Machine Learning Coursework

March 31, 2023

0.1 Individual Coursework

0.1.1 Support Vector Machine (SVM)

Importing libraries

[21]:
```
import pandas as pd
import seaborn
import matplotlib.pyplot as plt
import sklearn
import seaborn
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

0.2 Data preparation

0.2.1 Creating a dataframe

The dataset is a given csv file called diabetes.csv that contains a two-dimensional table with named columns that represent different types of information about each individual and rows that represent each person. The read csv method reads the dataset into a dataframe.

[22]:
```
diabetes_data = pd.read_csv('../Coursework/Datasets/diabetes.csv')
diabetes_data.head(5)
```

[22]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
```

```
3                        0.167   21        0
4                        2.288   33        1
```

0.2.2 Analysing data

The dataset contains a lot of "0" values in each of the columns, so it was planned to remove all rows with "0" values except for those in the "Outcome" columns. However, when the rows are removed, it shows that the dataset has shrunk significantly because the majority of them have been eliminated; consequently, no further removal is done.

[23]:
```python
# diabetes_data = diabetes_data.rename(columns={'BMI' : 'BodyMassIndex'})
# diabetes_data = diabetes_data[(diabetes_data.Insulin != 0) & (diabetes_data.
 ↪Glucose != 0) & (diabetes_data.Age != 0) & (diabetes_data.
 ↪DiabetesPedigreeFunction != 0) & (diabetes_data.BloodPressure != 0) &␣
 ↪(diabetes_data.BodyMassIndex != 0) & (diabetes_data.SkinThickness != 0)]
# diabetes_data = diabetes_data[(diabetes_data.Age != 0) & (diabetes_data.
 ↪BodyMassIndex != 0)]
diabetes_data.head(5)
```

[23]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

This line of code is tend to check null values in the dataset, but it returns that there are no null value in the dataset because the data type of the features is integer and it assumed that "0" is a value, but the fact is there are a lot of "0"s in the dataset.

[24]:
```python
diabetes_data.isnull().sum()
```

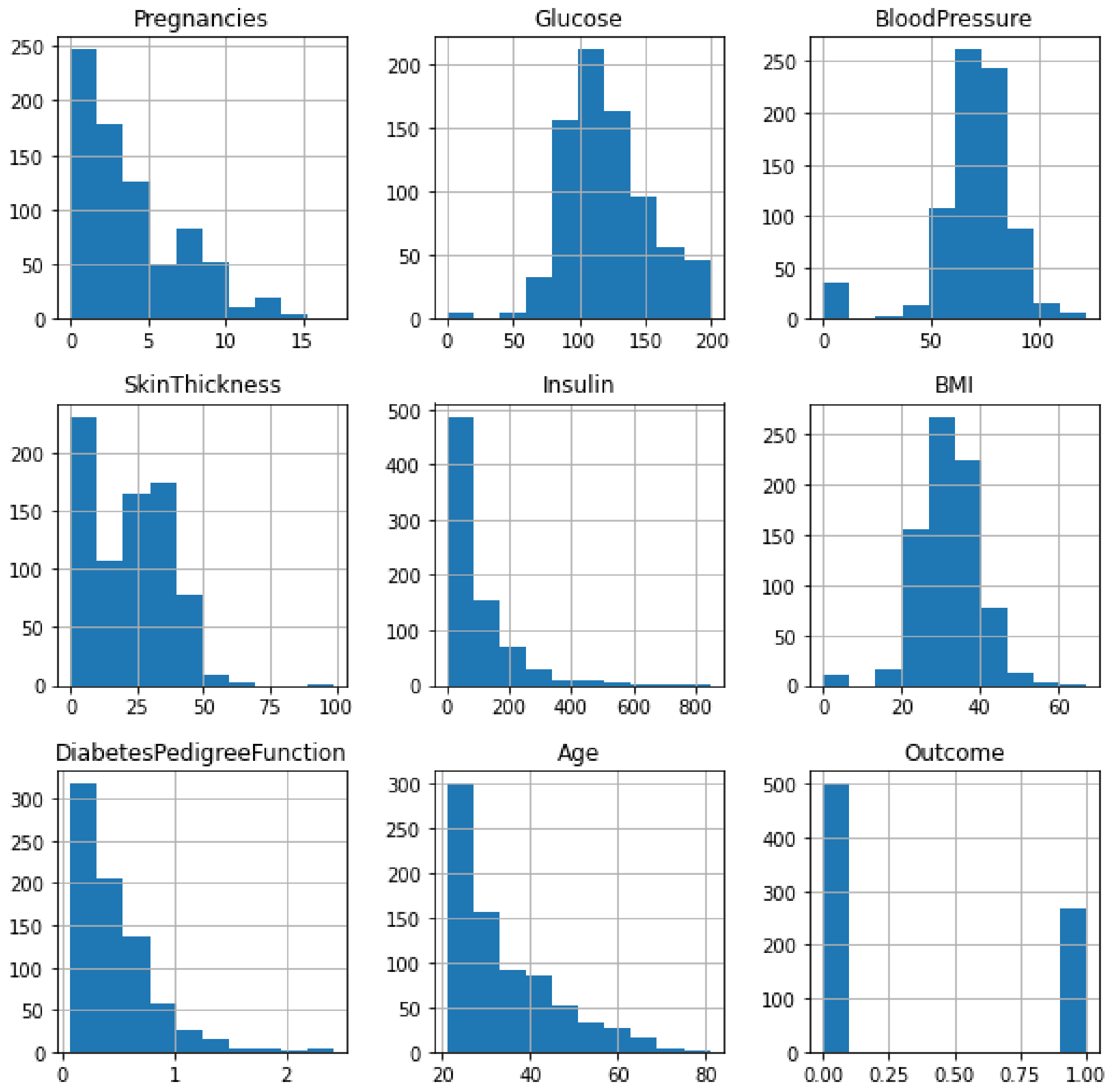
[24]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
```

```
dtype: int64
```

[25]:
```
diabetes_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[26]:
```
diabetes_data.hist(figsize=(10,10))
plt.show()
```



0.2.3 Generate Correlation Graph

Generating correlation graph using the given dataset to check the correlation between each features.

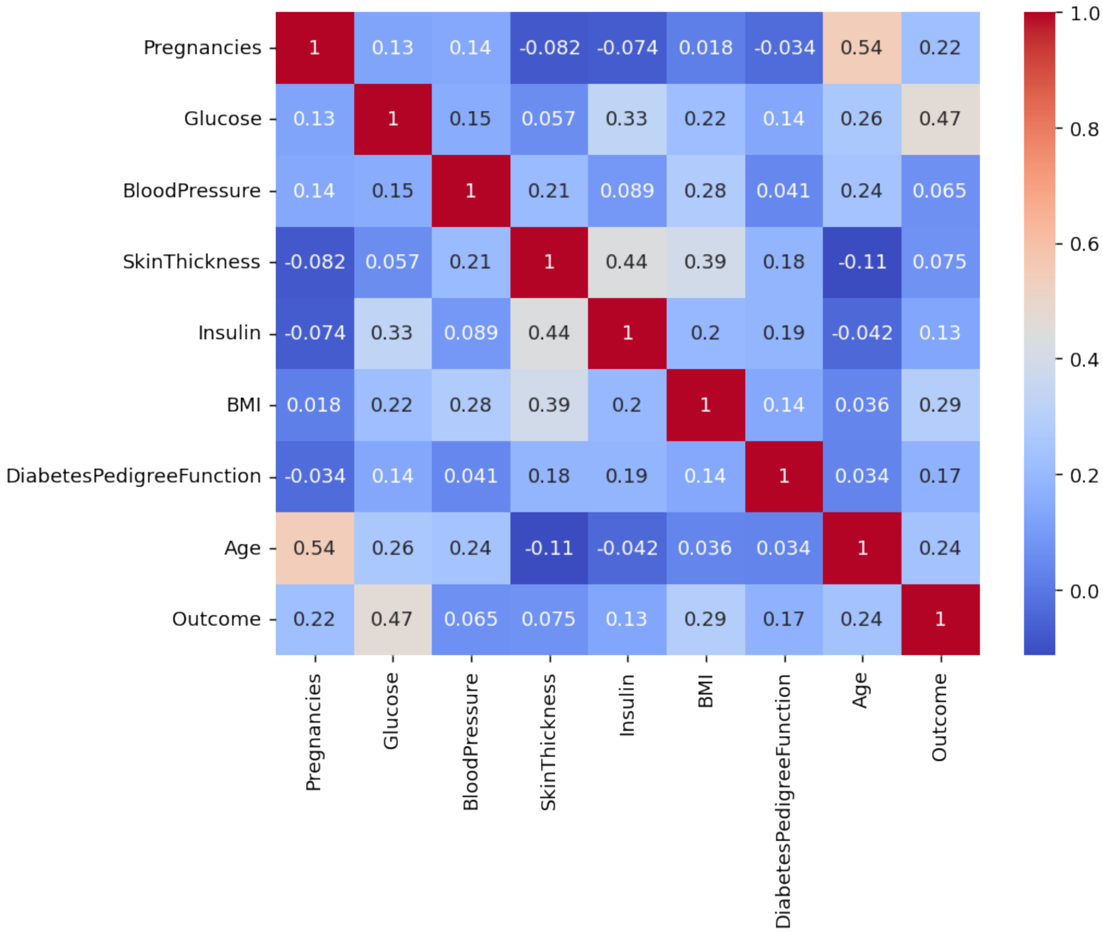
[27]:
```
corr_matrix = diabetes_data.corr()
print(corr_matrix['Outcome'])
```

```
Pregnancies                 0.221898
Glucose                     0.466581
BloodPressure               0.065068
SkinThickness               0.074752
Insulin                     0.130548
BMI                         0.292695
DiabetesPedigreeFunction    0.173844
```

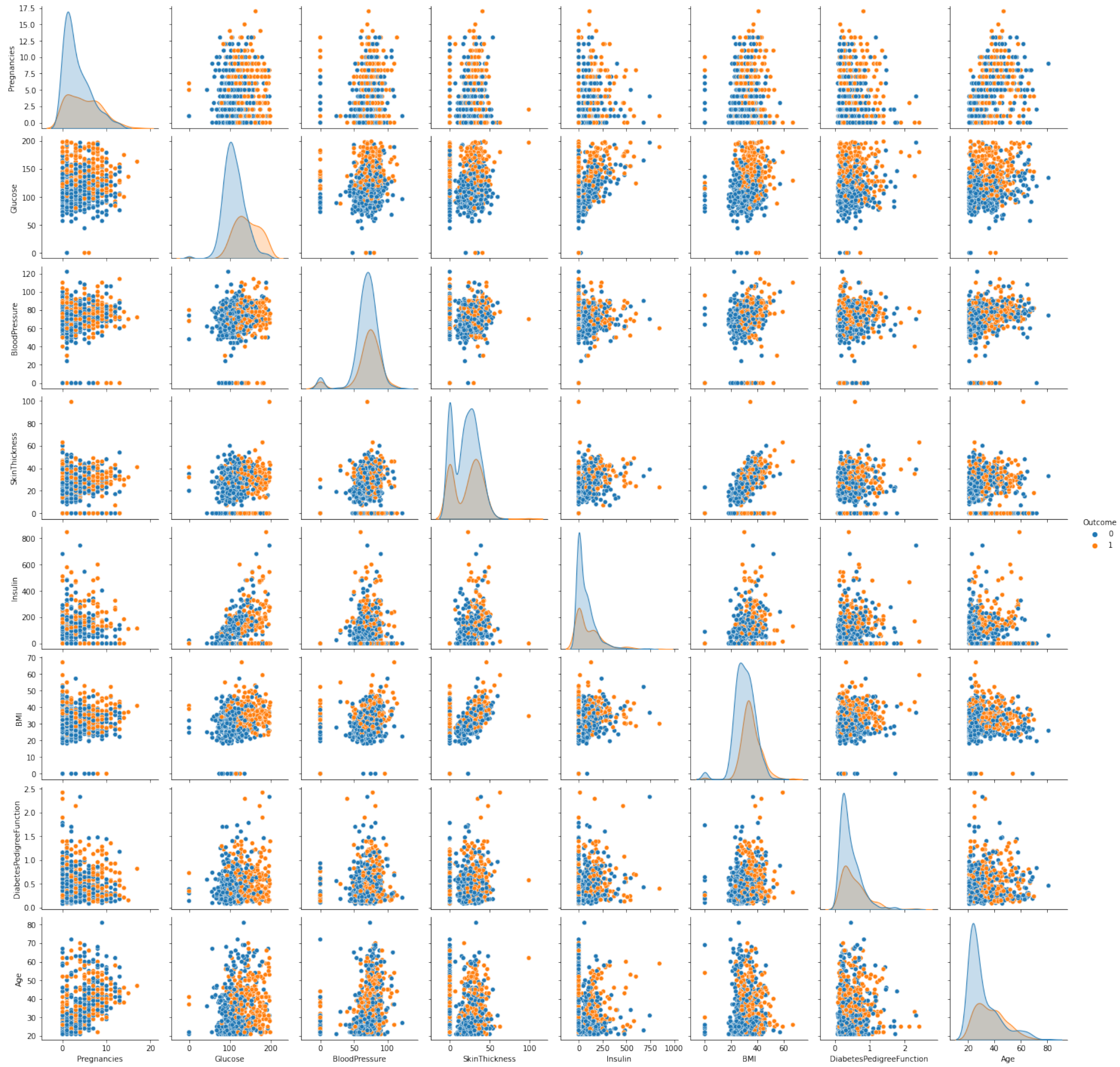
```
Age                         0.238356
Outcome                     1.000000
Name: Outcome, dtype: float64
```

[28]:
```python
plt.figure(figsize=[8,6], dpi=130)
seaborn.heatmap(diabetes_data.corr(), annot=True, cmap='coolwarm')
```

[28]: <AxesSubplot: >



[29]:
```python
seaborn.pairplot(diabetes_data, hue="Outcome")
plt.show()
```



The result shows that blood pressure and skin thickness have a very low correlation with the outcome.

0.3 Training and testing the model

0.3.1 Defining variables X and y

For variable X, skin thickness and blood pressure is removed. It is because from the correlation graph, it shows that the relationship between skin thickness and outcome or blood pressure and outcome is nearly “0” which is low relationship.

[30]:
```
X = diabetes_data.drop(['Outcome','SkinThickness','BloodPressure'], axis=1)
# X = diabetes_data[['BMI','Glucose']]
y = diabetes_data['Outcome']
```

```
X = X.reset_index(drop=True)

X[:1]
```

[30]:

	Pregnancies	Glucose	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	0	33.6	0.627	50

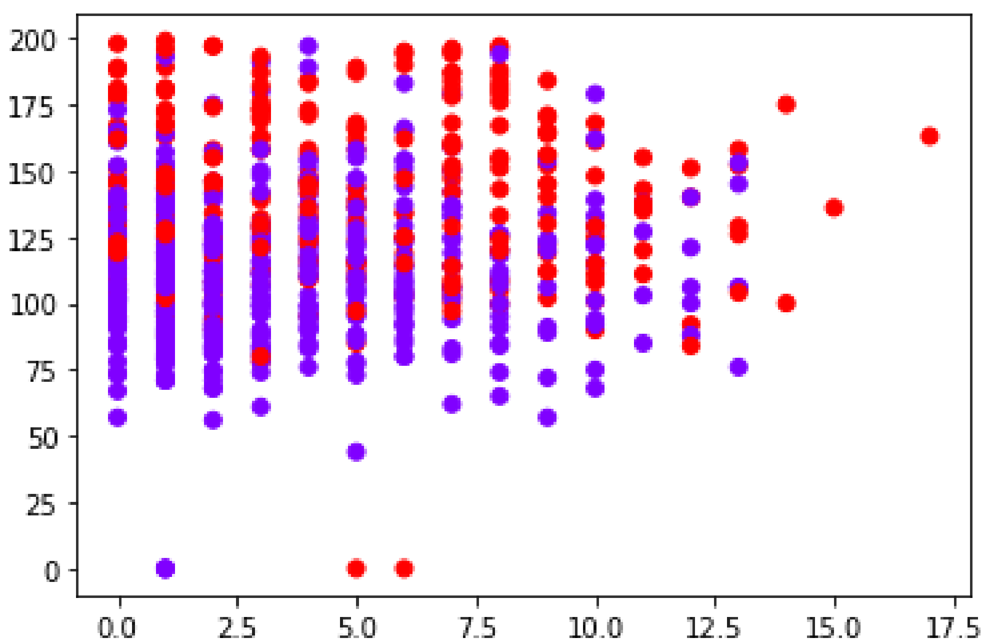
[31]:
```
y[0:1]
```

[31]:
```
0    1
Name: Outcome, dtype: int64
```

0.3.2 Generate scatter graph

Generating a scatter graph to check is the classes are well separated and form distinct clusters.

[32]:
```
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y, cmap='rainbow')
plt.show()
```



0.3.3 Spliting X and y into training and testing dataset

Splitting the dataset into 8:2 by using train_test_split, the training data will be 80% of the dataset and the test data will be 20% of the dataset as the test_size is set to 0.2.

[33]:
```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,␣
↪random_state=42)
```

0.3.4 Data Preprocessing

In this model, two scaling features, StandardScaler and normalization, are tried to bring all the features to a similar scale so that one feature does not dominate the others during the training. But StandardScaler is chosen in the end. This is because, unlike normalisation, which has a predefined range of transformation features that force the data between 0 and 1, StandardScaler is more resistant to outliers.

[34]:
```
scaler = sklearn.preprocessing.StandardScaler()
scaler.fit(X_train)
# scaler = sklearn.preprocessing.MinMaxScaler()
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

print(X_train.shape)
print(X_test.shape)
```

```
(614, 6)
(154, 6)
```

0.3.5 Building SVM models

Three different kernels models, linear, polynomial, and radius basis function, are built to used to compare each other which would generate the highest accuracy.

[35]:
```
clf_linear_test = SVC(kernel='linear')
clf_poly_test = SVC(kernel='poly')
clf_rbf_test = SVC(kernel='rbf')
```

0.3.6 Cross Validation

Cross validation is used to estimate the performance of each maching learning model. The cross_val_score function perform k-fold cross-validation on the dataset and returns the average score across all folds. By using this function, the decision of choosing which kernel should be using to perform diabetes prediction could be done by comparing the performance of different SVM kernel and choose the one that gives the best average score.

[36]:
```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(clf_linear_test, X_train, y_train, scoring =␣
↪'accuracy', cv = 10).mean()
print('Linear Accuracy: ', scores)

scores = cross_val_score(clf_poly_test, X_train, y_train, scoring = 'accuracy',␣
↪cv = 10).mean()
print('Poly Accuracy: ', scores)
```

```
scores = cross_val_score(clf_rbf_test, X_train, y_train, scoring = 'accuracy',␣
↪cv = 10).mean()
print('RBF Accuracy: ', scores)
```

```
Linear Accuracy:  0.7605764145954522
Poly Accuracy:  0.7198572184029615
RBF Accuracy:  0.7653358011634055
```

According to the cross-validation results, the linear and RBF kernels have the highest levels of accuracy. However, RBF is the chosen method not only because it is more accurate than linear, but also because the scatter graph demonstrates that the classes overlap and some of them are dispersed across the feature space, indicating a likelihood that they are non-linearly separable.

0.4 SVM model with Radial Basis Function (RBF) kernel

0.4.1 Testing hyperparameters

[37]:
```
rbf_clf = SVC(kernel='rbf', C=0.1, gamma=0.1)
rbf_clf.fit(X_train, y_train)
y_rbf_pred = rbf_clf.predict(X_test)
print(classification_report(y_test, y_rbf_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.89      0.84        99
           1       0.74      0.58      0.65        55

    accuracy                           0.78       154
   macro avg       0.77      0.74      0.75       154
weighted avg       0.78      0.78      0.77       154
```

[38]:
```
rbf_clf = SVC(kernel='rbf', C=100, gamma=10)
rbf_clf.fit(X_train, y_train)
y_rbf_pred = rbf_clf.predict(X_test)
print(classification_report(y_test, y_rbf_pred))
```

```
              precision    recall  f1-score   support

           0       0.64      0.99      0.78        99
           1       0.00      0.00      0.00        55

    accuracy                           0.64       154
   macro avg       0.32      0.49      0.39       154
weighted avg       0.41      0.64      0.50       154
```

The hyperparameters: { ‘C’: [0.1, 1, 10, 100], ‘gamma’: [0.01, 0.1, 1, 10] } is tried in the RBF kernel model. The outcome demonstrates that the model performs best on the validation set when

C = 1 and gamma = 0.01. The results shows that when C = 1, the model has the best performance on the validation set, which indicates that the model is well-regularized and does not suffer from underfitting or overfitting. Also, when C = 1, it means that the model has a good balance between maximizing the margin and minimizing the classificatiaon error on the training set. For the gamma hyperparameter, it affects the flexibility of the border, which determines the shape of the decision boundary. It offers the best balance between bias and variance when set to 0.01.

[39]:
```
rbf_clf = SVC(kernel='rbf', C=1, gamma=0.01)
rbf_clf.fit(X_train, y_train)
```

[39]: SVC(C=1, gamma=0.01)

[40]:
```
y_rbf_pred = rbf_clf.predict(X_test)
```

0.5 Performance evaluation

0.5.1 Generate a confusion matrix

The confusion matrix is calculated by comparing the predicted values of a model to the actual values in a test dataset. In this predicting diabetes which is a binary classification problem, the confusion matrix would have four elements which are True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN). TP means that the model correctly predicted the positive class. FP means that the model predicted the positive class but it was actually negative. TN means that the model correctly predicted the negative class and FN means that the model predicted the negative class, but it was actually positive.

[41]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_rbf_pred)
print(cm)
```

```
[[87 12]
 [21 34]]
```

According to the confusion matrix generated by the model, the negative class was correctly predicted 87 times (TN), the positive class was incorrectly predicted 12 times (FP), the negative class was incorrectly predicted 21 times (FN), and the positive class was correctly predicted 34 times (TP).

0.5.2 Calculate precision, recall, f1-score and support

By using the above confusion matrix, calculate the precision, recall, f1-score and support for the model.

[42]:
```
tn, fp, fn, tp = cm[0][0], cm[0][1], cm[1][0], cm[1][1]

precision = tp / (tp + fp)
recall = tp / (tp + fn)
f1_score = 2 * (precision * recall) / (precision + recall)
support_negative = tn + fp
support_positive = tp + fn
```

```python
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
print("Support (negative):", support_negative)
print("Support (positive):", support_positive)
```

```
Precision: 0.7391304347826086
Recall: 0.6181818181818182
F1-score: 0.6732673267326733
Support (negative): 99
Support (positive): 55
```

0.5.3 Generate a classification report

The calculation above should be verified by creating a classification report, which also provides a summary of the model's performance. The percentage of accurate positive predictions among all positive predictions made by the model is known as the precision score. Recall displays the percentage of accurate positive predictions among all instances of actual positive data. It gauges how effectively the model can spot positive cases. The F1-score provides an overall assessment of the model's accuracy by displaying a harmonic mean of precision and recall. The support displays the dataset's number of observations for each class.

[43]:
```python
print(classification_report(y_test, y_rbf_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84        99
           1       0.74      0.62      0.67        55

    accuracy                           0.79       154
   macro avg       0.77      0.75      0.76       154
weighted avg       0.78      0.79      0.78       154
```

According to the classification report's precision, 73% of the model's optimistic predictions were accurate. Recall score of 0.62 indicates that 62% of the actual positive cases were correctly identified by the model. The 0.67 F1-score indicates a moderate level of accuracy for the model. This model's support section contains 99 negative cases and 55 positive cases.

0.6 Ensemble

0.6.1 Creating adaptive boosting classifier (AdaBoost)

The classification models' accuracy is increased with AdaBoost. To see which model could achieve the highest accuracy, several models were created. The number of weak learners is determined by the number of estimators. The model performs best overall on the validation set when the number of estimators is set to 50. Since each new weak learner has the same weight as the previous ones when the learning rate is set to 1, overfitting can be avoided.

0.6.2 Testing hyperparameters

[46]:
```
tree_clf = DecisionTreeClassifier(max_depth=3)
tree_clf.fit(X_train, y_train)
tree_pred = tree_clf.predict(X_test)
print(classification_report(y_test, tree_pred ))
```

```
              precision    recall  f1-score   support

           0       0.80      0.84      0.82        99
           1       0.68      0.62      0.65        55

    accuracy                           0.76       154
   macro avg       0.74      0.73      0.73       154
weighted avg       0.76      0.76      0.76       154
```

[47]:
```
tree_clf = DecisionTreeClassifier(max_depth=5)
tree_clf.fit(X_train, y_train)
tree_pred = tree_clf.predict(X_test)
print(classification_report(y_test, tree_pred ))
```

```
              precision    recall  f1-score   support

           0       0.82      0.85      0.84        99
           1       0.71      0.67      0.69        55

    accuracy                           0.79       154
   macro avg       0.77      0.76      0.76       154
weighted avg       0.78      0.79      0.78       154
```

[48]:
```
tree_clf = DecisionTreeClassifier(max_depth=1)
tree_clf.fit(X_train, y_train)
tree_pred = tree_clf.predict(X_test)
print(classification_report(y_test, tree_pred ))
```

```
              precision    recall  f1-score   support

           0       0.81      0.78      0.79        99
           1       0.63      0.67      0.65        55

    accuracy                           0.74       154
   macro avg       0.72      0.73      0.72       154
weighted avg       0.75      0.74      0.74       154
```

[51]:
```
tree_clf = DecisionTreeClassifier(max_depth=5)
tree_ada_clf = AdaBoostClassifier(tree_clf, n_estimators=50, learning_rate=1)
```

```
log_clf = LogisticRegression()
log_ada_clf = AdaBoostClassifier(log_clf, n_estimators=50, learning_rate=1)

rnd_clf = RandomForestClassifier()
rnd_ada_clf = AdaBoostClassifier(rnd_clf, n_estimators=50, learning_rate=1)

svm_clf = SVC(kernel='rbf', probability=True)
svm_ada_clf = AdaBoostClassifier(svm_clf, n_estimators=50, learning_rate=1)
```

0.6.3 Cross Validation

By using cross validation, the decision of choosing which model should be used to perform AdaBoost could be done by comparing each performance and choose the one that gives the best average score.

[52]:
```
scores = cross_val_score(tree_ada_clf, X_test, y_test, scoring="accuracy",␣
    ↪cv=5).mean()
print("Decision Tree Accuracy: ", scores)
scores = cross_val_score(log_ada_clf, X_test, y_test, scoring="accuracy", cv=5).
    ↪mean()
print("Logistic Regression Accuracy: ", scores)
scores = cross_val_score(rnd_ada_clf, X_test, y_test, scoring="accuracy", cv=5).
    ↪mean()
print("Random Forest Accuracy: ", scores)
scores = cross_val_score(svm_ada_clf, X_test, y_test, scoring="accuracy", cv=5).
    ↪mean()
print("SVM Accuracy: ", scores)
```

```
Decision Tree Accuracy:  0.7406451612903225
Logistic Regression Accuracy:  0.786236559139785
Random Forest Accuracy:  0.7146236559139785
SVM Accuracy:  0.6427956989247312
```

According to the cross-validation results, the logistic regression has the highest accuracy, therefore, it is chosen to perform AdaBoost

[235]:
```
log_ada_clf.fit(X_train,y_train)
ada_y_pred = log_ada_clf.predict(X_test)
```

[236]:
```
print(confusion_matrix(y_test, ada_y_pred))
```

```
[[80 19]
 [20 35]]
```

According to the confusion matrix generated by the model, the negative class was correctly predicted 80 times (TN), the positive class was incorrectly predicted 19 times (FP), the negative class was incorrectly predicted 20 times (FN), and the positive class was correctly predicted 35 times (TP).

[237]:
```
print(classification_report(y_test, ada_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.81      0.80        99
           1       0.65      0.64      0.64        55

    accuracy                           0.75       154
   macro avg       0.72      0.72      0.72       154
weighted avg       0.75      0.75      0.75       154
```

According to the classification report's precision, 65% of the model's optimistic predictions were accurate. Recall score of 0.64 indicates that 64% of the actual positive cases were correctly identified by the model. The 0.64 F1-score indicates a moderate level of accuracy for the model. This model's support section contains 99 negative cases and 55 positive cases.

0.6.4 Creating voting classifier

Voting classifier combine the predictions of multiple models to generate a single prediction. As different models might have different strengthens and weaknesses, combining them could create a more accurate prediction. Hard voting is chosen in this model because it could be simple and effective way to combine the predictions of multiple model as it takes the majority vote of the predicted classes by each individual model without considering confidence and probability estimates of each model. Moreover, diabetes prediction is a binary classification problems which does not need complex voting system.

[238]:
```
from sklearn.ensemble import VotingClassifier

log_model = LogisticRegression()
dec_tree_model = DecisionTreeClassifier(max_depth=3)

voting_model = ␣
↪VotingClassifier(estimators=[('lr',log_model),('dectree',dec_tree_model),('rbf',rbf_clf),('
↪voting = 'hard')
voting_model.fit(X_train, y_train)
```

[238]:
```
VotingClassifier(estimators=[('lr', LogisticRegression()),
                             ('dectree', DecisionTreeClassifier(max_depth=3)),
                             ('rbf', SVC(C=1, gamma=0.01)),
                             ('ada',
                              AdaBoostClassifier(estimator=LogisticRegression(),
                                                 learning_rate=1))])
```

[239]:
```
for clf in (log_model, dec_tree_model,rbf_clf, voting_model):
    clf.fit(X_train, y_train)
    vote_y_pred = clf.predict(X_test)
```

[240]:
```
print(confusion_matrix(y_test, vote_y_pred))
```

```
[[87 12]
 [19 36]]
```

According to the confusion matrix generated by the model, the negative class was correctly predicted 87 times (TN), the positive class was incorrectly predicted 12 times (FP), the negative class was incorrectly predicted 19 times (FN), and the positive class was correctly predicted 36 times (TP).

[241]:
```
print(classification_report(y_test, vote_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.88      0.85        99
           1       0.75      0.65      0.70        55

    accuracy                           0.80       154
   macro avg       0.79      0.77      0.77       154
weighted avg       0.80      0.80      0.80       154
```

According to the classification report's precision, 75% of the model's optimistic predictions were accurate. Recall score of 0.65 indicates that 65% of the actual positive cases were correctly identified by the model. The 0.70 F1-score indicates a moderate level of accuracy for the model. This model's support section contains 99 negative cases and 55 positive cases.

0.7 Conclusion

0.7.1 Comparing models' performance

Classification reports

[242]:
```
df1 = pd.DataFrame.from_dict(classification_report(y_test, y_rbf_pred,␣
 ↪output_dict=True))
df2 = pd.DataFrame.from_dict(classification_report(y_test, vote_y_pred,␣
 ↪output_dict=True))

df1 = df1.transpose()
df2 = df2.transpose()

df1['model'] = 'RBF SVM Model'
df2['model'] = 'Voting Classifier'

df = pd.concat([df1, df2])

print(df)
```

```
              precision    recall  f1-score     support            model
0              0.805556  0.878788  0.840580   99.000000    RBF SVM Model
1              0.739130  0.618182  0.673267   55.000000    RBF SVM Model
accuracy       0.785714  0.785714  0.785714    0.785714    RBF SVM Model
macro avg      0.772343  0.748485  0.756924  154.000000    RBF SVM Model
weighted avg   0.781832  0.785714  0.780825  154.000000    RBF SVM Model
```

```
0               0.820755  0.878788  0.848780   99.000000  Voting Classifier
1               0.750000  0.654545  0.699029   55.000000  Voting Classifier
accuracy        0.798701  0.798701  0.798701    0.798701  Voting Classifier
macro avg       0.785377  0.766667  0.773905  154.000000  Voting Classifier
weighted avg    0.795485  0.798701  0.795298  154.000000  Voting Classifier
```

Confusion matrix

[243]:
```python
print("RBF SVM Model:")
print(cm)
print("\nVoting Classifier:")
print(confusion_matrix(y_test, vote_y_pred))
```

```
RBF SVM Model:
[[87 12]
 [21 34]]

Voting Classifier:
[[87 12]
 [19 36]]
```

Based on the aforementioned findings, it can be concluded that the voting classifier outperforms the RBF model in terms of accuracy, precision, recall, and f1-score. It has TP = 36, FP = 12, TN = 87, FN = 19 from the voting classifier, and TP = 34, FP = 12, TN = 87, FN = 21 from the RBF Model. This comparison demonstrates that the voting classifier has a higher true positive value, which indicates a higher level of accuracy in predicting the positive class. Additionally, the voting classifier has a higher false negative value, indicating a higher level of accuracy in predicting negative classes. Other than that, the RBF model has a 78% overall accuracy, while the voting classifier has an overall accuracy of 80%.

0.7.2 Pros and Cons on each model

The RBF model is a versatile kernel that can learn non-linear decision boundaries, making it possible to solve more challenging classification problems. When working with real-world datasets that could contain errors, the RBF kernel's ability to handle noise and outliers is crucial. The selection of the kernel parameters, however, is crucial when using RBF kernel SVM because it will have an impact on the model's performance. Hyperparameters like gammma and C are difficult to calculate at their ideal values, which can result in subpar performance.

The benefits of voting classifiers include improved accuracy, flexibility, and robustness. Voting classifiers can combine various models with various strengths and weaknesses by achieving flexibility. Voting classifiers are regarded as robust because they combine the predictions of various models, whereas a single model might be overfitting. It improves accuracy by combining the predictions of several different classifiers, which is better than using just one classifier. Voting classifiers, however, can be complicated because they need to train and optimise numerous models, which takes a lot of time. Last but not least, because voting classifier combines the predictions of various models, it is challenging to understand how each model contributed to the final prediction.

0.8 References

Editorial (2022) Pros and cons of Support Vector Machine (SVM) RoboticsBiz.10 September 2022 [online]. Available from: https://roboticsbiz.com/pros-and-cons-of-support-vector-machine-svm/.

AmarKumar (2019) Voting Classifier in Machine Learning Analytics Vidhya.29 December 2019 [online]. Available from: https://medium.com/analytics-vidhya/voting-classifier-in-machine-learning-9534504eba39 [Accessed 31 March 2023].

'ML | Voting Classifier using Sklearn' (2019) GeeksforGeeks.23 November 2019 [online]. Available from: https://www.geeksforgeeks.org/ml-voting-classifier-using-sklearn/.