APPENDIX

IMPORTING THE DATA AND LIBRARY

```
import pandas as pd
import numpy as np
import io

from google.colab import files
uploaded = files.upload()

shop_df = pd.read_csv(io.BytesIO(uploaded['ecommerce_data.csv']))
shop_df.head()

 shop_df.shape

 shop_df.info()
```

CLEANING THE DATA

```
 print(shop_df['Month'].value_counts(), "\n")
 print(shop_df['VisitorType'].value_counts())

 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}
 shop_df['Month'] = shop_df['Month'].map(ubah1).fillna(0).astype(int)

 ubah2 = {'Returning_Visitor':0, 'New_Visitor':1, 'Other':2}
 shop_df['VisitorType']                                                =
shop_df['VisitorType'].map(ubah2).fillna(0).astype(int)

 from sklearn.preprocessing import LabelEncoder
 le = LabelEncoder()

 # labeling revenue
 shop_df['Revenue'] = le.fit_transform(shop_df['Revenue'])
 shop_df['Revenue'].value_counts()

 # labeling weekend
 shop_df['Weekend'] = le.fit_transform(shop_df['Weekend'])
 shop_df['Weekend'].value_counts()

 shop_df.info()

 shop_df.head()

 X = shop_df.iloc[:,:-1]
 X.head()

 y = shop_df["Revenue"]
 y.head()

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

DIMENSIONALITY REDUCTION *ONLY* FOR LINEAR KERNEL

```
47. from sklearn.preprocessing import StandardScaler
48. scaler = StandardScaler()
49. X_scaled = scaler.fit_transform(X)
50.
51. from sklearn.decomposition import PCA
52. pca = PCA(n_components=2)
53. X_pca = pca.fit_transform(X_scaled)
54.
55. import matplotlib.pyplot as plt
56. plt.scatter(X_pca[y==0, 0], X_pca[y==0, 1], c='r', label='class 0')
57. plt.scatter(X_pca[y==1, 0], X_pca[y==1, 1], c='b', label='class 1')
58. plt.xlabel('First Principal Component')
59. plt.ylabel('Second Principal Component')
60. plt.legend()
61. plt.show()
```

SPLITTING THE DATA *ONLY* FOR LINEAR KERNEL

```
62. from sklearn.model_selection import train_test_split
63. X_train, X_test, y_train, y_test = train_test_split(X_pca, y,
    test_size=0.2)
64.
65. from sklearn.svm import SVC
66. svm = SVC(kernel='linear')
67. svm.fit(X_train, y_train)
68.
69. from sklearn.metrics import classification_report
70. y_pred = svm.predict(X_test)
71. print(classification_report(y_test, y_pred))
```

SPLITTING, RESAMPLING, AND BALANCING THE DATA

```
72. from sklearn.model_selection import train_test_split
73. from imblearn.over_sampling import ADASYN
74.
75. adasyn = ADASYN()
76. X_adasyn, y_adasyn = adasyn.fit_resample(X, y)
77.
78. X_train, X_test, y_train, y_test = train_test_split(X_adasyn, y_adasyn,
    test_size = 0.2)
79.
80. print("X_train :", X_train.shape)
81. print("y_train :", y_train.shape)
82. print("X_test  :", X_test.shape)
83. print("y_test  :", y_test.shape)
```

SUPPORT VECTOR MACHINE (SVM)

```
84. from sklearn import svm, metrics
85. from sklearn.svm import SVC
86. from sklearn.metrics import classification_report
87. from sklearn.preprocessing import StandardScaler
88.
```

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

model1 = svm.SVC()
model1.fit(X_train, y_train)

y_pred = model1.predict(X_test)

print(classification_report(y_test, y_pred))

print("Akurasi Training: ", model1.score(X_train, y_train))
print("Akurasi Testing: ", model1.score(X_test, y_test))
```

RANDOMIZEDSEARCHCV ON SVM

```
from sklearn.model_selection import RandomizedSearchCV

parametersRS1 = {'kernel': ['linear', 'rbf', 'sigmoid'],
'C': [0.5, 1, 10, 100],
'gamma': [1, 0.1, 0.01, 0.001]
}

random_searchSVM1 = RandomizedSearchCV(estimator=SVC(),
param_distributions=parametersRS1,
n_jobs=6,
verbose=1,
scoring='accuracy'
)

random_searchSVM1.fit(X_train, y_train)

from sklearn.model_selection import RandomizedSearchCV

parametersRS2 = {'kernel': ['rbf', 'sigmoid'],
'C': np.linspace(0.0001, 100, 100),
'gamma': np.linspace(0.0001, 100, 100)
}

random_searchSVM2 = RandomizedSearchCV(estimator=SVC(random_state=0),
param_distributions=parametersRS2,
n_jobs=6,
verbose=1,
scoring='accuracy'
)

random_searchSVM2.fit(X_train, y_train)
```

GRIDSEARCHCV ON SVM

```
from sklearn.model_selection import GridSearchCV

parametersGS1 = {'kernel': ['linear', 'rbf', 'sigmoid'],
'C': [0.5, 1, 10, 100],
'gamma': [1, 0.1, 0.01, 0.001]
}
```

```
140.
141.grid_searchSVM1 = GridSearchCV(estimator=SVC(),
142.param_grid=parametersGS1,
143.n_jobs=6,
144.verbose=1,
145.scoring='accuracy'
146.)
147.
148.grid_searchSVM1.fit(X_train, y_train)
149.
150.from sklearn.model_selection import GridSearchCV
151.
152.parametersGS2 = {'kernel': ['rbf', 'sigmoid'],
153.'C': np.linspace(0.0001, 100, 100),
154.'gamma': np.linspace(0.0001, 100, 100)
155.}
156.
157.grid_searchSVM2 = GridSearchCV(estimator=SVC(random_state=0),
158.param_grid=parametersGS2,
159.n_jobs=6,
160.verbose=1,
161.scoring='accuracy'
162.)
163.
164.grid_searchSVM2.fit(X_train, y_train)
```

HYPERPARAMETER TUNING EVALUATION FOR SVM

```
165.print(f'Best Score for RandomizedSearchCV with spesific value:
   {random_searchSVM1.best_score_}')
166.print(f'Best Score for RandomizedSearchCV with ranged value:
   {random_searchSVM2.best_score_}')
167.print(f'Best Score for GridSearchCV with spesific value:
   {grid_searchSVM1.best_score_}')
168.print(f'Best Score for GridSearchCV with ranged value:
   {grid_searchSVM2.best_score_}')
169.
170.best_paramsSVM1 = random_searchSVM1.best_estimator_.get_params()
171.print(f'\nBest Parameters:')
172.for param in parametersRS1:
173.print(f'\t{param}: {best_paramsSVM1[param]}')
174.
175.best_paramsSVM2 = random_searchSVM2.best_estimator_.get_params()
176.print(f'\nBest Parameters:')
177.for param in parametersRS2:
178.print(f'\t{param}: {best_paramsSVM2[param]}')
179.
180.best_paramsSVM3 = grid_searchSVM1.best_estimator_.get_params()
181.print(f'\nBest Parameters:')
182.for param in parametersGS1:
183.print(f'\t{param}: {best_paramsSVM3[param]}')
184.
185.best_paramsSVM4 = grid_searchSVM2.best_estimator_.get_params()
186.print(f'Best Parameters:')
187.for param in parametersGS2:
188.print(f'\t{param}: {best_paramsSVM4[param]}')
189.
```

```
# evaluasi
y_pred1 = random_searchSVM1.predict(X_test)
print(f'RandomizedSearchCV with spesific value \n{classification_report(y_test, y_pred1)}')

y_pred2 = random_searchSVM2.predict(X_test)
print(f'RandomizedSearchCV with ranged value \n{classification_report(y_test, y_pred2)}')

y_pred3 = grid_searchSVM1.predict(X_test)
print(f'GridSearchCV with spesific value \n{classification_report(y_test, y_pred3)}')

y_pred4 = grid_searchSVM2.predict(X_test)
print(f'GridSearchCV with ranged value \n{classification_report(y_test, y_pred4)}')
```

STOCHASTIC GRADIENT DESCENT (SGD)

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

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

model2 = SGDClassifier()
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))
```

RANDOMIZEDSEARCHCV ON SGD

```
parametersRS3 = { 'penalty' : ['l1'],
'alpha' : [0.0001, 0.001, 0.01, 0.1, 1, 10],
'max_iter' : np.linspace(1000, 10000, 20),
'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_searchSGD1 = RandomizedSearchCV(estimator = SGDClassifier(),
param_distributions = parametersRS3,
n_jobs = 6,
verbose = 1,
scoring = 'accuracy'
)

random_searchSGD1.fit(X_train, y_train)
```

```
235.
236.parametersRS4 = { 'penalty' : ['l1'],
237.'alpha' : np.linspace(0.0001, 10, 20),
238.'max_iter' : np.linspace(1000, 10000, 20),
239.'loss':    ['hinge',    'log',    'modified_huber',    'squared_hinge',
   'perceptron'],
240.'learning_rate' : ['constant', 'optimal', 'invscaling', 'adaptive'],
241.'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}],
242.'eta0' : np.linspace(1, 100, 20)
243.}
244.
245.random_searchSGD2 = RandomizedSearchCV(estimator = SGDClassifier(),
246.param_distributions = parametersRS4,
247.n_jobs = 6,
248.verbose = 1,
249.scoring = 'accuracy'
250.)
251.
252.random_searchSGD2.fit(X_train, y_train)
```

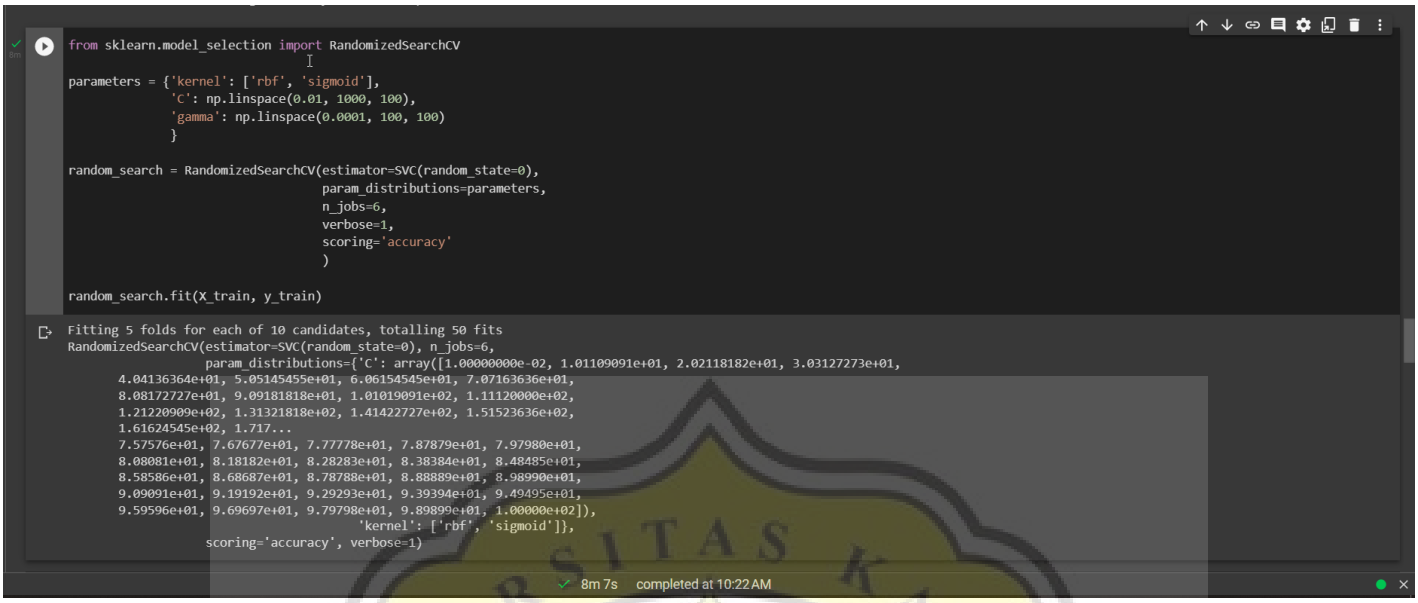
GRIDSEARCHCV ON SGD

```
253.from sklearn.model_selection import GridSearchCV
254.
255.parametersGS3 = { 'penalty' : ['l1'],
256.'alpha' : [0.0001, 0.001, 0.01, 0.1, 1, 10],
257.'max_iter' : np.linspace(1000, 10000, 20),
258.'loss':    ['hinge',    'log',    'modified_huber',    'squared_hinge',
   'perceptron'],
259.'learning_rate' : ['constant', 'optimal', 'invscaling', 'adaptive'],
260.'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}],
261.'eta0' : [1, 10, 100]
262.}
263.
264.grid_searchSGD1 = GridSearchCV(estimator = SGDClassifier(),
265.param_grid = parametersGS3,
266.n_jobs = 6,
267.verbose = 1,
268.scoring = 'accuracy'
269.)
270.
271.grid_searchSGD1.fit(X_train, y_train)
272.
273.from sklearn.model_selection import GridSearchCV
274.
275.parametersGS4 = { 'penalty' : ['l1'],
276.'alpha' : np.linspace(0.0001, 10, 20),
277.'max_iter' : np.linspace(1000, 10000, 20),
278.'loss':    ['hinge',    'log',    'modified_huber',    'squared_hinge',
   'perceptron'],
279.'learning_rate' : ['constant', 'optimal', 'invscaling', 'adaptive'],
280.'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}],
281.'eta0' : np.linspace(1, 100, 20)
282.}
```

```
283.
284.grid_searchSGD2 = GridSearchCV(estimator = SGDClassifier(),
285.param_grid = parametersGS4,
286.n_jobs = 6,
287.verbose = 1,
288.scoring = 'accuracy'
289.)
290.
291.grid_searchSGD2.fit(X_train, y_train)
```

HYPERPARAMETER TUNING EVALUATION FOR SGD

```
292.print(f'Best Score for RandomizedSearchCV with spesific value:
   {random_searchSGD1.best_score_}')
293.print(f'Best Score for RandomizedSearchCV with ranged value:
   {random_searchSGD2.best_score_}')
294.print(f'Best Score for GridSearchCV with spesific value:
   {grid_searchSGD1.best_score_}')
295.print(f'Best Score for GridSearchCV with ranged value:
   {grid_searchSGD2.best_score_}')
296.
297.best_paramsSGD1 = random_searchSGD1.best_estimator_.get_params()
298.print(f'\nBest Parameters:')
299.for param in parametersRS3:
300.print(f'\t{param}: {best_paramsSGD1[param]}')
301.
302.best_paramsSGD2 = random_searchSGD2.best_estimator_.get_params()
303.print(f'\nBest Parameters:')
304.for param in parametersRS3:
305.print(f'\t{param}: {best_paramsSGD2[param]}')
306.
307.best_paramsSGD3 = grid_searchSGD1.best_estimator_.get_params()
308.print(f'\nBest Parameters:')
309.for param in parametersGS3:
310.print(f'\t{param}: {best_paramsSGD3[param]}')
311.
312.best_paramsSGD4 = grid_searchSGD2.best_estimator_.get_params()
313.print(f'Best Parameters:')
314.for param in parametersGS4:
315.print(f'\t{param}: {best_paramsSGD4[param]}')
316.
317.# evaluasi
318.y_pred5 = random_searchSGD1.predict(X_test)
319.print(f'RandomizedSearchCV with spesific value
   \n{classification_report(y_test, y_pred5)}')
320.
321.y_pred6 = random_searchSGD2.predict(X_test)
322.print(f'RandomizedSearchCV with ranged value
   \n{classification_report(y_test, y_pred6)}')
323.
324.y_pred7 = grid_searchSGD1.predict(X_test)
325.print(f'GridSearchCV with spesific value
   \n{classification_report(y_test, y_pred7)}')
326.
327.y_pred8 = grid_searchSGD2.predict(X_test)
328.print(f'GridSearchCV with ranged value \n{classification_report(y_test,
   y_pred8)}')
```



```
from sklearn.model_selection import RandomizedSearchCV

parameters = {'kernel': ['rbf', 'sigmoid'],
              'C': np.linspace(0.01, 1000, 100),
              'gamma': np.linspace(0.0001, 100, 100)
              }

random_search = RandomizedSearchCV(estimator=SVC(random_state=0),
                                   param_distributions=parameters,
                                   n_jobs=6,
                                   verbose=1,
                                   scoring='accuracy'
                                   )

random_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
RandomizedSearchCV(estimator=SVC(random_state=0), n_jobs=6,
                   param_distributions={'C': array([1.00000000e-02, 1.01109091e+01, 2.02118182e+01, 3.03127273e+01,
       4.04136364e+01, 5.05145455e+01, 6.06154545e+01, 7.07163636e+01,
       8.08172727e+01, 9.09181818e+01, 1.01019091e+02, 1.11120000e+02,
       1.21220909e+02, 1.31321818e+02, 1.41422727e+02, 1.51523636e+02,
       1.61624545e+02, 1.717...
       7.57576e+01, 7.67677e+01, 7.77778e+01, 7.87879e+01, 7.97980e+01,
       8.08081e+01, 8.18182e+01, 8.28283e+01, 8.38384e+01, 8.48485e+01,
       8.58586e+01, 8.68687e+01, 8.78788e+01, 8.88889e+01, 8.98990e+01,
       9.09091e+01, 9.19192e+01, 9.29293e+01, 9.39394e+01, 9.49495e+01,
       9.59596e+01, 9.69697e+01, 9.79798e+01, 9.89899e+01, 1.00000e+02]),
                                        'kernel': ['rbf', 'sigmoid']},
                   scoring='accuracy', verbose=1)
```

8m 7s completed at 10:22 AM

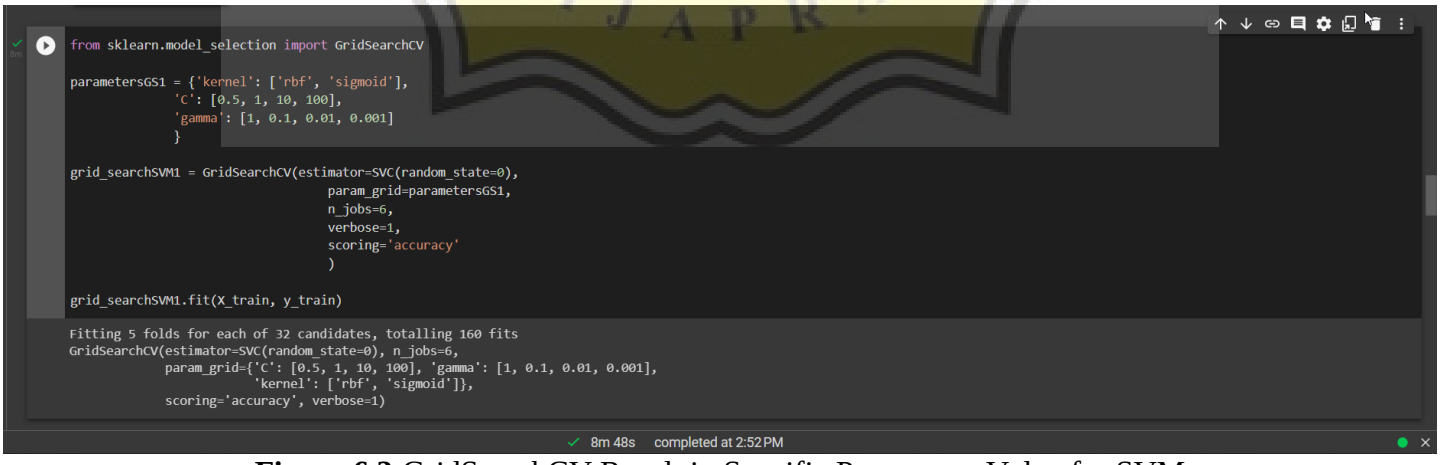
Figure 6.1 RandomizedSearchCV Result in Ranged Parameters Value for SVM

```
from sklearn.model_selection import RandomizedSearchCV

parametersRS1 = {'kernel': ['rbf', 'sigmoid'],
                 'C': [0.5, 1, 10, 100],
                 'gamma': [1, 0.1, 0.01, 0.001]
                 }

random_searchSVM1 = RandomizedSearchCV(estimator=SVC(random_state=0),
                                       param_distributions=parametersRS1,
                                       n_jobs=6,
                                       verbose=1,
                                       scoring='accuracy'
                                       )

random_searchSVM1.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
RandomizedSearchCV(estimator=SVC(random_state=0), n_jobs=6,
                   param_distributions={'C': [0.5, 1, 10, 100],
                                        'gamma': [1, 0.1, 0.01, 0.001],
                                        'kernel': ['rbf', 'sigmoid']},
                   scoring='accuracy', verbose=1)
```

3m 0s completed at 2:42 PM

Figure 6.2 RandomizedSearchCV Result in Specific Parameters Value for SVM



```
from sklearn.model_selection import GridSearchCV

parametersGS1 = {'kernel': ['rbf', 'sigmoid'],
                 'C': [0.5, 1, 10, 100],
                 'gamma': [1, 0.1, 0.01, 0.001]
                 }

grid_searchSVM1 = GridSearchCV(estimator=SVC(random_state=0),
                               param_grid=parametersGS1,
                               n_jobs=6,
                               verbose=1,
                               scoring='accuracy'
                               )

grid_searchSVM1.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
GridSearchCV(estimator=SVC(random_state=0), n_jobs=6,
             param_grid={'C': [0.5, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001],
                         'kernel': ['rbf', 'sigmoid']},
             scoring='accuracy', verbose=1)
```

8m 48s completed at 2:52 PM

Figure 6.3 GridSearchCV Result in Specific Parameters Value for SVM

h

Figure 6.4 GridSearchCV Result in Specific Parameters Value for SVM (failed)

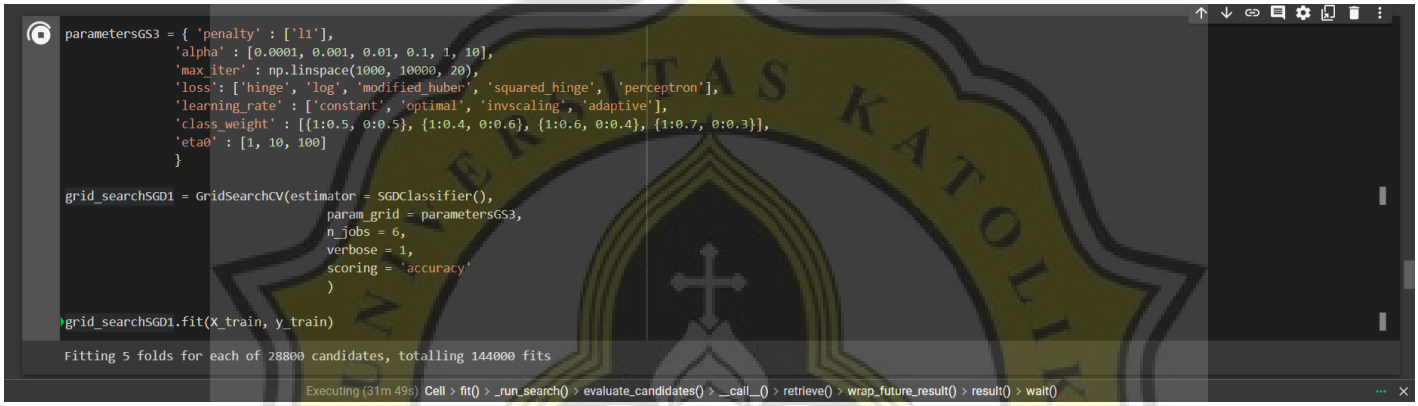


Figure 6.5 GridSearchCV Result in Specific Parameters Value for SGD (failed)

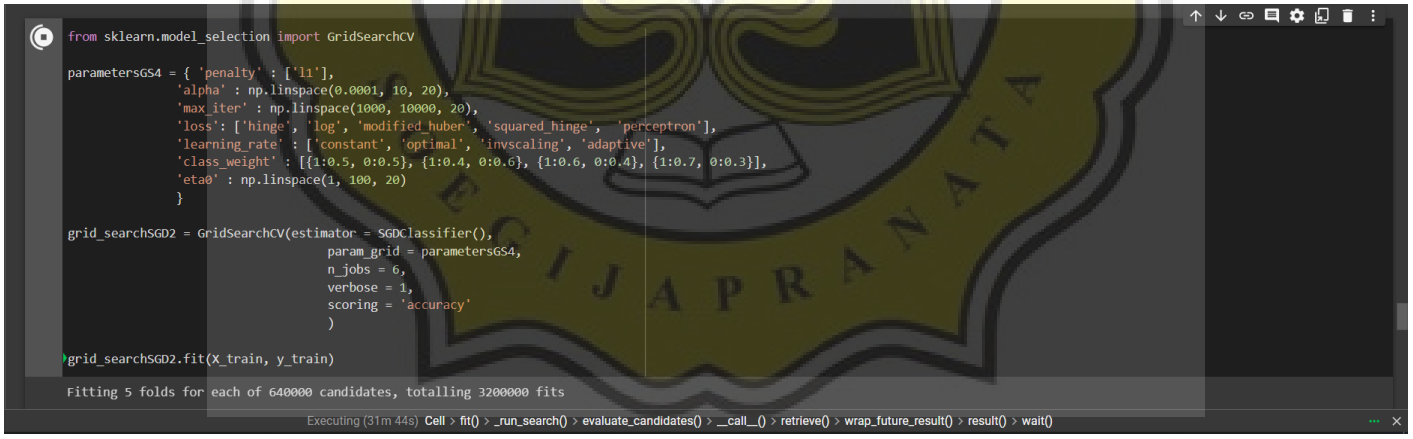
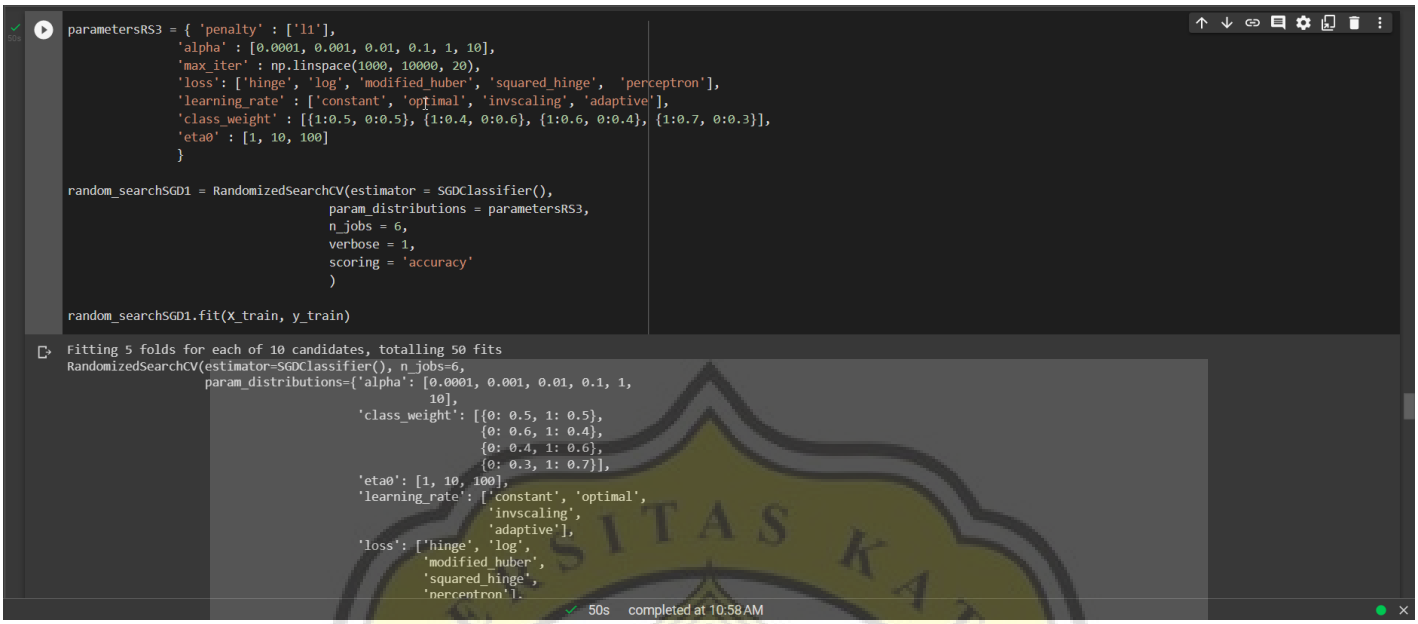


Figure 6.6 GridSearchCV Result in Ranged Parameters Value for SGD (failed)

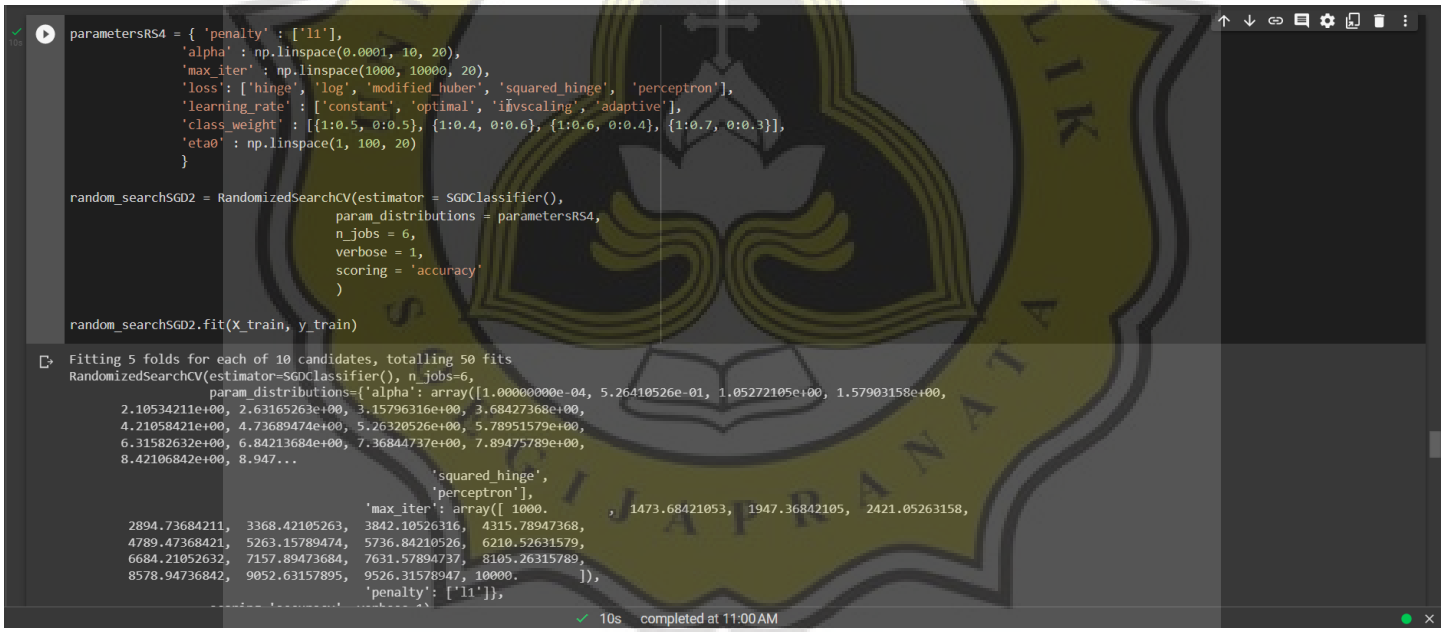


```
parametersRS3 = { 'penalty' : ['l1'],
                'alpha' : [0.0001, 0.001, 0.01, 0.1, 1, 10],
                'max_iter' : np.linspace(1000, 10000, 20),
                'loss': ['hinge', 'log', 'modified_huber', 'squared_hinge',  'perceptron'],
                'learning_rate' : ['constant', '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_searchSGD1 = RandomizedSearchCV(estimator = SGDClassifier(),
                                       param_distributions = parametersRS3,
                                       n_jobs = 6,
                                       verbose = 1,
                                       scoring = 'accuracy'
                                       )

random_searchSGD1.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
RandomizedSearchCV(estimator=SGDClassifier(), n_jobs=6,
                   param_distributions={'alpha': [0.0001, 0.001, 0.01, 0.1, 1,
                                                  10],
                                        'class_weight': [{0: 0.5, 1: 0.5},
                                                         {0: 0.6, 1: 0.4},
                                                         {0: 0.4, 1: 0.6},
                                                         {0: 0.3, 1: 0.7}],
                                        'eta0': [1, 10, 100],
                                        'learning_rate': ['constant', 'optimal',
                                                          'invscaling',
                                                          'adaptive'],
                                        'loss': ['hinge', 'log',
                                                 'modified_huber',
                                                 'squared_hinge',
                                                 'perceptron'],
```

50s completed at 10:58 AM

Figure 6.7 RandomizedSearchCV Result in Specific Parameters Value for SGD



```
parametersRS4 = { 'penalty' : ['l1'],
                'alpha' : np.linspace(0.0001, 10, 20),
                'max_iter' : np.linspace(1000, 10000, 20),
                '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' : np.linspace(1, 100, 20)
                }

random_searchSGD2 = RandomizedSearchCV(estimator = SGDClassifier(),
                                       param_distributions = parametersRS4,
                                       n_jobs = 6,
                                       verbose = 1,
                                       scoring = 'accuracy'
                                       )

random_searchSGD2.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
RandomizedSearchCV(estimator=SGDClassifier(), n_jobs=6,
                   param_distributions={'alpha': array([1.00000000e-04, 5.26410526e-01, 1.05272105e+00, 1.57903158e+00,
       2.10534211e+00, 2.63165263e+00, 3.15796316e+00, 3.68427368e+00,
       4.21058421e+00, 4.73689474e+00, 5.26320526e+00, 5.78951579e+00,
       6.31582632e+00, 6.84213684e+00, 7.36844737e+00, 7.89475789e+00,
       8.42106842e+00, 8.947...
                                           'squared_hinge',
                                           'perceptron'],
                                'max_iter': array([ 1000.        ,  1473.68421053,  1947.36842105,  2421.05263158,
        2894.73684211,  3368.42105263,  3842.10526316,  4315.78947368,
        4789.47368421,  5263.15789474,  5736.84210526,  6210.52631579,
        6684.21052632,  7157.89473684,  7631.57894737,  8105.26315789,
        8578.94736842,  9052.63157895,  9526.31578947, 10000.        ]),
                                'penalty': ['l1']},
```

10s completed at 11:00 AM

Figure 6.8 RandomizedSearchCV Result in Ranged Parameters Value for SGD

```python
# evaluasi
y_pred1 = random_searchSVM1.predict(X_test)
print(f'RandomizedSearchCV with spesific value \n{classification_report(y_test, y_pred1)}')

y_pred2 = random_searchSVM2.predict(X_test)
print(f'RandomizedSearchCV with ranged value \n{classification_report(y_test, y_pred2)}')

y_pred3 = grid_searchSVM1.predict(X_test)
print(f'GridSearchCV with spesific value \n{classification_report(y_test, y_pred3)}')

# y_pred4 = grid_searchSVM2.predict(X_test)
# print(f'GridSearchCV with ranged value \n{classification_report(y_test, y_pred4)}')
```

```
RandomizedSearchCV with spesific value
              precision    recall  f1-score   support

           0       0.91      0.89      0.90      2055
           1       0.89      0.91      0.90      2111

    accuracy                           0.90      4166
   macro avg       0.90      0.90      0.90      4166
weighted avg       0.90      0.90      0.90      4166

RandomizedSearchCV with ranged value
              precision    recall  f1-score   support

           0       0.79      0.89      0.84      2055
           1       0.88      0.78      0.82      2111

    accuracy                           0.83      4166
   macro avg       0.84      0.83      0.83      4166
weighted avg       0.84      0.83      0.83      4166

GridSearchCV with spesific value
              precision    recall  f1-score   support

           0       0.91      0.89      0.90      2055
           1       0.89      0.91      0.90      2111

    accuracy                           0.90      4166
   macro avg       0.90      0.90      0.90      4166
weighted avg       0.90      0.90      0.90      4166
```

Figure 6.9 Classification Report for SVM in Hyperparameter Tuning with RandomizedSearchCV

```python
[30] print(f'Best Score for RandomizedSearchCV with spesific value: {random_searchSVM1.best_score_}')
     print(f'Best Score for RandomizedSearchCV with ranged value: {random_searchSVM2.best_score_}')
     print(f'Best Score for GridSearchCV with spesific value: {grid_searchSVM1.best_score_}')
     # print(f'Best Score for GridSearchCV with ranged value: {grid_searchSVM2.best_score_}')

     best_paramsSVM1 = random_searchSVM1.best_estimator_.get_params()
     print(f'\nBest Parameters:')
     for param in parametersRS1:
         print(f'\t{param}: {best_paramsSVM1[param]}')

     best_paramsSVM2 = random_searchSVM2.best_estimator_.get_params()
     print(f'\nBest Parameters:')
     for param in parametersRS2:
         print(f'\t{param}: {best_paramsSVM2[param]}')

     best_paramsSVM3 = grid_searchSVM1.best_estimator_.get_params()
     print(f'\nBest Parameters:')
     for param in parametersGS1:
         print(f'\t{param}: {best_paramsSVM3[param]}')

     # best_paramsSVM4 = grid_searchSVM2.best_estimator_.get_params()
     # print(f'Best Parameters:')
     # for param in parametersGS2:
     #     print(f'\t{param}: {best_paramsSVM4[param]}')
```

```
Best Score for RandomizedSearchCV with spesific value: 0.8887957163063245
Best Score for RandomizedSearchCV with ranged value: 0.8306422839162668
Best Score for GridSearchCV with spesific value: 0.8887957163063245

Best Parameters:
        kernel: rbf
        C: 100
        gamma: 0.1

Best Parameters:
        kernel: sigmoid
        C: 82.8283
        gamma: 0.0001

Best Parameters:
        kernel: rbf
        C: 100
        gamma: 0.1
```

Figure 6.10 Best Parameters Result for SVM in RandomizedSearchCV

```
# evaluasi
y_pred5 = random_searchSGD1.predict(X_test)
print(f'RandomizedSearchCV with spesific value \n{classification_report(y_test, y_pred5)}')

y_pred6 = random_searchSGD2.predict(X_test)
print(f'RandomizedSearchCV with ranged value \n{classification_report(y_test, y_pred6)}')

# y_pred7 = grid_searchSGD1.predict(X_test)
# print(f'GridSearchCV with spesific value \n{classification_report(y_test, y_pred7)}')

# y_pred8 = grid_searchSGD2.predict(X_test)
# print(f'GridSearchCV with ranged value \n{classification_report(y_test, y_pred8)}')
```

```
RandomizedSearchCV with spesific value
              precision    recall  f1-score   support

           0       0.86      0.79      0.82      2079
           1       0.81      0.87      0.84      2087

    accuracy                           0.83      4166
   macro avg       0.83      0.83      0.83      4166
weighted avg       0.83      0.83      0.83      4166

RandomizedSearchCV with ranged value
              precision    recall  f1-score   support

           0       0.76      0.92      0.83      2079
           1       0.90      0.71      0.79      2087

    accuracy                           0.81      4166
   macro avg       0.83      0.81      0.81      4166
weighted avg       0.83      0.81      0.81      4166
```

Figure 6.11 Classification Report for SGD in Hyperparameter Tuning with RandomizedSearchCV

```
print(f'Best Score for RandomizedSearchCV with spesific value: {random_searchSGD1.best_score_}')
print(f'Best Score for RandomizedSearchCV with ranged value: {random_searchSGD2.best_score_}')
# print(f'Best Score for GridSearchCV with spesific value: {grid_searchSGD1.best_score_}')
# print(f'Best Score for GridSearchCV with ranged value: {grid_searchSGD2.best_score_}')

best_paramsSGD1 = random_searchSGD1.best_estimator_.get_params()
print(f'\nBest Parameters:')
for param in parametersRS3:
    print(f'\t{param}: {best_paramsSGD1[param]}')

best_paramsSGD2 = random_searchSGD2.best_estimator_.get_params()
print(f'\nBest Parameters:')
for param in parametersRS3:
    print(f'\t{param}: {best_paramsSGD2[param]}')

best_paramsSGD3 = grid_searchSGD1.best_estimator_.get_params()
print(f'\nBest Parameters:')
for param in parametersGS3:
    print(f'\t{param}: {best_paramsSGD3[param]}')

# best_paramsSGD4 = grid_searchSGD2.best_estimator_.get_params()
# print(f'Best Parameters:')
# for param in parametersGS4:
#     print(f'\t{param}: {best_paramsSGD4[param]}')
```

```
Best Score for RandomizedSearchCV with spesific value: 0.8308222659000594
Best Score for RandomizedSearchCV with ranged value: 0.8056774464961503

Best Parameters:
        penalty: l1
        alpha: 0.001
        max_iter: 7631.578947368421
        loss: log
        learning_rate: adaptive
        class_weight: {1: 0.6, 0: 0.4}
        eta0: 1

Best Parameters:
        penalty: l1
        alpha: 0.0001
        max_iter: 5263.157894736842
        loss: modified_huber
        learning_rate: adaptive
        class_weight: {1: 0.4, 0: 0.6}
        eta0: 27.052631578947366
```

Figure 6.12 Best Parameters Result for SGD in RandomizedSearchCV

```python
from sklearn import svm, metrics
from sklearn.svm import SVC
from sklearn.metrics import classification_report

model1 = svm.SVC()
model1.fit(X_train, y_train)
# model1.fit(X_train_reduced, y_train)
# model1.fit(X_train_resampled, y_train_resampled)

y_pred = model1.predict(X_test)
# y_pred = model1.predict(X_test_reduced)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2081
           1       1.00      0.02      0.04       385

    accuracy                           0.85      2466
   macro avg       0.92      0.51      0.48      2466
weighted avg       0.87      0.85      0.78      2466
```

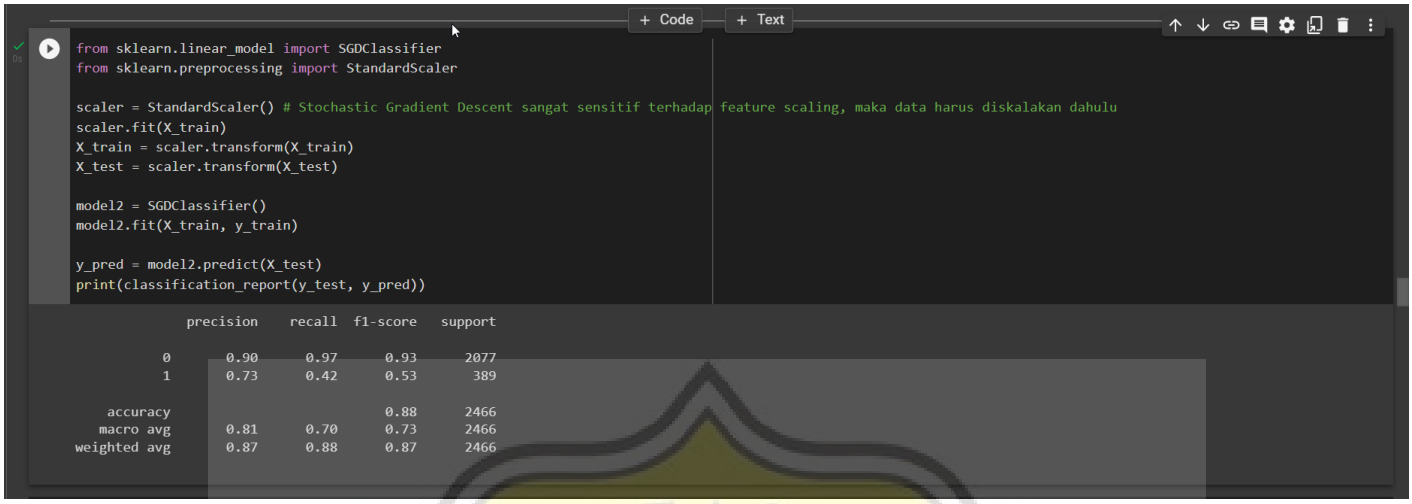
Figure 6.13 Before Data Balancing with ADASYN on SVM

```python
from sklearn import svm, metrics
from sklearn.svm import SVC
from sklearn.metrics import classification_report

model1 = svm.SVC()
model1.fit(X_train, y_train)
# model1.fit(X_train_reduced, y_train)
# model1.fit(X_train_resampled, y_train_resampled)

y_pred = model1.predict(X_test)
# y_pred = model1.predict(X_test_reduced)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.66      0.70      2150
           1       0.67      0.75      0.71      2016

    accuracy                           0.70      4166
   macro avg       0.71      0.71      0.70      4166
weighted avg       0.71      0.70      0.70      4166
```

Figure 6.14 After Data Balancing with ADASYN on SVM

m



```
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()
model2.fit(X_train, y_train)

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

```
              precision    recall  f1-score   support

           0       0.90      0.97      0.93      2077
           1       0.73      0.42      0.53       389

    accuracy                           0.88      2466
   macro avg       0.81      0.70      0.73      2466
weighted avg       0.87      0.88      0.87      2466
```

Figure 6.15 Before Data Balancing with ADASYN on SGD

```
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()
model2.fit(X_train, y_train)

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

```
              precision    recall  f1-score   support

           0       0.81      0.90      0.85      2150
           1       0.88      0.77      0.82      2016

    accuracy                           0.84      4166
   macro avg       0.84      0.83      0.83      4166
weighted avg       0.84      0.84      0.83      4166
```

Figure 6.16 After Data Balancing with ADASYN on SGD

n

PAPER NAME

TA-18.K1.0011.docx

WORD COUNT	CHARACTER COUNT
7800 Words	41642 Characters
PAGE COUNT	FILE SIZE
19 Pages	30.8KB
SUBMISSION DATE	REPORT DATE
Jan 23, 2023 9:59 AM GMT+7	Jan 23, 2023 9:59 AM GMT+7

15% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 15% Internet database
- Crossref database
- 10% Submitted Works database
- 9% Publications database
- Crossref Posted Content database

Excluded from Similarity Report

- Bibliographic material
- Cited material
- Manually excluded text blocks
- Quoted material
- Small Matches (Less then 10 words)