2.fit(X_train, y_train)

y_pred = model2.predict(X_test)
print(classification_report(y_test, y_pred))

print("Akurasi Training: ", model2.score(X_train, y_train))
print("Akurasi Testing: ", model2.score(X_test, y_test))
```

Line 115 until 130, load the SGD algorithm to perform the prediction calculation process. Also uses the StandardScaler because this algorithm is very sensitive to feature scaling, so the data must be scaled first to avoid miscalculations. It then prints out the values that show how accurately this algorithm works.

```
parameters2 = { 'penalty' : ['l1'],
                'alpha' : [1e-4, 1e-3, 1e-2, 1e-1, 1e0],
                'max_iter' : [int(x) for x in np.linspace(1000, 10000, num = 19)],
                'loss':     ['hinge',     'log',     'modified_huber', 'squared_hinge',  'perceptron'],
                'learning_rate' : ['constant', 'optimal', 'invscaling', 'adaptive'],
                'class_weight' : [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}],
                'eta0' : [1, 10, 100]
                }

random_search2 = RandomizedSearchCV(estimator = SGDClassifier(),
                                    param_distributions = parameters2,
                                    n_jobs = 6,
                                    verbose = 1,
                                    scoring = 'accuracy'
                                    )

random_search2.fit(X_train, y_train)
```

Line 131 until 147, perform hyperparameter tuning using RandomizedSearchCV. The parameters used are in accordance with the SGD algorithm provided. This tuning process is carried out randomly by trying all the parameters of the algorithm to find which parameter is the best for the calculation process.

```
148.print(f'Best Score: {random_search2.best_score_}')
149.
150.best_params2 = random_search2.best_estimator_.get_params()
151.print(f'Best Parameters:')
152.for param in parameters2:
153.    print(f'\t{param}: {best_params2[param]}')
```

Line 148 until 153, to check the results of RandomizedSearchCV which parameters are found best for this algorithm.

```
154.# evaluasi
155.y_pred = random_search2.predict(X_test)
156.print(classification_report(y_test, y_pred))
```

Line 154 until 156, re-evaluate the score results after tuning using RandomizedSearchCV

5.2. Results

The Experiment for the optimal result is using 40% of training data and 60% test data. For SVM the best parameter used in hyperparameter tuning using RandomizedSearchCV are :

- Kernel : RBF
- C : 10
- Gamma : Scale

For SGD the best parameter used in hyperparameter tuning using RandomizedSearchCV are :

- Penalty: l1
- Alpha: 0.001
- Max_iter: 7000
- Loss: hinge
- Learning_rate: invscaling
- Class_weight: {1: 0.4, 0: 0.6}
- Eta0: 100

Below are detailed result for each training and test data calculated

Support Vector Machine (SVM)								
	SCORING (without RandomizedSearchCV)				*SCORING (with RandomizedSearchCV)*			
	Test Size = ***20%*** || Training Size = ***80%***							
Label	**Precision**	**Recall**	**F1-Score**	**Support**	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.83	1.00	0.91	2052	0.87	0.95	0.91	2052
1	0.88	0.02	0.03	414	0.53	0.28	0.37	414
	Accuracy = 0.83				***Accuracy = 0.84***			
	Test Size = ***40%*** || Training Size = ***60%***							
Label	**Precision**	**Recall**	**F1-Score**	**Support**	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.84	1.00	0.92	4151	0.84	1.00	0.91	4151
1	0.92	0.02	0.03	781	0.60	0.01	0.02	781
	Accuracy = 0.84				***Accuracy = 0.84***			
	Test Size = ***60%*** || Training Size = ***40%***							
Label	**Precision**	**Recall**	**F1-Score**	**Support**	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.85	1.00	0.92	6248	0.89	0.98	0.93	6248
1	0.90	0.01	0.02	1150	0.75	0.37	0.49	1150
	Accuracy = 0.85				***Accuracy = 0.88***			
	Test Size = ***80%*** || Training Size = ***20%***							
Label	**Precision**	**Recall**	**F1-Score**	**Support**	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.85	1.00	0.92	8349	0.85	1.00	0.92	8349
1	0.50	0.00	0.00	1515	0.87	0.04	0.07	1515
	Accuracy = 0.85				***Accuracy = 0.85***			

Figure 5.1 SVM Result

This is the result of SVM algorithm on predicting online shoper intentions with hyperparameter tuning using RandomizedSearchCV. The data split into 80%, 60%, 40%, 20% on each training set dan testing set. Precision is the percentage of positive predictions that were correct relative to the total positive predictions. Recall is the percentage of positive predictions that were correct relative to the total actual positives. F1-Score is the weighted harmonic mean of precision and recall. The closer to 1, the better the model. Accuracy is the total score of the whole prediction process calculated from three value above.

Stochastic Gradient Descent (SGD)								
	SCORING (without RandomizedSearchCV)				*SCORING (with RandomizedSearchCV)*			
	Test Size = ***20%*** || Training Size = ***80%***							
Label	**Precision**	**Recall**	**F1-Score**	**Support**	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.89	0.97	0.93	2052	0.88	0.98	0.93	2052
1	0.75	0.39	0.51	414	0.76	0.33	0.46	414
	Accuracy = 0.88				***Accuracy = 0.87***			
	Test Size = ***40%*** || Training Size = ***60%***							
Label	**Precision**	**Recall**	**F1-Score**	**Support**	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.89	0.98	0.93	4151	0.91	0.97	0.94	4151
1	0.78	0.34	0.47	781	0.72	0.47	0.57	781
	Accuracy = 0.88				***Accuracy = 0.89***			
	Test Size = ***60%*** || Training Size = ***40%***							
Label	**Precision**	**Recall**	**F1-Score**	**Support**	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.90	0.98	0.93	6248	0.92	0.95	0.94	6248
1	0.75	0.39	0.51	1150	0.69	0.55	0.61	1150
	Accuracy = 0.88				***Accuracy = 0.89***			
	Test Size = ***80%*** || Training Size = ***20%***							
Label	**Precision**	**Recall**	**F1-Score**	**Support**	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.90	0.98	0.94	8349	0.92	0.96	0.94	8349
1	0.77	0.37	0.50	1515	0.70	0.53	0.60	1515
	Accuracy = 0.89				***Accuracy = 0.89***			

Figure 5.2 SGD Result

This is the result of SGD algorithm on predicting online shoper intentions with hyperparameter tuning using RandomizedSearchCV. The data split into 80%, 60%, 40%, 20% on each training set dan testing set. Precision is the percentage of positive predictions that were correct relative to the total positive predictions. Recall is the percentage of positive predictions that were correct relative to the total actual positives. F1-Score is the weighted harmonic mean of precision and recall. The closer to 1, the better the model. Accuracy is the total score of the whole prediction process calculated from three value above.

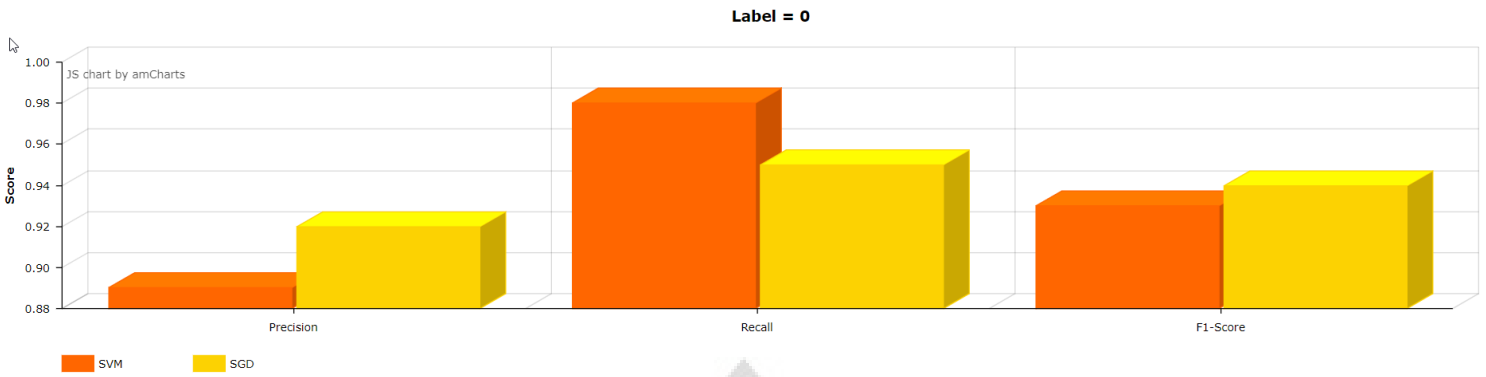

Figure 5.3 Best Parameter Comparison Between SVM and SGD (Label = 0)

This is the best comparison result for "label = 0" of the two algorithms that have been tuned using RandomizedSearchCV.

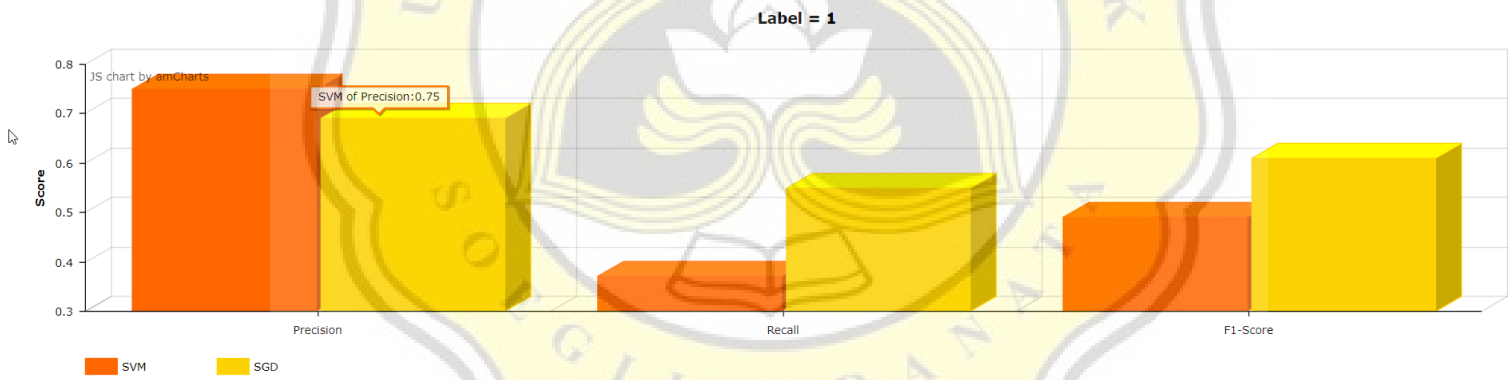

Figure 5.4 Best Parameter Comparison Between SVM and SGD (Label = 1)

This is the best comparison result for "label = 1" of the two algorithms that have been tuned using RandomizedSearchCV.

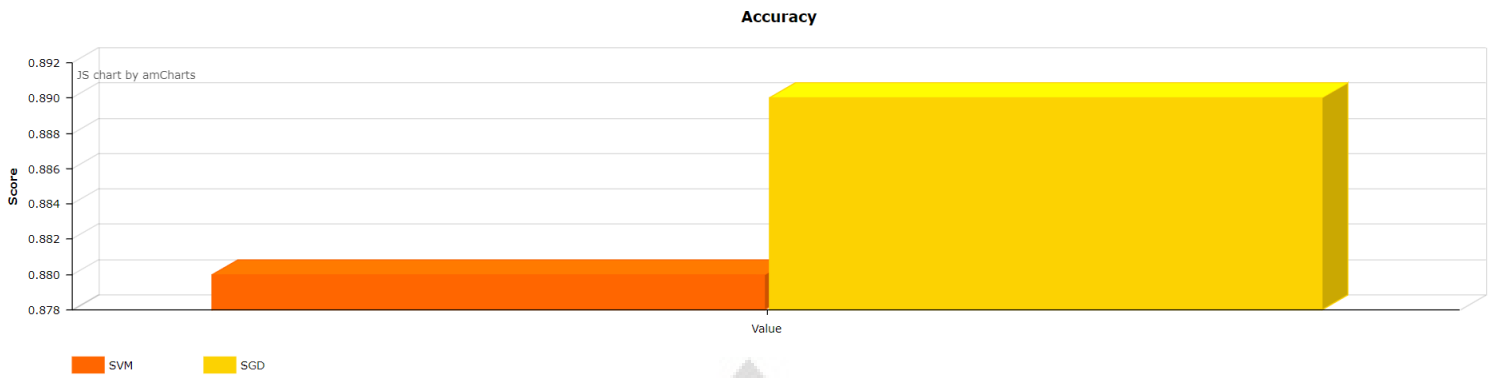

Figure 5.5 Accuracy Comparison Between SVM and SGD

This is the accuracy result for of the two algorithms that have been tuned using RandomizedSearchCV.

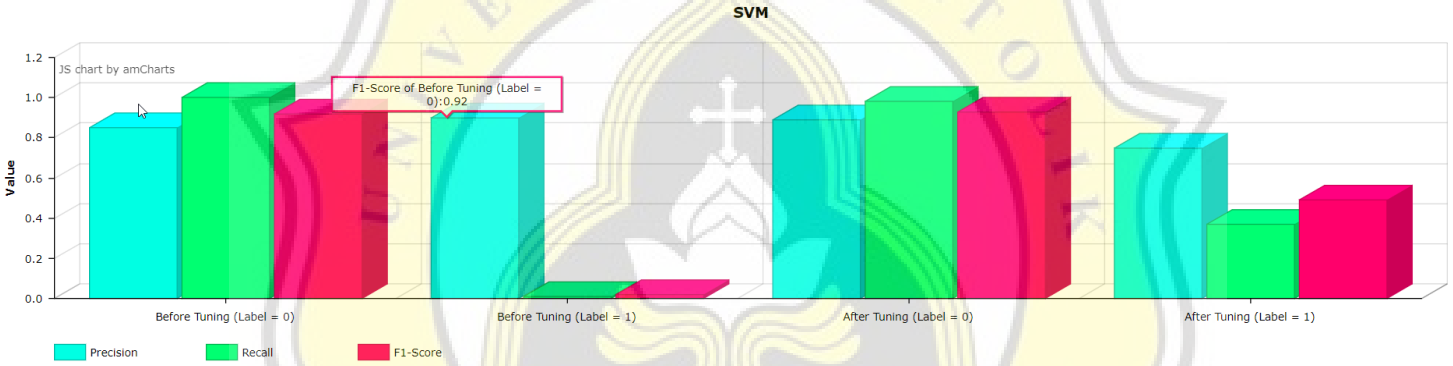

Figure 5.6 RandomizedSearchCV Comparison on SVM

This is the result of a comparison of tuning of SVM using RandomizedSearchCV, taken from the best results for each parameter.

Figure 5.7 RandomizedSearchCV Comparison on SGD

This is the result of a comparison of tuning SGD using RandomizedSearchCV, taken from the best results for each parameter.