CHAPTER 5
IMPLEMENTATION AND RESULTS

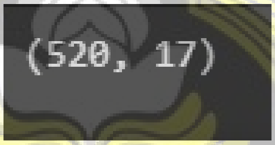
5.1. Implementation

The implementation was done using Python programming language on *Google Colaboratory*. This subchapter is about how the code in the models from loading the dataset , preparing the dataset , training the dataset with both algorithms ,analyzing the results and lastly comparing the results of both algorithms.

First, import the necessary libraries , set the seed to 42, so we can get the same result every time we run the code, and then load the dataset and do the data preprocessing

```
1. data = pd.read_csv('/content/drive/My Drive/TA/Dataset/diabetes.csv')
2. data.shape
```

Line 1 is by using the pandas library to load and read the dataset that has data type of csv. And the line 2 is for check the dataset shape, and the output as below



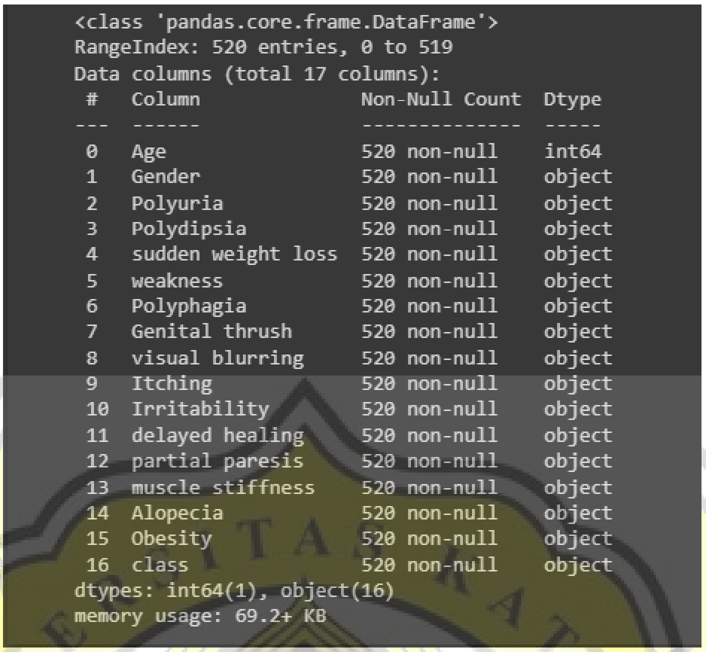
```
(520, 17)
```

Figure 5.1 Dataset shape

Figure 5.1 shows that the dataset has 520 rows and 17 columns or the 520 record and 17 attributes. Next, we need to check is there any missing values in the dataset by using code as below

```
3. data.info()
```

The output of line 3 as below.



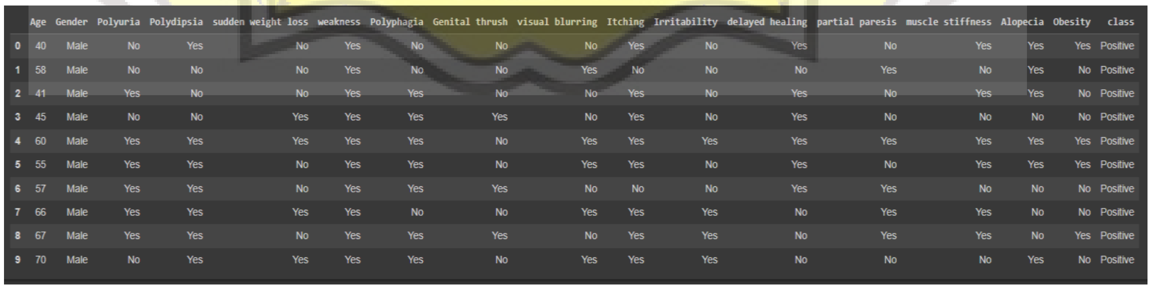
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Age                 520 non-null    int64 
 1   Gender              520 non-null    object
 2   Polyuria            520 non-null    object
 3   Polydipsia          520 non-null    object
 4   sudden weight loss  520 non-null    object
 5   weakness            520 non-null    object
 6   Polyphagia          520 non-null    object
 7   Genital thrush      520 non-null    object
 8   visual blurring     520 non-null    object
 9   Itching             520 non-null    object
 10  Irritability        520 non-null    object
 11  delayed healing     520 non-null    object
 12  partial paresis     520 non-null    object
 13  muscle stiffness    520 non-null    object
 14  Alopecia            520 non-null    object
 15  Obesity             520 non-null    object
 16  class               520 non-null    object
dtypes: int64(1), object(16)
memory usage: 69.2+ KB
```

Figure 5.2 Dataset Information

Figure 5.2 shows how many non-null or not NaN and not empty rows for each column. The data shows that each column has the same amount of data which is 520 rows non-null, and as for the datatype of "Age" is int64 and for the rest is object. And as we can know from Figure 5.1 that the dataset has 520 rows and from Figure 5.2 shows that each column has 520 rows of non null , so there are no missing values in the dataset.

```
4. data.head(10)
```

Line 4 is for showing the head of the dataset, the parameter "10" can be changed depending on how many we want to, for example here we use 10 to show the top 10 record of the dataset, the, and the output as below.



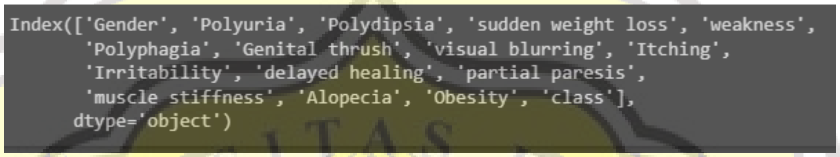
	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	muscle stiffness	Alopecia	Obesity	class
0	40	Male	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes	Yes	Yes	Positive
1	58	Male	No	No	No	Yes	No	No	Yes	No	No	No	Yes	No	Yes	No	Positive
2	41	Male	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No	Positive
3	45	Male	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No	Positive
4	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Positive
5	55	Male	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Positive
6	57	Male	Yes	Yes	No	Yes	Yes	Yes	No	No	No	Yes	Yes	No	No	No	Positive
7	66	Male	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No	No	Positive
8	67	Male	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Positive
9	70	Male	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	No	Yes	No	Positive

Figure 5.3 The Top 10 of the Dataset

Next step is data processing, label encoder was used for one hot encoding to change the value “yes”, “positive”, “male” into 1 and “no”, “negative”, “female” into 0 [16].

```
5. labels = data.columns[1:]
6. labels
```

Line 5 is done for excluding the first columns, in this case its “Age” because “Age” data type is in int64, and we change the attribute from “Gender” to “class”. the [1:] indicates the columns after the first columns which is the “Age”, and the output as below.



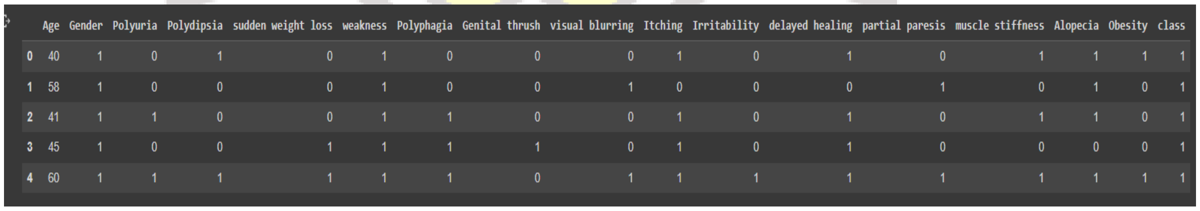
```
Index(['Gender', 'Polyuria', 'Polydipsia', 'sudden weight loss', 'weakness',
       'Polyphagia', 'Genital thrush', 'visual blurring', 'Itching',
       'Irritability', 'delayed healing', 'partial paresis',
       'muscle stiffness', 'Alopecia', 'Obesity', 'class'],
      dtype='object')
```

Figure 5.4 Labels

Next, we do label encoding to change the non numeric values into numeric values [16] , [17].

```
7. for i in labels:
8.             data[i].loc[data[i].isin(['Yes'])] = 1
9.             data[i].loc[data[i].isin(['No'])] = 0
10.             data[i].loc[data[i].isin(['Positive'])] = 1
11.             data[i].loc[data[i].isin(['Negative'])] = 0
12.             data[i].loc[data[i].isin(['Male'])] = 1
13.             data[i].loc[data[i].isin(['Female'])] = 0
14.             data[i] = data[i].astype(int)
15. data.head(10)
```

In line 7, “ i ” indicates each value in variable “labels”, and loop from the first value in “labels” until the last value. Line 8 to 13 is to change the values into either 1 or 0, so where data in label “i” if its value is “Yes”, it will be changed into 1 and so on as the code above. Line 14 is to change the datatype from object into int. And the output is as below.



	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	muscle stiffness	Alopecia	Obesity	class
0	40	1	0	1	0	1	0	0	0	1	0	1	0	1	1	1	1
1	58	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0	1
2	41	1	1	0	0	1	1	0	0	1	0	1	0	1	1	0	1
3	45	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	1
4	60	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1

Figure 5.5 Top 10 of the Dataset After The Label Encoding

After the label encoding is done, then splitting the dataset into 2 variables X and Y, where Y is the target or labels which the "class" attribute.

```
16. X = data.drop('class',axis=1)
17. Y = data['class'] #target/labels
```

Line 1 is for defining X where drop the 'class' and the axis is 1 or columns [23], so the attribute "class" will not be included in the the X set, and as for the line 2 is for defining Y equal to the "class" attribute in the dataset so the X contains all the data except for "class", while Y contain only the "class" attribute.

```
18. def cm(target, prediction):
19.     TP = 0
20.     TN = 0
21.     FP = 0
22.     FN = 0
23.     j = 0
24.     for i in enumerate(target):
25.         if i[1] == 1:
26.             if prediction[j] == 1:
27.                 TP += 1
28.             else:
29.                 FN += 1
30.         elif i[1] == 0:
31.             if prediction[j] == 1:
32.                 FP += 1
33.             else:
34.                 TN += 1
35.         j += 1
36.     return TP, TN, FP, FN
```

The code above is to define a function to calculate the confusion matrix. Line 19 to 23 is to declare variables to save the value. Line 24 looping for every value in target (the real value). Line 25 to 29 is if the target or the real value is 1 and the prediction is 1 then True Positive is increased by 1, and if the prediction is 0 then the False Negative is increased by 1. Line 30 to 34 is if the target is 0 and the prediction is 1 the False Positive is increased by 1, and if the prediction is 0 the True Negative is increased by 1. Line 35 is to add 1 into the "j" for the index of the prediction. Line 36 is to return the values.

```
37. def result(target, prediction):
38.     TP, TN, FP, FN = cm( target, prediction)
39.     accuracy = (TP+TN)/(TP+TN+FP+FN)
40.     recall = TP/(TP+FN)
41.     precision = TP/(TP+FP)
42.     f1 = 2*((precision*recall)/(precision+recall))
43.     return accuracy, recall, precision, f1
```

The code above is to define a function to calculate the result of the models, later it will be used for the analysis of the comparison between the two algorithms. Line 37 is to define the function, line 38 is to call the function cm to calculate the matrix by using the parameter target and the prediction result and save the result into values TP, TN, FP, FN. Line 39 to calculate the recall, precision, f1 score and the accuracy of the model. Line 22 is to return the results.

After splitting the dataset into X and Y, then the dataset is split into the training and testing set for both algorithms.

```
44. def split(testsize, X, Y):
45.     print('Train set = ', (100-testsize),'% , Test set = ', testsize,
            '%')
46.         testsize = testsize/100
47.         X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
             test_size=testsize, random_state=seed)
48.     return X_train, X_test, Y_train, Y_test
```

Line 44 is to define a function to split the dataset into X_train, Y_train for the train set, and X_test, Y_test. Line 47 is to call the library to split the dataset with random state , so every time we run the code the result will be the same. Line 48 is to return the X_train, X_test, Y_train, Y_test.

After the data preprocessing is done, we define the models, so the first one is Artificial Neural Network with the design that has been explained in the previous chapter. The codes below are the first ann model with 3 hidden layers.

```
49. def ann_model1(X_train, Y_train, X_test, Y_test):
50.     tf.keras.utils.set_random_seed(seed)
51.     ann1 = Sequential()
52.     ann1.add(Dense(units=      200,      kernel_initializer='uniform',
    activation
                  = 'relu', input_dim=16))
53.     ann1.add(Dense(units= 200, kernel_initializer='uniform', activation
           = 'relu'))
54.     ann1.add(Dense(units= 150, kernel_initializer='uniform', activation
           = 'relu'))
55.     ann1.add(Dense(units = 1, kernel_initializer='uniform', activation
   =
          'sigmoid')) #output
56.     opt = keras.optimizers.Adam(learning_rate=0.001)
57.     ann1.compile(optimizer = opt, loss = 'binary_crossentropy', metrics
           = ['accuracy'])
58.     ann1.fit(X_train, Y_train, batch_size = 500, epochs = 500,
           verbose=0)
59.     ann_pred_test = ann1.predict(X_test)
60.     ann_pred_test=ann_pred_test.round()
61.     ann_accuracy, ann_recall, ann_precision, ann_f1 = result(Y_test,
           ann_pred_test)
62.     return ann_accuracy, ann_recall, ann_precision, ann_f1
```

Line 49 is to define a function to call the model and return the result. Line 50 is to set the random state to a fixed state so the result will be the same as we run it every time. Line 51 initiates the model as sequential, line 52 to 55 is to define the layers, line 52 defines the first layer with 200 neurons and the activation function is ReLu and with the 16 nodes for input layers. Line 53 is the second layer with 200 neurons, line 54 is defining the third layer with 150 neurons and both second and third layer using the same activation function as the first layer. Line 55 is to define the output layer with sigmoid activation function. Line 56 is to define the optimizer of the model which is Adam Optimizer with 0.0001 of learning rate. Line 57 is to compile the model or apply the optimizer to the model using accuracy metrics and binary cross entropy loss. Line 58 is to train the dataset with the model with 500 of batch size and 500 epochs, the verbose set to 0 is to not show the training log. Line 59 is to save the result of prediction, line 60 to round the values of the prediction result. Line 61 is to call the result function to calculate the accuracy, recall, precision, f1 score and save it into variables. Line 62 is to return the results. The codes below are the second ANN model.

```
63. def ann_model2(X_train, Y_train, X_test, Y_test):
64.     tf.keras.utils.set_random_seed(seed)
65.     ann2 = Sequential()
66.     ann2.add(Dense(units=      200,      kernel_initializer='uniform',
   activation
```

```
          = 'relu', input_dim=16))
67.     ann2.add(Dense(units= 200, kernel_initializer='uniform', activation
          = 'relu'))
68.     ann2.add(Dense(units= 200, kernel_initializer='uniform', activation
          = 'relu'))
69.     ann2.add(Dense(units= 150, kernel_initializer='uniform', activation
          = 'relu'))
70.     ann2.add(Dense(units = 1, kernel_initializer='uniform', activation
   =
          'sigmoid'))
71.     opt = keras.optimizers.Adam(learning_rate=0.001)
72.     ann2.compile(optimizer = opt, loss = 'binary_crossentropy', metrics
          = ['accuracy'])
73.     ann2.fit(X_train, Y_train, batch_size = 500, epochs = 500,
          verbose=0)
74.     ann_pred_test2 = ann2.predict(X_test)
75.     ann_pred_test2=ann_pred_test2.round()
76.     ann_accuracy2, ann_recall2, ann_precision2, ann_f12 = result(Y_test,
              ann_pred_test2)
77.     return ann_accuracy2, ann_recall2, ann_precision2, ann_f12
```

Line 64 is to set the random state to a fixed state so the result will be the same as we run it every time. Line 65 initiates the model as sequential, line 66 is to define the first layer with 200 neurons and the activation function is ReLu and with the 16 nodes for input layers. Line 67 is the second layer with 200 neurons, line 68 is defining the third layer with 200 neurons, line 69 is defining the fourth layer with 150 neurons, all hidden layers are using the same activation function as the first layer. Line 70 is to define the output layer with sigmoid activation function. Line 71 is to define the optimizer of the model which is Adam Optimizer with 0.0001 of learning rate. Line 73 is to compile the model or apply the optimizer to the model using accuracy metrics and binary cross entropy loss. Line 73 is to train the dataset with the model with 500 of batch size and 500 epochs, the verbose set to 0 is to not show the training log. Line 74 is to save the result of prediction, line 76 to round the values of the prediction result. Line 76 is to call the result function to calculate the accuracy, recall, precision, f1 score and save it into variables. Line 77 is to return the results. The codes below are the third ANN model.

```
78. def ann_model3(X_train, Y_train, X_test, Y_test):
79.    tf.keras.utils.set_random_seed(seed)
```

```
80.    ann3 = Sequential()
81.    ann3.add(Dense(units= 200, kernel_initializer='uniform',  activation
   = 'relu', input_dim=16))
82.    ann3.add(Dense(units= 200, kernel_initializer='uniform', activation
   = 'relu'))
83.    ann3.add(Dense(units= 150, kernel_initializer='uniform', activation
   = 'relu'))
84.    ann3.add(Dense(units= 150, kernel_initializer='uniform', activation
   = 'relu'))
85.    ann3.add(Dense(units= 150, kernel_initializer='uniform', activation
   = 'relu'))
86.    ann3.add(Dense(units = 1, kernel_initializer='uniform', activation =
   'sigmoid')) #output
87.     opt = keras.optimizers.Adam(learning_rate=0.001)
88.    ann3.compile(optimizer = opt, loss = 'binary_crossentropy', metrics
   = ['accuracy'])
89.    ann3.fit(X_train, Y_train, batch_size = 500, epochs = 500, verbose=0)
90.    ann_pred_test3 = ann3.predict(X_test)
91.    ann_pred_test3=ann_pred_test3.round()
92.    ann_accuracy3, ann_recall3, ann_precision3, ann_f13 = result(Y_test,
   ann_pred_test3)
93.    return ann_accuracy3, ann_recall3, ann_precision3, ann_f13
```

Line 79 is to set the random state to a fixed state so the result will be the same as we run it every time. Line 80 initiates the model as sequential, line 81 is to define the first layer with 200 neurons and the activation function is ReLu and with the 16 nodes for input layers. Line 82 is to define the second and line 83 to 85 is to define the third to fifth layer with 150 neurons, all hidden layers are using the same activation function as the first layer. Line 86 is to define the output layer with sigmoid activation function. Line 87 is to define the optimizer of the model which is Adam Optimizer with 0.0001 of learning rate. Line 88 is to compile the model or apply the optimizer to the model using accuracy metrics and binary cross entropy loss. Line 89 is to train the dataset with the model with 500 of batch size and 500 epochs, the verbose set to 0 is to not show the training log. Line 90 is to save the result of prediction, line 91 to round the values of the prediction result. Line 92 is to call the result function to calculate the accuracy, recall, precision, f1 score and save it into variables. Line 93 is to return the results. The codes below are the fourth ANN model.

```
94. def ann_model4(X_train, Y_train, X_test, Y_test):
95.     tf.keras.utils.set_random_seed(seed)
96.     ann4 = Sequential()
```

```
97.     ann4.add(Dense(units=      200,      kernel_initializer='uniform',
   activation = 'relu', input_dim=16))
98.    ann4.add(Dense(units= 200, kernel_initializer='uniform', activation
   = 'relu'))
99. ann4.add(Dense(units= 200, kernel_initializer='uniform', activation =
   'relu'))
100.ann4.add(Dense(units= 150, kernel_initializer='uniform', activation =
   'relu'))
101.ann4.add(Dense(units= 150, kernel_initializer='uniform', activation =
   'relu'))
102.ann4.add(Dense(units= 150, kernel_initializer='uniform', activation =
   'relu'))
103.ann4.add(Dense(units = 1, kernel_initializer='uniform', activation =
   'sigmoid')) #output
104.opt = keras.optimizers.Adam(learning_rate=0.001)
105.ann4.compile(optimizer = opt, loss = 'binary_crossentropy', metrics =
   ['accuracy'])
106.ann4.fit(X_train, Y_train, batch_size = 500, epochs = 500, verbose=0)
107.ann_pred_test4 = ann4.predict(X_test)
108.ann_pred_test4=ann_pred_test4.round()
109.ann_accuracy4, ann_recall4, ann_precision4, ann_f14 = result(Y_test,
   ann_pred_test4)
110.return ann_accuracy4, ann_recall4, ann_precision4, ann_f14
```

Line 95 is to set the random state to a fixed state so the result will be the same as we run it every time. Line 97 to 99 is to define the first layer to third layer with 200 neurons and the activation function is ReLu and with the 16 nodes for input layers. Line 100 and 102 is defining the fourth to sixth layer with 150 neurons. The rest is the same as the ann models before, it has the same optimizer, learning rate, batch , epochs, predict with the models, calculate the accuracy, recall, precision , f1 score and return the results.

Next, we define the XGBoost model, the first xgboost model is with the default parameters [17] . The code below is to define the first xgboost model.

```
111.def xgb_model1(X_train, Y_train, X_test, Y_test):
112.      xgb_clf = xgb.XGBClassifier(seed=seed, max_depth=6, gamma=0,
           n_estimators=100, learning_rate=0.3, colsample_bytree=1)
113.     xgb_model = xgb_clf.fit(X_train, Y_train)
114.     xgb_pred = xgb_model.predict(X_test)
115.     xgb_accuracy, xgb_recall, xgb_precision, xgb_f1 = result(Y_test,
              xgb_pred)
```

```
116.    return xgb_accuracy, xgb_recall, xgb_precision, xgb_f1
```

Line 112 is to define the model with the default parameters, line 113 is to train the model with the training set. Line 114 to 116 is to calculate the results and return the results.

Next we define the second model with hyperparameter tuning with bayesian optimization.

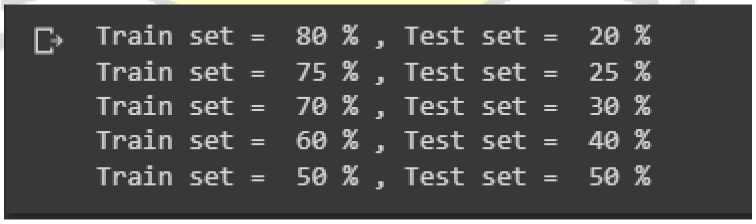
```
117.def   xgb_tune(max_depth,   gamma,   n_estimators   ,learning_rate,
   min_child_weight,  colsample_bytree):
118. cv = KFold(n_splits=10, random_state=seed, shuffle=True)
119. estimators  =  xgb.XGBClassifier(seed=seed,  max_depth=int(max_depth),
   gamma=gamma,                           n_estimators=int(n_estimators),
   learning_rate=learning_rate, colsample_bytree=colsample_bytree)
120. result = cross_val_score(estimators, X, Y, cv=cv, scoring='roc_auc')
121. result = result.mean()
122. return result
```

Line 117 is to define the function to set the estimator for bayesian optimization with the parameters, line 118 is to define the 10-fold with random state set for fixed state, line 119 is to define the estimators for the cross validation. Line 120 is to do cross validation with 10 fold validation and with the estimators that we have already defined before and save the results into the 'result' variable. Line 121 to get the mean of the cross validation result, line 122 is to return the result, the result will be used in the Bayesian optimization.

Next set is we run the training by calling the function that we have defined before.

```
123.X_train, X_test, Y_train, Y_test = split(20, X, Y)
124.X_train2, X_test2, Y_train2, Y_test2 = split(25, X, Y)
125.X_train3, X_test3, Y_train3, Y_test3 = split(30, X, Y)
126.X_train4, X_test4, Y_train4, Y_test4 = split(40, X, Y)
127.X_train5, X_test5, Y_train5, Y_test5 = split(50, X, Y)
```

Line 123 to 127 is to call the split function to split the dataset with the desired portions, and the output is as below.

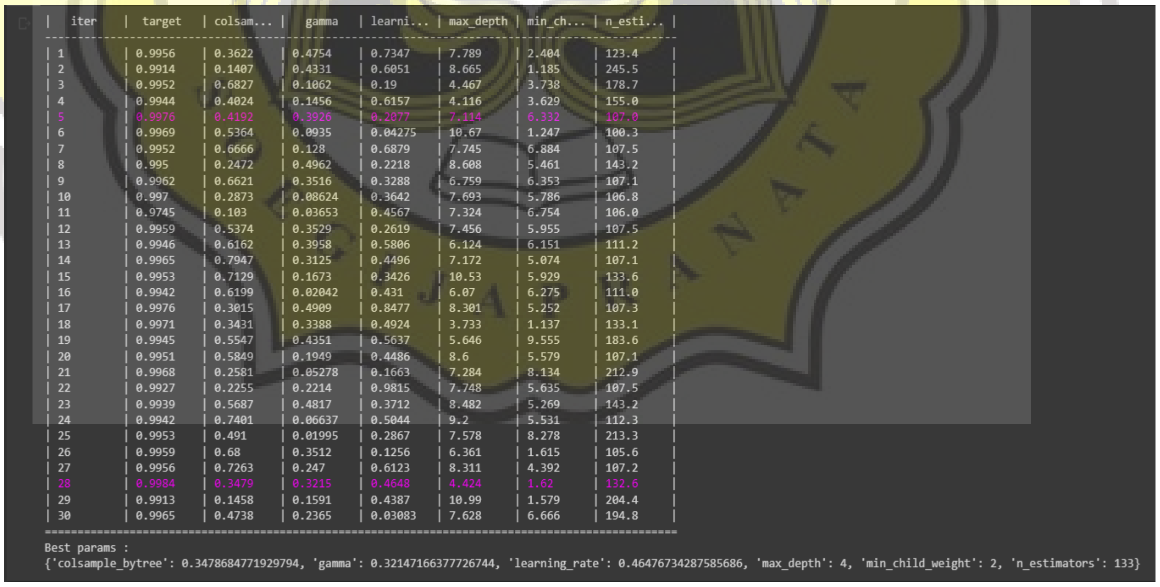


```
Train set =  80 % , Test set =  20 %
Train set =  75 % , Test set =  25 %
Train set =  70 % , Test set =  30 %
Train set =  60 % , Test set =  40 %
Train set =  50 % , Test set =  50 %
```

Figure 5.6 Dataset Splitting

```
128. tune = BayesianOptimization(xgb_tune, {'max_depth': (3, 11),
129.                               'gamma': (1e-09, 0.5),
130.                               'n_estimators': (100, 250),
131.                               'learning_rate': (0.01,1.0),
132.                               'min_child_weight' : (1, 10),
133.                               'colsample_bytree' : (0.1, 0.8),
134.                           }, random_state=seed)
135. tune.maximize()
136. parameters = tune.max['params']
137. parameters['max_depth']= round(parameters['max_depth'])
138. parameters['n_estimators']= round(parameters['n_estimators'])
139. parameters['min_child_weight']= round(parameters['min_child_weight'])
140. print("Best params :")
141. print(parameters)
```

Line 128 is to define the bayesian optimization with the function xgb_tune as the estimators with the search spaces for the hyperparameters, and using fixed random state. Line 135 to run the optimization, line 136 to get the parameters with the best score or to get the optimal parameters. Line 137 to 139 to round the values so the values are rounded to the nearest integer. Line 140 and 141 is to print the parameters, and the output is as below.



```
|   iter    |  target   | colsam... |   gamma   | learni... | max_depth | min_ch... | n_esti... |
-------------------------------------------------------------------------------------------------
| 1         | 0.9956    | 0.3622    | 0.4754    | 0.7347    | 7.789     | 2.404     | 123.4     |
| 2         | 0.9914    | 0.1407    | 0.4331    | 0.6051    | 8.665     | 1.185     | 245.5     |
| 3         | 0.9952    | 0.6827    | 0.1062    | 0.19      | 4.467     | 3.738     | 178.7     |
| 4         | 0.9944    | 0.4024    | 0.1456    | 0.6157    | 4.116     | 3.629     | 155.0     |
| 5         | 0.9976    | 0.4192    | 0.3926    | 0.2077    | 7.114     | 6.332     | 107.0     |
| 6         | 0.9969    | 0.5364    | 0.0935    | 0.04275   | 10.67     | 1.247     | 100.3     |
| 7         | 0.9952    | 0.6666    | 0.128     | 0.6879    | 7.745     | 6.884     | 107.5     |
| 8         | 0.995     | 0.2472    | 0.4962    | 0.2218    | 8.608     | 5.461     | 143.2     |
| 9         | 0.9962    | 0.6621    | 0.3516    | 0.3288    | 6.759     | 6.353     | 107.1     |
| 10        | 0.997     | 0.2873    | 0.08624   | 0.3642    | 7.693     | 5.786     | 106.8     |
| 11        | 0.9745    | 0.103     | 0.03653   | 0.4567    | 7.324     | 6.754     | 106.0     |
| 12        | 0.9959    | 0.5374    | 0.3529    | 0.2619    | 7.456     | 5.955     | 107.5     |
| 13        | 0.9946    | 0.6162    | 0.3958    | 0.5806    | 6.124     | 6.151     | 111.2     |
| 14        | 0.9965    | 0.7947    | 0.3125    | 0.4496    | 7.172     | 5.074     | 107.1     |
| 15        | 0.9953    | 0.7129    | 0.1673    | 0.3426    | 10.53     | 5.929     | 133.6     |
| 16        | 0.9942    | 0.6199    | 0.02042   | 0.431     | 6.07      | 6.275     | 111.0     |
| 17        | 0.9976    | 0.3015    | 0.4909    | 0.8477    | 8.301     | 5.252     | 107.3     |
| 18        | 0.9971    | 0.3431    | 0.3388    | 0.4924    | 3.733     | 1.137     | 133.1     |
| 19        | 0.9945    | 0.5547    | 0.4351    | 0.5637    | 5.646     | 9.555     | 183.6     |
| 20        | 0.9951    | 0.5849    | 0.1949    | 0.4486    | 8.6       | 5.579     | 107.1     |
| 21        | 0.9968    | 0.2581    | 0.05278   | 0.1663    | 7.284     | 8.134     | 212.9     |
| 22        | 0.9927    | 0.2255    | 0.2214    | 0.9815    | 7.748     | 5.635     | 107.5     |
| 23        | 0.9939    | 0.5687    | 0.4817    | 0.3712    | 8.482     | 5.269     | 143.2     |
| 24        | 0.9942    | 0.7401    | 0.06637   | 0.5044    | 9.2       | 5.531     | 112.3     |
| 25        | 0.9953    | 0.491     | 0.01995   | 0.2867    | 7.578     | 8.278     | 213.3     |
| 26        | 0.9959    | 0.68      | 0.3512    | 0.1256    | 6.361     | 1.615     | 105.6     |
| 27        | 0.9956    | 0.7263    | 0.247     | 0.6123    | 8.311     | 4.392     | 107.2     |
| 28        | 0.9984    | 0.3479    | 0.3215    | 0.4648    | 4.424     | 1.62      | 132.6     |
| 29        | 0.9913    | 0.1458    | 0.1591    | 0.4387    | 10.99     | 1.579     | 204.4     |
| 30        | 0.9965    | 0.4738    | 0.2365    | 0.03083   | 7.628     | 6.666     | 194.8     |
=================================================================================================
Best params :
{'colsample_bytree': 0.3478684771929794, 'gamma': 0.32147166377726744, 'learning_rate': 0.46476734287585686, 'max_depth': 4, 'min_child_weight': 2, 'n_estimators': 133}
```

Figure 5.7 Bayesian Optimization Result

After we get the optimum parameters, we define the second xgboost model and train it with new parameters we get from the optimization.

```
142.def xgb_model2(X_train, Y_train, X_test, Y_test, parameters):
143. clf = xgb.XGBClassifier(seed=seed, max_depth=6, gamma=0,
   n_estimators=100, learning_rate=0.3, colsample_bytree=1).fit(X_train,
   Y_train)
144.   clf = xgb.XGBClassifier(**parameters,seed=seed).fit(X_train,
   Y_train)
145.   xgb_pred2 = clf.predict(X_test)
146.  xgb_accuracy, xgb_recall, xgb_precision, xgb_f1 = result(Y_test,
   xgb_pred2)
147.    return xgb_accuracy, xgb_recall, xgb_precision, xgb_f1
```

Line 143 is to define the model with the default parameters first and train it with the training set. Line 144 is to set the parameters with the new parameters that we get from hyperparameters tuning, and train it with the training set. Line 145 to 146 is to calculate the accuracy, recall. precision, and f1 score. Line 147 is to return the results.

```
148.ann_accuracy1_1, ann_recall1_1, ann_precision1_1, ann_f11_1 =
   ann_model1(X_train, Y_train, X_test, Y_test)
149.ann_accuracy1_2, ann_recall1_2, ann_precision1_2, ann_f11_2 =
   ann_model1(X_train2, Y_train2, X_test2, Y_test2)
150.ann_accuracy1_3, ann_recall1_3, ann_precision1_3, ann_f11_3 =
   ann_model1(X_train3, Y_train3, X_test3, Y_test3)
151.ann_accuracy1_4, ann_recall1_4, ann_precision1_4, ann_f11_4 =
   ann_model1(X_train4, Y_train4, X_test4, Y_test4)
152.ann_accuracy1_5, ann_recall1_5, ann_precision1_5, ann_f11_5 =
   ann_model1(X_train5, Y_train5, X_test5, Y_test5)
153.ann_accuracy2_1, ann_recall2_1, ann_precision2_1, ann_f12_1 =
   ann_model2(X_train, Y_train, X_test, Y_test)
154.ann_accuracy2_2, ann_recall2_2, ann_precision2_2, ann_f12_2 =
   ann_model2(X_train2, Y_train2, X_test2, Y_test2)
155.ann_accuracy2_3, ann_recall2_3, ann_precision2_3, ann_f12_3 =
   ann_model2(X_train3, Y_train3, X_test3, Y_test3)
156.ann_accuracy2_4, ann_recall2_4, ann_precision2_4, ann_f12_4 =
   ann_model2(X_train4, Y_train4, X_test4, Y_test4)
157.ann_accuracy2_5, ann_recall2_5, ann_precision2_5, ann_f12_5 =
   ann_model2(X_train5, Y_train5, X_test5, Y_test5)
158.ann_accuracy3_1, ann_recall3_1, ann_precision3_1, ann_f13_1 =
   ann_model3(X_train, Y_train, X_test, Y_test)
159.ann_accuracy3_2, ann_recall3_2, ann_precision3_2, ann_f13_2 =
   ann_model3(X_train2, Y_train2, X_test2, Y_test2)
160.ann_accuracy3_3, ann_recall3_3, ann_precision3_3, ann_f13_3 =
   ann_model3(X_train3, Y_train3, X_test3, Y_test3)
161.ann_accuracy3_4, ann_recall3_4, ann_precision3_4, ann_f13_4 =
   ann_model3(X_train4, Y_train4, X_test4, Y_test4)
```

```
162.ann_accuracy3_5,   ann_recall3_5,   ann_precision3_5,   ann_f13_5   =
   ann_model3(X_train5, Y_train5, X_test5, Y_test5)
163.ann_accuracy4_1,   ann_recall4_1,   ann_precision4_1,   ann_f14_1   =
   ann_model4(X_train, Y_train, X_test, Y_test)
164.ann_accuracy4_2,   ann_recall4_2,   ann_precision4_2,   ann_f14_2   =
   ann_model4(X_train2, Y_train2, X_test2, Y_test2)
165.ann_accuracy4_3,   ann_recall4_3,   ann_precision4_3,   ann_f14_3   =
   ann_model4(X_train3, Y_train3, X_test3, Y_test3)
166.ann_accuracy4_4,   ann_recall4_4,   ann_precision4_4,   ann_f14_4   =
   ann_model4(X_train4, Y_train4, X_test4, Y_test4)
167.ann_accuracy4_5,   ann_recall4_5,   ann_precision4_5,   ann_f14_5   =
   ann_model4(X_train5, Y_train5, X_test5, Y_test5)
168.xgb_accuracy1_1,   xgb_recall1_1,   xgb_precision1_1,   xgb_f11_1   =
   xgb_model1(X_train, Y_train, X_test, Y_test)
169.xgb_accuracy1_2,   xgb_recall1_2,   xgb_precision1_2,   xgb_f11_2   =
   xgb_model1(X_train2, Y_train2, X_test2, Y_test2)
170.xgb_accuracy1_3,   xgb_recall1_3,   xgb_precision1_3,   xgb_f11_3   =
   xgb_model1(X_train3, Y_train3, X_test3, Y_test3)
171.xgb_accuracy1_4,   xgb_recall1_4,   xgb_precision1_4,   xgb_f11_4   =
   xgb_model1(X_train4, Y_train4, X_test4, Y_test4)
172.xgb_accuracy1_5,   xgb_recall1_5,   xgb_precision1_5,   xgb_f11_5   =
   xgb_model1(X_train5, Y_train5, X_test5, Y_test5)
173.xgb_accuracy2_1,   xgb_recall2_1,   xgb_precision2_1,   xgb_f12_1   =
   xgb_model2(X_train, Y_train, X_test, Y_test, parameters)
174.xgb_accuracy2_2,   xgb_recall2_2,   xgb_precision2_2,   xgb_f12_2   =
   xgb_model2(X_train2, Y_train2, X_test2, Y_test2, parameters)
175.xgb_accuracy2_3,   xgb_recall2_3,   xgb_precision2_3,   xgb_f12_3   =
   xgb_model2(X_train3, Y_train3, X_test3, Y_test3, parameters)
176.xgb_accuracy2_4,   xgb_recall2_4,   xgb_precision2_4,   xgb_f12_4   =
   xgb_model2(X_train4, Y_train4, X_test4, Y_test4, parameters)
177.xgb_accuracy2_5,   xgb_recall2_5,   xgb_precision2_5,   xgb_f12_5   =
   xgb_model2(X_train5, Y_train5, X_test5, Y_test5, parameters)
```

Line 148 to 167 is to call all ann models functions to run the training and calculate the result with the training set and test set that we have already defined from line 123 to 127. Line 168 to 172 is to train the first xgboost model and calculate the result, line 173 to 177 is to train the second xgboost model with the new parameters we got from hyperparameters tuning and also calculate the results.

5.2. Results

In this subchapter, the results of the models will be compared. ANN 1 indicates the first ann model that has 3 hidden layers with 200, 200, and 150 neurons respectively, ANN 2 indicates the second ann model which has 4 hidden layers with 200, 200, 200, and 150 neurons respectively. ANN 3 indicates the third ann model which has 5 hidden layers with 200, 200, 150, 150, and 150 neurons respectively. ANN 4 indicates the fourth ann model which has 6 hidden layers with 200, 200, 200, 150, 150, and 150 neurons respectively. XGBoost 1 is the first xgboost model without hyperparameters tuning or with the default parameters [17], and XGboost 2 is the second model with hyperparameters tuning.

Table 5.1 Results Comparison with Train set 80% and Test set 20%

Model	Recall	Precision	F1 Score	Accuracy
ANN 1	0.958	0.986	0.971	0.962
ANN 2	0.958	0.986	0.971	0.962
ANN 3	0.986	0.986	0.986	0.981
ANN 4	0.972	0.986	0.979	0.971
XGBoost 1	0.958	1.0	0.978	0.971
XGBoost 2	0.958	1.0	0.978	0.971

Table 5.1 shows that with ratio 80:20, ANN 3 has the best overall performance of all models, followed by ANN 4, XGBoost 1 and XGBoost 2. ANN 4 and the XGBoost models have the same accuracy score but ANN 4 have better recall and F1 score, even though the precision is better on the XGBoost models.

Table 5.2 Results Comparison with Train set 75% and Test set 25%

Model	Recall	Precision	F1 Score	Accuracy
ANN 1	0.952	0.988	0.97	0.962
ANN 2	0.952	1.0	0.976	0.969
ANN 3	0.964	0.988	0.976	0.969
ANN 4	0.976	0.863	0.916	0.885
XGBoost 1	0.952	1.0	0.976	0.969
XGBoost 2	0.964	1.0	0.982	0.977

Table 5.2 shows that XGBoost 2 has the best performances over all models, followed by ANN 2 , ANN 3 and XGBoost 1. ANN 2 and XGBoost 1 have lower Recall than the ANN 3 but they have the same F1 score and have better precision.

Table 5.3 Results Comparison with Train set 70% and Test set 30%

Model	Recall	Precision	F1 Score	Accuracy
ANN 1	0.961	0.99	0.975	0.968
ANN 2	0.912	0.979	0.944	0.929
ANN 3	0.98	1.0	0.99	0.987
ANN 4	0.98	0.99	0.985	0.981
XGBoost 1	0.98	1.0	0.99	0.987
XGBoost 2	0.98	1.0	0.99	0.987

Table 5.3 shows that with 70% of the training set and 30% of the test set, both XGBoost models and ANN 3 achieve the same result and have the best performances, followed by ANN 4, ANN 1 and the last is ANN 2.

Table 5.4. Results Comparison with Train set 60% and Test set 40%

Model	Recall	Precision	F1 Score	Accuracy
ANN 1	0.939	0.992	0.965	0.957
ANN 2	0.939	0.992	0.965	0.957
ANN 3	0.947	0.984	0.965	0.957
ANN 4	0.939	0.939	0.939	0.923
XGBoost 1	0.932	1.0	0.965	0.957
XGBoost 2	0.924	1.0	0.961	0.952

Table 5.4 shows that with 60% of the training set and 40% of the test set, ANN 1, ANN 2, ANN 3, and XGBoost 1 able to achieve the same accuracy score. ANN 1 has the same results as the ANN 2, but has lower recall , precision than the ANN 3. XGBoost 1 has a lower recall than the three ann models mentioned.

Table 5.5 Results Comparison with Train set 50% and Test set 50%

Model	Recall	Precision	F1 Score	Accuracy
ANN 1	0.91	1.0	0.953	0.942
ANN 2	0.904	1.0	0.949	0.938
ANN 3	0.934	0.994	0.963	0.954
ANN 4	0.795	0.971	0.874	0.854
XGBoost 1	0.94	1.0	0.969	0.962
XGBoost 2	0.946	1.0	0.972	0.965

Table 5.5 shows that with 50% of the training set and with 50% of the test set, XGBoost 2 has the best overall performance, followed by XGBoost 1, ANN 3, ANN1, and ANN 4.

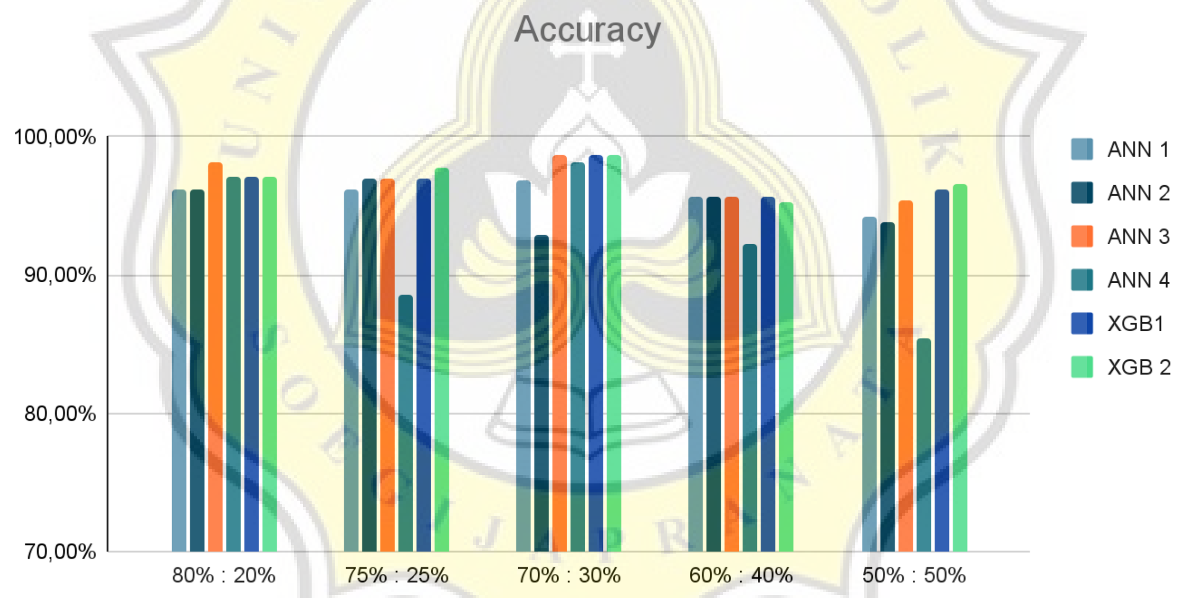


Figure 5.8 Accuracy Comparison

And as we can see from Figure 5.1 that most of the results from different ratios of training data XGBoost models are able to outperform the ANN models, although ANN 3 is able to achieve better accuracy scores at 80% of training data and achieve the same accuracy score at 70% of training data as the XGBoost models. As we can see from the figure above with 80%:20% ANN 4 also has a high accuracy score and it's the same as the XGBoost models, as ANN 1 and ANN 2 have lower scores than the other models. But with a 75%:25% ratio ANN 4 score decreased below 90%, and XGBoost 2 have the highest score, followed by ANN 2, ANN 3, and XGBoost where 3 of them have the same score. Although the three models have the same accuracy score , as we can see from Table XGBoost 1 and ANN 2 has the same result but has lower recall than ANN 3. With 70%:30 ratio XGBoost models and ANN 3 able to achieve the same accuracy scores. With 60%:40%, ANN 1, ANN 2, ANN 3, and XGBoost 1 able to achieve the same accuracy scores but Table 5.4 shows that XGBoost 1 have lower recall than ANN models, and ANN 3 has the highest recall score but ANN 3 have lower precision than ANN 1, ANN 2 and XGBoost 1. And the last with 50%:50% ratio, XGBoost 2 have the best accuracy score compared to other models, and ANN 4 have the lowest score.