Diabetes Diagnosis

I. Introduction

Despite its prevalence being on the rise, diabetes (diabetes mellitus) has become one of the one of most common chronic diseases worldwide. 537 million people around the world suffer from this disease, and it is foreseen that this number will increase to 643 million by 2030 and 783 million by 2045. Low and middle-income countries account for more than three out of four people with diabetes. One death every five seconds (about 6.7 million deaths in one year) was caused by diabetes in 2021 and in the last 15 years, diabetes has caused a 316% increase in health expenditures by 966 billion USD `(International Diabetes Federation, 2021)`.
The condition of diabetes is characterized by high blood glucose levels. The condition can occur when your body does not produce enough insulin or when the insulin it produces is ineffective. Or a condition in which your body is not able to produce any insulin at all `(The British Diabetic Association operating as Diabetes UK, 2022)`.
Having high glucose levels in your blood over time can cause serious problems with your heart, eyes, feet, and kidneys. These are known as diabetic complications But the early detection and the proper treatment and care can reduce the risk of experiencing these complications, and diabetics can live a healthy life. It is important to early make a diagnosis which will allow for earlier precise treatment `(The British Diabetic Association operating as Diabetes UK, 2022)`.
The main objective of this report is to study machine learning/deep learning models in order to predict whether the diabetes test result was positive or negative. 6 models studied in MA336, listed below, will be implemented to diagnose diabetes and the performance of each method will be evaluated and compared.

> (1) Logistic Regression
> (2) K Nearest Neighbors (KNN)
> (3) Decision Trees algorithm
> (4) Random Forests
> (5) Support vector machines (SVM)
> (6) Artificial Neural Network (ANN)

The report consists of 4 sections. First is this `Introduction` section where the background of this study is introduced, next is the `Method` section where the data set will be explored and all models will be implemented and adjusted to improve the performances. In the third section, `Result`, performances of each model will be compared and discussed. Finally, the overall result of the study will be summarised in the `Conclusion` section.
All experiments in this report are conducted by Python version 3.7.13 and the majority of code is from the MA336 module.

II. Methods

Data preparation

In this section, we will first do some exploration of the dataset , proceed the feature engineeting process, in case if the data cleaning process is required, it will also done in this section. Then the dataset will be splited into 2 portions as a training set and test set.

1) Data Exploration

The dataset used in this study is retrieved from Kaggle `(Mehmet Akturk, 2020)` , which is original from the National Institute of Diabetes and Digestive and Kidney Diseases. This 768 rows x 9 columns dataset is a minimized version of a larger database, selecting only from females aged over 21 years old of Pima Indian heritage.
The diabetes dataset consists of 9 variables, 8 independent variables or input features and 1 output variable or label. The explanation of all variables is listed below.

> (1) Pregnancies represent the number of pregnancies
> (2) Glucose is the plasma glucose concentration in an oral glucose tolerance test
> (3) BloodPressure shows the diastolic blood pressure (mm Hg)
> (4) SkinTickness means the triceps skin fold thickness (mm)
> (5) Insulin exhibits 2-Hour serum insulin (mu U/ml)
> (6) BMI (Body mass index) is a measure of body fat based on height and weight (weight in kg/(height in m)^2)
> (7) DiabetesPedigreeFunction represents diabetes pedigree function
> (8) Age is the participant's age (years)
> (9) Outcome, the binary classification output variable which represents the diabetes test result (0 represents the negative test and 1 implies the positive test result).

The data used in this study is in CSV format, which is quite convenient to operate using a pandas data frame. First, we will download the dataset and do some data exploration. Starting by confirming Univariate Descriptive Statistics of each variable using the describe() function.

In [351…

```python
# import all required libaries
import warnings
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn. neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, plot_confusion_m
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.backend import clear_session
from sklearn import model_selection
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedKFold
# prevent printing warning
warnings.filterwarnings('ignore')
diabetes = pd.read_csv("diabetes.csv")
# confirm Univariate Descriptive Statistics of each variable
print(round(diabetes.describe().T,1))
```

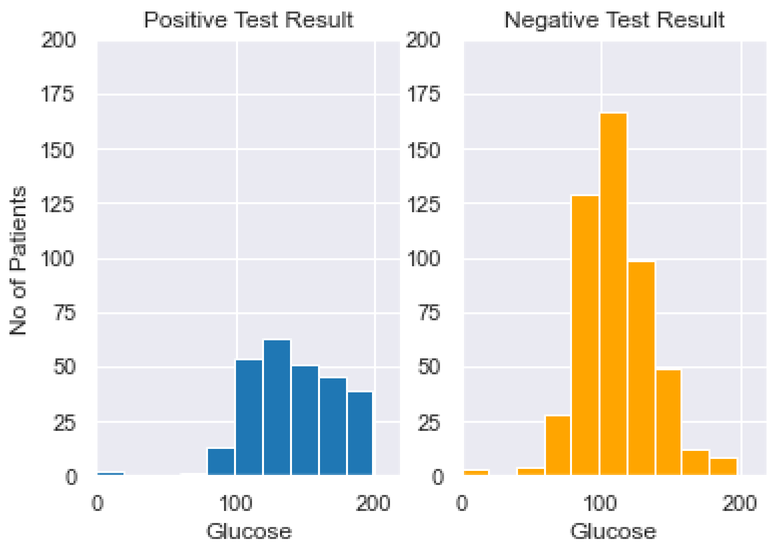
```
                          count   mean    std   min   25%    50%    75%    max
Pregnancies               768.0    3.8    3.4   0.0   1.0    3.0    6.0   17.0
Glucose                   768.0  120.9   32.0   0.0  99.0  117.0  140.2  199.0
BloodPressure             768.0   69.1   19.4   0.0  62.0   72.0   80.0  122.0
SkinThickness             768.0   20.5   16.0   0.0   0.0   23.0   32.0   99.0
Insulin                   768.0   79.8  115.2   0.0   0.0   30.5  127.2  846.0
BMI                       768.0   32.0    7.9   0.0  27.3   32.0   36.6   67.1
DiabetesPedigreeFunction  768.0    0.5    0.3   0.1   0.2    0.4    0.6    2.4
Age                       768.0   33.2   11.8  21.0  24.0   29.0   41.0   81.0
Outcome                   768.0    0.3    0.5   0.0   0.0    0.0    1.0    1.0
```

We can see the univariate descriptive statistics include central tendency (mean), minimum and maximum value, quartiles and standard deviation of each variable. From the result above, the abnormality of the data is clearly displayed. `The minimum value of Glucose, BloodPressure, SkinThickness, Insulin and BMI are 0` which is impossible and in this case, it exhibits the missing values. We will use this information to clean our dataset in the next part of the report.

Next, the bivariate analysis will be used to explore more whether the input variable influences the response variable or not. Since the characteristic of diabetes is evidently explained by high blood glucose levels, we will then investigate more on `Glucose` in association with the diabetes test result by plotting histograms of the Glucose distributions for the two labels 'Positive' and 'Negative'.

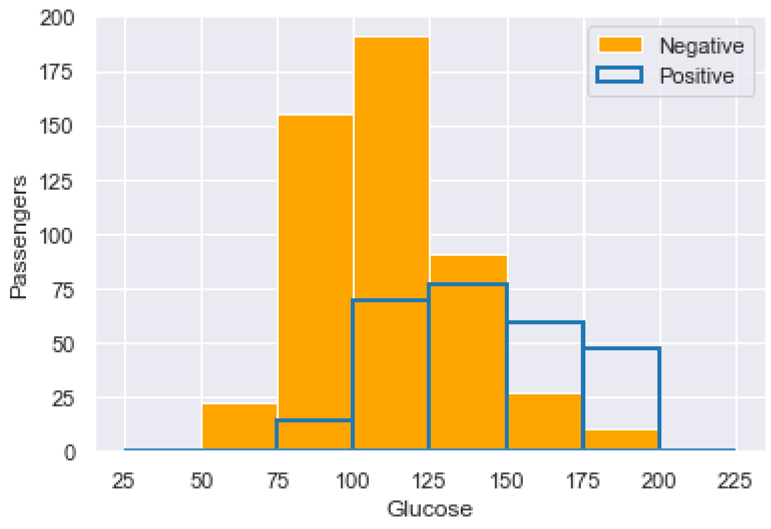
In [352…

```python
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.hist(diabetes.loc[diabetes['Outcome']==1].Glucose,color='tab:blue')
ax1.set_xlabel('Glucose')
ax1.set_ylabel('No of Patients')
ax1.set_title('Positive Test Result')
ax2.hist(diabetes.loc[diabetes['Outcome']==0].Glucose,color='orange')
ax2.set_xlabel('Glucose')
ax2.set_title('Negative Test Result')
# Set the axis limits to be the same for both both plots
xlimits = [0, max(diabetes.Glucose)+20]
ylimits = [0, 200]
ax1.set_xlim(xlimits)
ax1.set_ylim(ylimits)
ax2.set_xlim(xlimits)
ax2.set_ylim(ylimits)
plt.show()
```



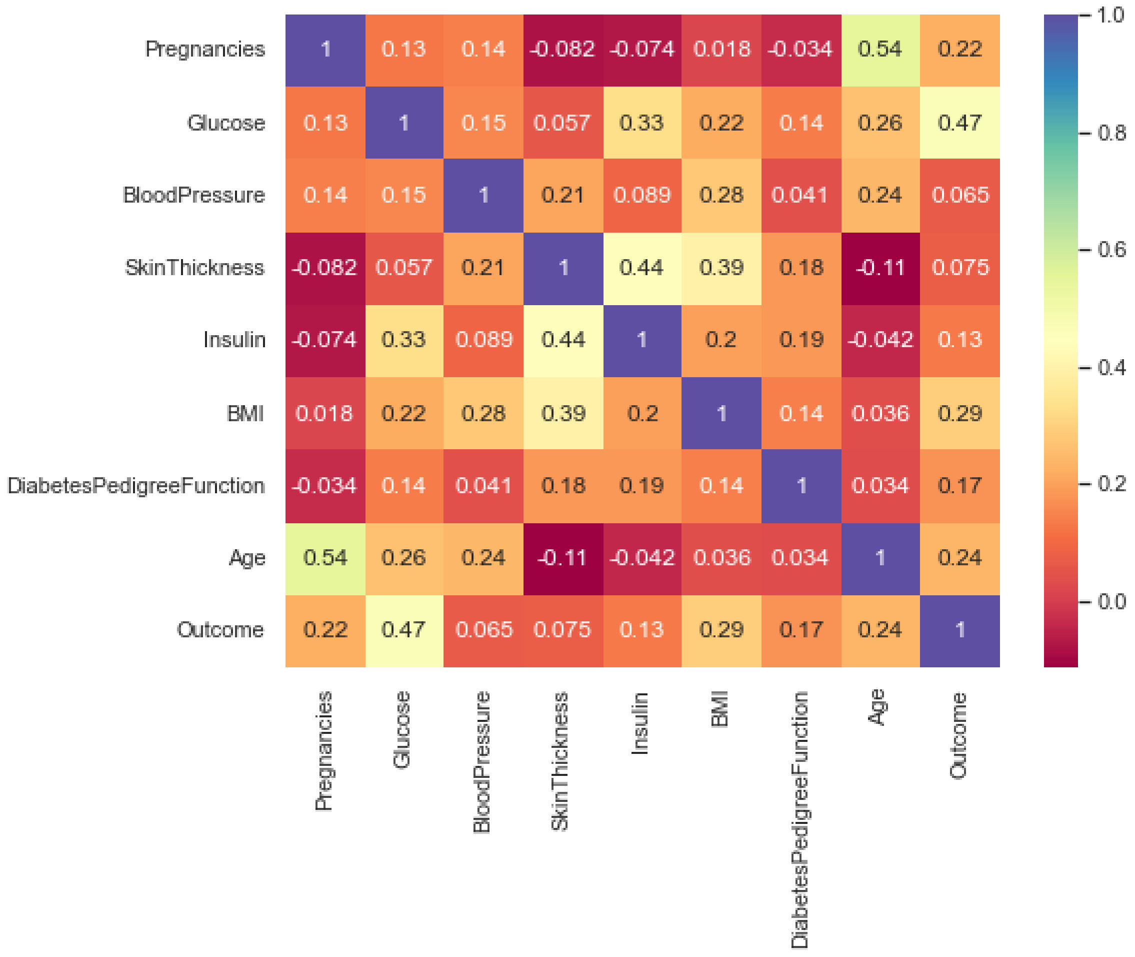
The figure above obviously demonstrates that `Glucose` affects diabetes, the person who has a test result positive, has a high value of Glucose. Most diabetics have Glucose values between 100 and 200, while the non-diabetes person has a normal distribution of Glucose values. Next, we will overlapping plot the graph of Glucose and the positive and negative test result to make the visualisation clearer.

In [353…
```
# Use the same bins for both plots
bins_glucose = [25,50,75,100,125,150,175,200,225]
fig, ax = plt.subplots(1)
ax.hist(diabetes.loc[diabetes['Outcome']==0].Glucose,bins=bins_glucose,color='orange'
ax.hist(diabetes.loc[diabetes['Outcome']==1].Glucose,bins=bins_glucose,facecolor='non
ax.set_xlabel('Glucose')
ax.set_ylabel('Passengers')
ax.legend()
plt.show()
```



The result apparently demonstrates that the ratio of non-diabetics is much higher than diabetics at the lower value of Glucose. While with the high value of Glucose, the ratio of the positive test result is extremely higher than the negative result. It clearly explains that the Glucose feature influences the response variable.
Next, we will confirm the relationship between other input features and the label. But this time, a correlation matrix will be used to confirm all at the same time.

In [354…
```
# plot correlation matrix
sns.set()
plt.figure(figsize=(8,6))
p=sns.heatmap(diabetes.corr(), annot=True,cmap ='Spectral')
```



The above correlation matrix reconfirms that Glucose is the most influential on the output Outcome, the second and third features which affect the Outcome are BMI and Age respectively. While the BloodPressure and SkinThickness correlate with the Outcome the least. As a feature engineering process, these 2 features might be excluded from the model in order to improve model performance and reduce the model complexity, which we will do further investigation in the modelling section. Apart from the relation between features and labels, we can also check the correlation between the input variables. The high correlation between the inputs should not be included in the same model. In this case, Age and Pregnancies display the highest correlation value but the value is not extremely high. We will check again in the model section whether these 2 variables will be both included in the model or not.

Next, since the label is a binary classification variable, before starting the analysis, it is required to confirm the balance of the data in each class.

In [355…

```
# Calculating the percentages of each class
print("Positive test result: ",str(round(len(diabetes.loc[diabetes['Outcome']==1])/le
print("Negative test result: ",str(round(len(diabetes.loc[diabetes['Outcome']==0])/le
```

```
Positive test result:  34.9 %
Negative test result:  65.1 %
```

From the result, we can see that the data is slightly unbalanced, the ratio of the negative test result is about two-thirds of the dataset. In this report, we will not do over/under-sampling to make it balanced. But each method in this report has to beat the negative test result percentage which is 65.1%.

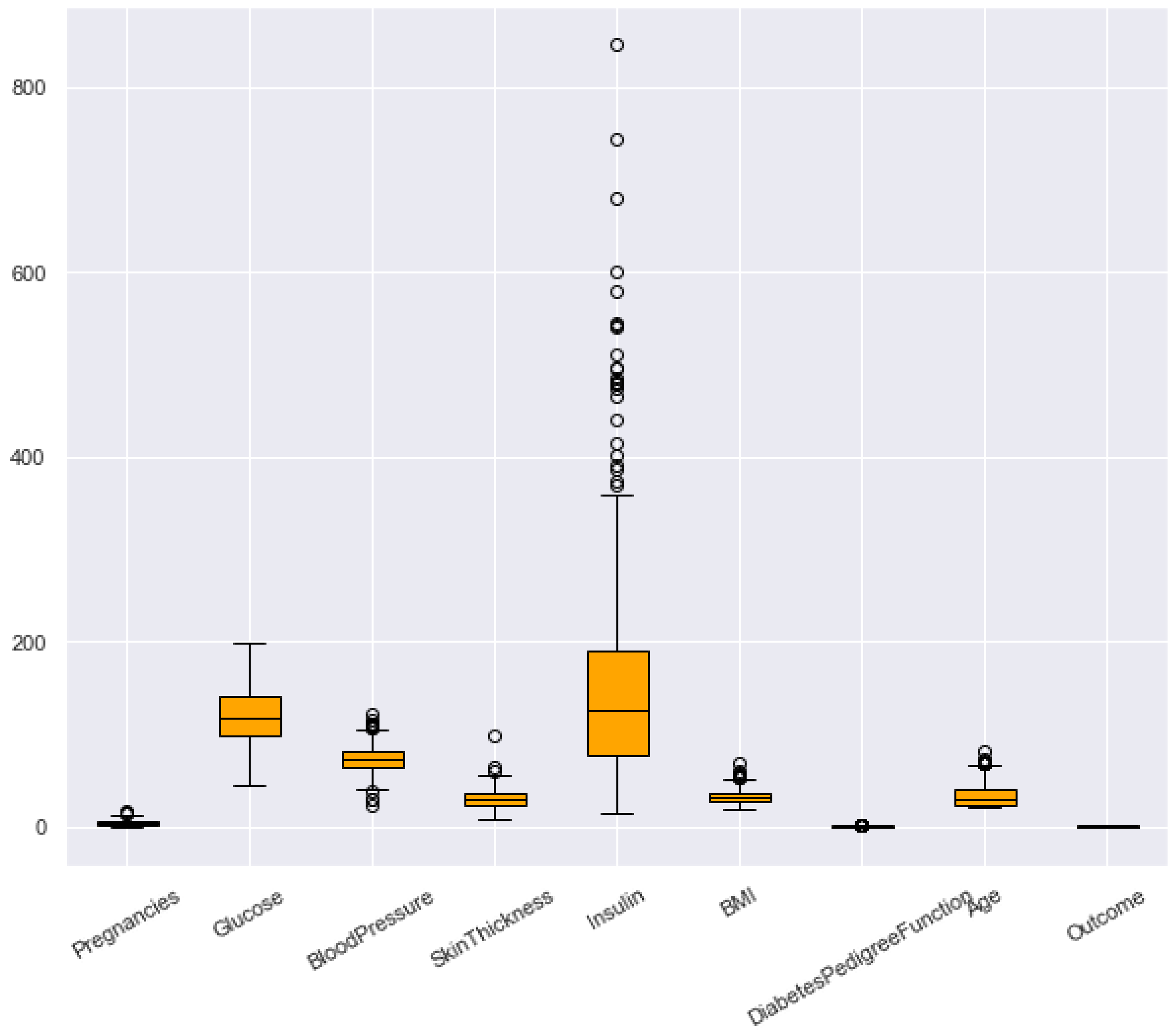
2) Data Cleaning

From the data exploration, we can see that there are several missing values which require to be handled. So in this part, we will replace the missing values `0` in Glucose, BloodPressure, SkinThickness, Insulin, and BMI columns. By first, converting the `0` in these columns to be `Nan`, then replacing them with the proper values. There are several ways to handle the missing values one of them is to replace missing values with the mean for the continuous variables or the mode for categorical variables. But if the variable contains outliers which will impact the mean, we will use the median instead. So we will first confirm the number of missing values in each column and then by using a boxplot, we will check if there are outliers in the data.

In [356…

```
data = diabetes.copy(deep = True)
#Replace 0 by NaN for all missing values
data[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = data[['Glucose','
#confirm missing values in each column
print(data.isna().sum())
# Plot box plot for each varable
plt.figure(figsize=(10,8))
c = 'black'
data.boxplot(patch_artist=True,boxprops=dict(facecolor = 'orange', color=c),
capprops=dict(color=c),
whiskerprops=dict(color=c),
flierprops=dict(color=c, markeredgecolor=c),
medianprops=dict(color= c))
plt.xticks(rotation=30)
plt.show()
```

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```



From the boxplot figure, we find that there is no outlier in 'Glucose' and a very small number of values of outliers in 'SkinThickness' column. So the mean will be applied to replace the missing values for these 2 columns and the median is for the remaining columns. First, the data is grouped by the Outcome value, and then the mean and median of each group are utilized to replace all missing values in each group.

In [357…

```
#Replace missing value in each column groupby the Outcome value by mean or median
data['Glucose'] = data['Glucose'].fillna(data.groupby('Outcome').transform('mean')['G
data['BloodPressure'] = data['BloodPressure'].fillna(data.groupby('Outcome').transfor
data['SkinThickness'] = data['SkinThickness'].fillna(data.groupby('Outcome').transfor
data['Insulin'] = data['Insulin'].fillna(data.groupby('Outcome').transform('median')[
data['BMI'] = data['BMI'].fillna(data.groupby('Outcome').transform('median')['BMI'])
#Reconfirm missing value
data.isna().sum()
```

Out[357]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Now, the data is cleaned, and the missing values are replaced by proper values. Next, we will separate data into 2 groups, train and test. Train data will be used to fit or train the model, while the test set is to evaluate or validate the model in order to confirm its performance.

3) Train/Test split

In [358…
```
# Full data
full_data = data.copy(deep = True)
# Make a list of the feature names
Features = full_data.columns.tolist()
# Separate data to training and test datasets
X=full_data.drop(['Outcome'],axis=1) # Input variables
X_names = Features[1:] # Remove 'Outcome' from feature names list
y=full_data['Outcome'] # Target variable
y_names=['Positive','Negative'] # Target labels
# Split data into training (80%) and test (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
```

Data is now ready to use, next the models will be implemented and adjusted.

Model implementation

All 6 models adopted in this study are the models studied in MA336 that can be applied to solve binary classification problems which are our problem here.
The first 5 models (Logistic Regression, KNN, Decision Tree, Random Forest, SVM) are `supervised machine learning` methods.
While the ANN is the only `Deep Learning` method implemented in this study.

1) Logistic Regression

From the nature of the binary classification problem, Logistic regression is a simple model to perform. So we will start with this model and its performance will be compared with other models.
First, the full logit model will be implemented, all features will be used to fit the model. Then, referring to the data exploration above, we will eliminate some features before fitting the model and compare the accuracy of the model.

In [359…
```
# Define the Logistic Regression model
logit = LogisticRegression()
# Train the model on the training dataset
logit.fit(X_train, y_train)
# Get the accuracy over the test data
score = logit.score(X_test, y_test)
print("Accuracy of Logistic Regression model:", round(score,4))
```

```
Accuracy of Logistic Regression model: 0.7727
```

In [360…
```
# Eliminate 2 features refering to EDA
X_train2 = X_train.drop(['SkinThickness','BloodPressure'],axis=1)
X_test2 = X_test.drop(['SkinThickness','BloodPressure'],axis=1)
X2 = X.drop(['SkinThickness','BloodPressure'],axis=1)
# Define the Logistic Regression model
logit2 = LogisticRegression()
# Train the model on the training dataset
logit2.fit(X_train2, y_train)
# Get the accuracy over the test data
score = logit2.score(X_test2, y_test)
print("Accuracy of Logistic Regression model:", round(score,4))
# Add model to the list, this will be used to evaluate the performance in Result sect
models = []
models.append(('Logistic Regression', logit2))
```

```
Accuracy of Logistic Regression model: 0.7857
```

The accuracy results show that the model which excludes 2 features ('SkinThickness' and 'BloodPressure') that influence the 'Outcome' the least has better accuracy than the full model. So we select this model to evaluate the performance and compare its performance with other models in the result section. Also, the remaining machine learning models (KNN, Decision Tree, Random Forest, SVM) will be implemented by using the dataset without 'SkinThickness' and 'BloodPressure'.

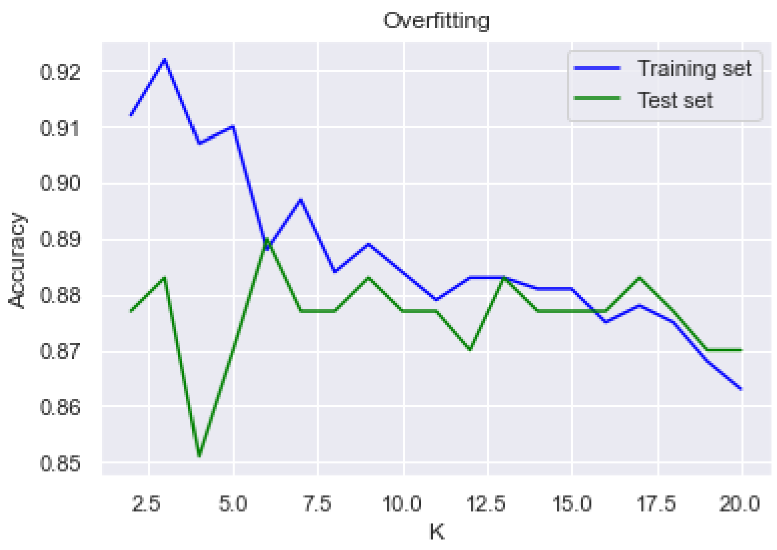
2) K Nearest Neighbors (KNN)

KNN is another simple model which can be applied for both classification and regression problems. In this study, KNN will be utilized to solve a binary classification problem. First, the model will be trained by using the default parameter `(n_neighbors = 5)`. After that, we will fit the model by trying several values of `n_neighbors` to define the value that makes the model more accurate.

In [361…
```
#Train knn model
knn2 = KNeighborsClassifier(n_neighbors=5)
knn2.fit(X_train2, y_train)
#Predict result on test dataset
ypred_train2 = knn2.predict(X_train2)
train_acc2 = round(metrics.accuracy_score(y_train,ypred_train2),3)
ypred_test2 = knn2.predict(X_test2)
test_acc2 = round(metrics.accuracy_score(y_test,ypred_test2),3)
print(test_acc2)
```

```
0.87
```

In [362…
```
k=[]
acctrain_list =[]
acctest_list = []
i=0
# Adjust k value from 2 to 20 & calculate the accuracy for each k value
for i in range(2,21):
    #train model
    knn3 = KNeighborsClassifier(n_neighbors=i)
    knn3.fit(X_train2, y_train)
    #predict using training dataset
    ypred_train = knn3.predict(X_train2)
    #calculate accuracy on training dataset
    train_acc = round(metrics.accuracy_score(y_train,ypred_train),3)
    #append accuracy on training dataset to list
    acctrain_list.append(train_acc)
    #predict using test dataset
    ypred_test = knn3.predict(X_test2)
    #calculate accuracy on test dataset
    test_acc = round(metrics.accuracy_score(y_test,ypred_test),3)
    #append accuracy on test dataset to list
    acctest_list.append(test_acc)
    k.append(i)
#plot accuracy on training and test dataset for each value of k
fig = plt.figure()
ax = plt.axes()
line1, = ax.plot(k,acctrain_list,color='blue',label='Training set')
line2, = ax.plot(k,acctest_list,color='green',label='Test set')
plt.legend(handles=[line1, line2])
plt.title("Overfitting")
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.show()
#show the maximum accuracy on test dataset
d = {'K':k,'Train':acctrain_list,'Test':acctest_list}
df = pd.DataFrame(d)
print('Maximum accuracy for Test data set is :\n',df[df.Test == df.Test.max()])
# Add model to the list, this will be used to evaluate the performance in Result sect
knn_f = KNeighborsClassifier(n_neighbors=6)
knn_f.fit(X_train2, y_train)
models.append(('KNN', knn_f))
```



```
Maximum accuracy for Test data set is :
    K  Train  Test
4  6  0.888  0.89
```

The result illustrates that the best accuracy on the test data is at K = 6, which the accuracy is about 89%, about 2% higher than the default value. Then the model with K = 6 will be used to evaluate the performance.

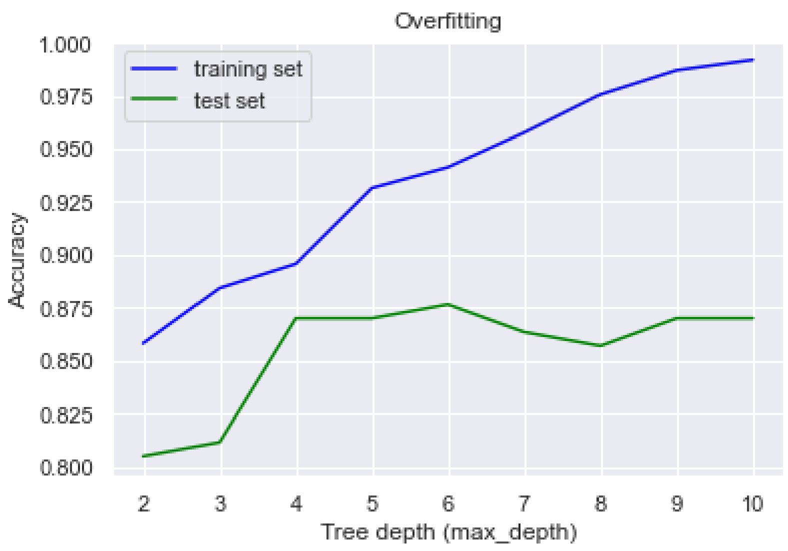
3) Decision Tree

The Decision Tree model can be applied also for both classification and regression problems. First, we will fit the model by fixing the random state `(random_state = 7)` to control the randomness involved in the mode and setting the maximum depth of the tree as its default value (None) which means the tree will expand until all data on the leaf are from the same group. The maximum depth value is to control the number of splits a tree makes before predicting (depth nodes from root to leaf) and it impacts the model performance. So maximum depth of the tree will be adjusted to find the value which improves the model performance.

In [363…
```
#set random state, this will be used for decision tree and random forest model
rd_st = 15
#train decision tree model
decision_tree = tree.DecisionTreeClassifier(max_depth = None, random_state=rd_st) # C
decision_tree.fit(X_train2,y_train)
#predict the result using test dataset
y_pred=decision_tree.predict(X_test2)
# Print the model accuracy on test dataset
print("Accuracy on test set:",np.round(metrics.accuracy_score(y_test, y_pred),4))
```

```
Accuracy on test set: 0.8766
```

In [364…
```
#clarify max depth to be test
max_depth_vals = [2,3,4,5,6,7,8,9,10]
accuracytrain_list=[]
accuracytest_list=[]
#Adjust max depth and finding the accuracy for each max depth
for i in range(0,len(max_depth_vals)):
    decision_tree2 = tree.DecisionTreeClassifier(max_depth=max_depth_vals[i], random_
    decision_tree2.fit(X_train2,y_train)
    # Calculate accuracy on training set
    ytrain_pred = decision_tree2.predict(X_train2)
    accuracy_train = metrics.accuracy_score(y_train, ytrain_pred)
    accuracytrain_list.append(accuracy_train)
    # Calculate accuracy on Test set
    ytest_pred = decision_tree2.predict(X_test2)
    accuracy_test = metrics.accuracy_score(y_test, ytest_pred)
    accuracytest_list.append(accuracy_test)

# Plot accuracy as a function of max_depth
fig = plt.figure()
ax = plt.axes()
line1, = ax.plot(max_depth_vals,accuracytrain_list,color='blue',label='training set')
line2, = ax.plot(max_depth_vals,accuracytest_list,color='green',label='test set')
plt.legend(handles=[line1, line2])
plt.title("Overfitting")
plt.xlabel("Tree depth (max_depth)")
plt.ylabel("Accuracy")
plt.show()
#show maximum accuracy
d = {'Max depth':max_depth_vals,'Train':accuracytrain_list,'Test':accuracytest_list}
df = pd.DataFrame(d)
print('Maximum accuracy for Test data set is :\n',df[df.Test == df.Test.max()])
# Add decision tree model to the list, this will be used to evaluate the performance
decision_tree_f = tree.DecisionTreeClassifier(max_depth=6, random_state=rd_st)
decision_tree_f.fit(X_train2,y_train)
models.append(('Decision Tree',decision_tree_f))
```



```
Maximum accuracy for Test data set is :
    Max depth     Train      Test
4           6  0.941368  0.876623
```

The graph above illustrates that at the Max depth = 6, we will have the best accuracy on the test dataset. So the model with this depth level is selected to be compared with other models.

4) Random Forest

A random forest is a multitude of decision trees which are trained on a different sample from the training dataset and the outcome is predicted based on the majority vote or mean. Same as other models, we will first fit the model by using the default value. Then parameters will be fine-tuned to get the best performance. The difference in max depth values will be tested, also the difference number of trees use to train the model.

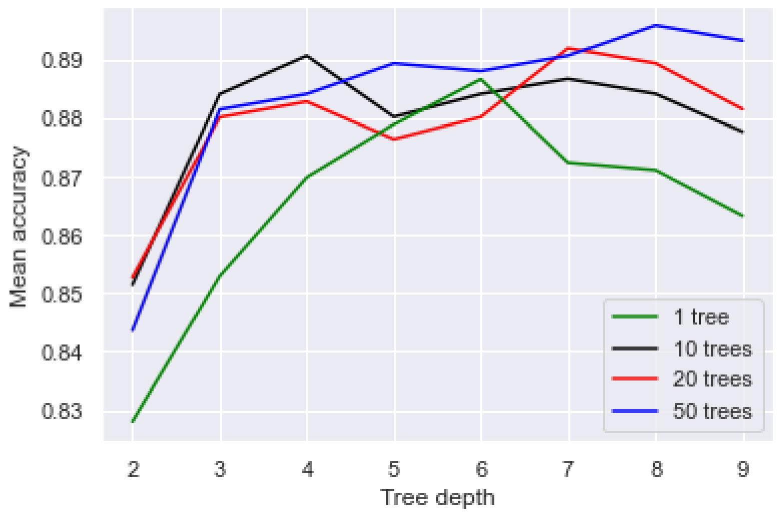
In [365…

```
#train random forest model
forest = RandomForestClassifier(n_estimators=100,bootstrap=True,max_features='auto',c
forest.fit(X_train2,y_train)
#predict the result and print accuracy on training and test dataset
y_train_pred = forest.predict(X_train2)
print("Accuracy on training set:",round(metrics.accuracy_score(y_train, y_train_pred)
y_test_pred = forest.predict(X_test2)
print("Accuracy on test set:",round(metrics.accuracy_score(y_test, y_test_pred),3))
```

```
Accuracy on training set: 1.0
Accuracy on test set: 0.883
```

In [366…

```
# clarify max depth and no. of tree (parameters to be tested)
max_depth_vals = [2,3,4,5,6,7,8,9]
n_estimators_vals = [10,20,50]
mean_accuracy_store = []
sd_accuracy_store = []
k=10
#Adjust max depth, no of tree and finding the accuracy for each parameter using 5-fol
for i, value in enumerate(n_estimators_vals):
    mean_accuracy_cv = []
    sd_cv = []
    for val in max_depth_vals:
        forest = RandomForestClassifier(n_estimators=value,bootstrap=True,max_feature
        cv_scores = cross_val_score(forest, X2, y, cv=k)
        avg = sum(cv_scores)/len(cv_scores)
        sd = math.sqrt(sum((cv_scores-avg)**2)/(len(cv_scores)-1))
        mean_accuracy_cv.append(avg)
        sd_cv.append(sd)
    mean_accuracy_store.append(mean_accuracy_cv)
    sd_accuracy_store.append(sd_cv)
#plot the result
fig = plt.figure()
ax = plt.axes()
line2, = ax.plot(max_depth_vals,mean_accuracy_store[0],color='black',label='10 trees'
line3, = ax.plot(max_depth_vals,mean_accuracy_store[1],color='red',label='20 trees')
line4, = ax.plot(max_depth_vals,mean_accuracy_store[2],color='blue',label='50 trees')
# Find accuracy scores as a function of tree depth for a single decision tree
mean_accuracy_cv = []
for i in range(0,len(max_depth_vals)):
    decision_tree = tree.DecisionTreeClassifier(max_depth=max_depth_vals[i], random_s
    cv_scores = cross_val_score(decision_tree, X2, y, cv=k)
    avg = sum(cv_scores)/len(cv_scores)
    mean_accuracy_cv.append(avg)
line1, = ax.plot(max_depth_vals,mean_accuracy_cv,color='green',label='1 tree')
plt.legend(handles=[line1, line2, line3,line4])
plt.xlabel("Tree depth")
plt.ylabel("Mean accuracy")
plt.show()
# Add random forest model to the list, this will be used to evaluate the performance
forest_fin = RandomForestClassifier(n_estimators=50,bootstrap=True,max_features='auto
forest_fin.fit(X_train2,y_train)
models.append(('Random Forest',forest_fin))
```



The best accuracy from 10-fold cross validation can get when we use 50 trees and the maximum depth equals to 8. So the model with these values will be selected to evaluate.

5) Support Vector Machine (SVM)

SVM is another model that can solve both regression and classification problems. It is most commonly used for classification. So this model will also be implemented to solve our binary classification problem and compared the result with other models. In SVM model, kernel will be the parameter which will be adjusted to find the best model.

In [367…
```
#train svm model
SVM_model = svm.SVC(kernel='rbf')
# Fit data
SVM_model = SVM_model.fit(X_train2, y_train)
predictions = SVM_model.predict(X_train2)
train_accc = round(metrics.accuracy_score(y_train,predictions),3)
print("Accuracy on train set:",train_accc)
SVM_predict = SVM_model.predict(X_test2)
test_accc = round(metrics.accuracy_score(y_test,SVM_predict),3)
print("Accuracy on test set:",test_accc)
```

```
Accuracy on train set: 0.85
Accuracy on test set: 0.851
```

In [368…
```
# define kernel to be tested
ker = ['linear', 'poly', 'rbf', 'sigmoid']
#Adjust kernel and calculate accuracy on each kernel
for i in range (0,len(ker)):
    SVM_model2 = svm.SVC(kernel=ker[i])
    # Fit data
    SVM_model2 = SVM_model2.fit(X_train2, y_train)
    predictions = SVM_model2.predict(X_train2)
    train_accc = round(metrics.accuracy_score(y_train,predictions),3)
    SVM_predict = SVM_model2.predict(X_test2)
    test_accc = round(metrics.accuracy_score(y_test,SVM_predict),3)
    #shown output accuracy
    print(f"Accuracy on test set when kernel = {ker[i]} : {test_accc}")
# Add SVM model to the list, this will be used to evaluate the performance in Result
models.append(('SVM',SVM_model))
```

```
Accuracy on test set when kernel = linear : 0.792
Accuracy on test set when kernel = poly : 0.825
Accuracy on test set when kernel = rbf : 0.851
Accuracy on test set when kernel = sigmoid : 0.409
```

Based on the finding accuracies listed above, we found that the 'rbf' kernel which is the default value makes the model accuracy best. Then the model with 'rbf' kernel is taken to compare with others.

6) Artificial Neural Network (ANN)

ANN is the only `Deep Learning` method used in this report. Since in Deep Learning methodolody, the feature engineering process is not required, the raw input data can be used as a model input. So the model will be fitted by using full dataset. Several parameters in ANN can be tuned to increase the performance of the model. Here, number of layers,

In [369…
```
##### clear_session()
epochs = 10
batch_size = 100
# make the model
ann_model = Sequential()

ann_model.add(layers.Dense(12, input_dim=X_train.shape[1], activation='relu'))
ann_model.add(layers.Dense(1, activation='sigmoid'))
ann_model.summary()
ann_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

### 2. Training (fit) ###
history = ann_model.fit(X_train, y_train,batch_size = batch_size, epochs = epochs,val
loss, accuracy = ann_model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy:",round(accuracy,2))
loss, accuracy = ann_model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

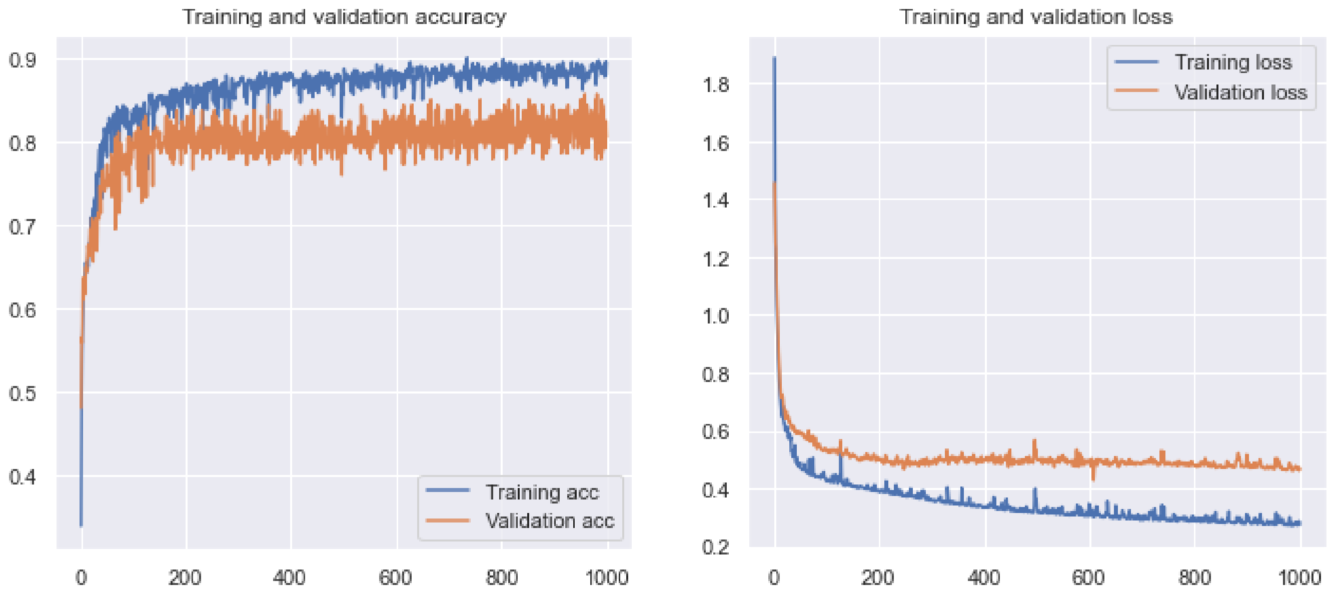
```
Model: "sequential_30"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_90 (Dense)             (None, 12)                108       
_________________________________________________________________
dense_91 (Dense)             (None, 1)                 13        
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data ad
apter that can handle input: <class 'pandas.core.frame.DataFrame'>, <class 'NoneTyp
e'>
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data ad
apter that can handle input: <class 'pandas.core.frame.DataFrame'>, <class 'NoneTyp
e'>
Training Accuracy: 0.65
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data ad
apter that can handle input: <class 'pandas.core.frame.DataFrame'>, <class 'NoneTyp
e'>
Testing Accuracy:  0.6494
```

In [370…

```
epochs = 1000
batch_size = 100
# make the model
ann_model2 = Sequential()
ann_model2.add(layers.Dense(12, input_dim=X_train.shape[1], activation='relu'))
ann_model2.add(layers.Dense(10, activation='relu'))
ann_model2.add(layers.Dense(8, activation='relu'))
ann_model2.add(layers.Dense(1, activation='sigmoid'))
ann_model2.summary()
ann_model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']
# Create function to plot history
def plot_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    x = range(1, len(acc) + 1)
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(x, acc, label='Training acc')
    plt.plot(x, val_acc, label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(x, loss, label='Training loss')
    plt.plot(x, val_loss, label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
### 2. Training (fit) ###
history2 = ann_model2.fit(X_train, y_train,batch_size = batch_size, epochs = epochs,v
loss, accuracy = ann_model2.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy:",round(accuracy,2))
loss, accuracy = ann_model2.evaluate(X_test, y_test, verbose=False)
```

```
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history2)
ann_pred = ann_model2.predict(X_test).round()
```

```
Model: "sequential_31"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_92 (Dense)             (None, 12)                108       
_________________________________________________________________
dense_93 (Dense)             (None, 10)                130       
_________________________________________________________________
dense_94 (Dense)             (None, 8)                 88        
_________________________________________________________________
dense_95 (Dense)             (None, 1)                 9         
=================================================================
Total params: 335
Trainable params: 335
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data ad
apter that can handle input: <class 'pandas.core.frame.DataFrame'>, <class 'NoneTyp
e'>
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data ad
apter that can handle input: <class 'pandas.core.frame.DataFrame'>, <class 'NoneTyp
e'>
Training Accuracy: 0.88
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data ad
apter that can handle input: <class 'pandas.core.frame.DataFrame'>, <class 'NoneTyp
e'>
Testing Accuracy:  0.8052
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data ad
apter that can handle input: <class 'pandas.core.frame.DataFrame'>, <class 'NoneTyp
e'>
```



From the figure above, we can see that both the accuracy curve and loss curve keep slightly fructuate even the epoch reaches 1000. It could illustrate that this model is not good fitted to this data set. It might be because the dataset does contain not enough data points for this kind of this learning model or it could be the method we use to clean or prepare the dataset. Another possible cause might be the tuned parameters are not proper for this dataset.

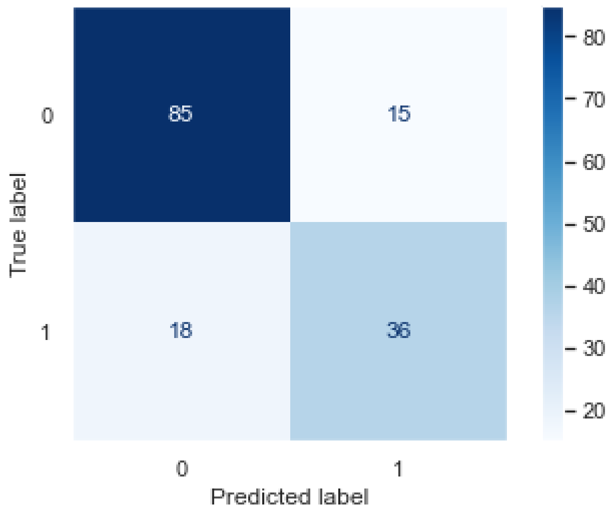
III. Result

In this section, the performance of all models will be evaluated and compared. The tuned models of each methodology are selected and appended to a list. Then they will be assessed, a confusion matrix will be used to demonstrate model performance, the accuracy, recall, specificity, and precision will be calculated and also k-fold cross validation method will be measured to be another method to evaluate the models.

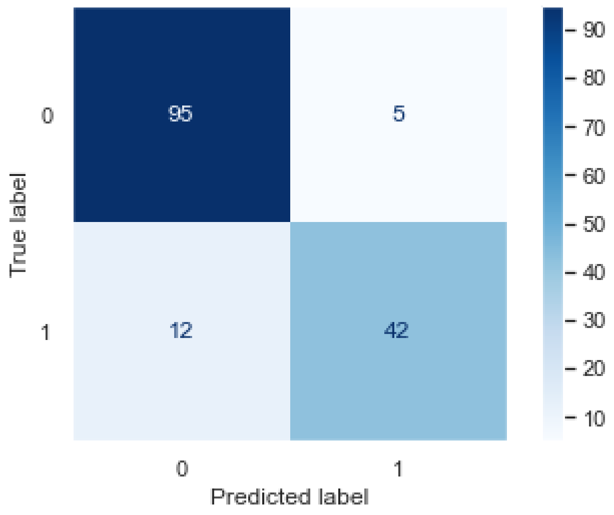
In [371…

```python
model_list=[]
acc_list=[]
recall_list=[]
# spec_list=[]
# pre_list=[]
cv_list=[]
# Evaluate machine learning method
for name, model in models:
    model_list.append(name)
    print(f'Confusion matrix for {name} model')
    plot_confusion_matrix(model, X_test2, y_test,cmap="Blues")
    plt.grid(False)
    plt.show()
    cm = metrics.confusion_matrix(y_test,model.predict(X_test2))
    tn, fp, fn, tp = cm.ravel()
    # Assigning columns names
    accuracy = round((tp+tn)/(tp+tn+fp+fn),2)
    acc_list.append(accuracy)
    recall = round(tp/(tp+fn),2)
    recall_list.append(recall)
    specificity = round(tn/(tn+fp),2)
    spec_list.append(specificity)
    precision = round(tp/(tp+fp),2)
    pre_list.append(precision)
    cv_scores = cross_val_score(model, X2, y, cv=10)
    avg = round((sum(cv_scores)/len(cv_scores)),2)
    cv_list.append(avg)

#Evaluate ANN Deep learning method
model_list.append('ANN')
print(f'Confusion matrix for ANN model')
cm_ann = metrics.confusion_matrix(y_test, ann_pred)
ann_df = pd.DataFrame(cm_ann,
            columns = ['Predicted Negative', 'Predicted Positive'],
            index = ['Actual Negative', 'Actual Positive'])
print(ann_df)
tn, fp, fn, tp = cm_ann.ravel()
accuracy = round((tp+tn)/(tp+tn+fp+fn),2)
acc_list.append(accuracy)
recall = round(tp/(tp+fn),2)
recall_list.append(recall)
# specificity = round(tn/(tn+fp),2)
# spec_list.append(specificity)
# precision = round(tp/(tp+fp),2)
# pre_list.append(precision)
cv_list.append('NA')

# dataframe algorithm comparison
performance = pd.DataFrame({
    'Model' : model_list,
    'Accuracy' : acc_list,
    '10-folds cv': cv_list,
    'Recall' : recall_list,
    # 'Specificity':spec_list,
    # 'Precision':pre_list

})
display(performance)
```

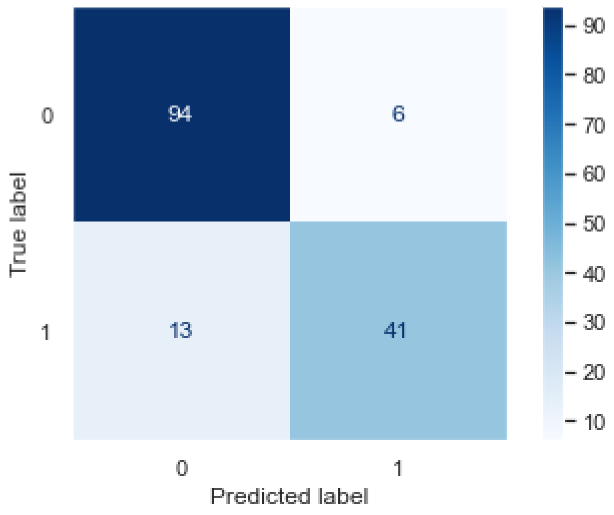
```
Confusion matrix for Logistic Regression model
```



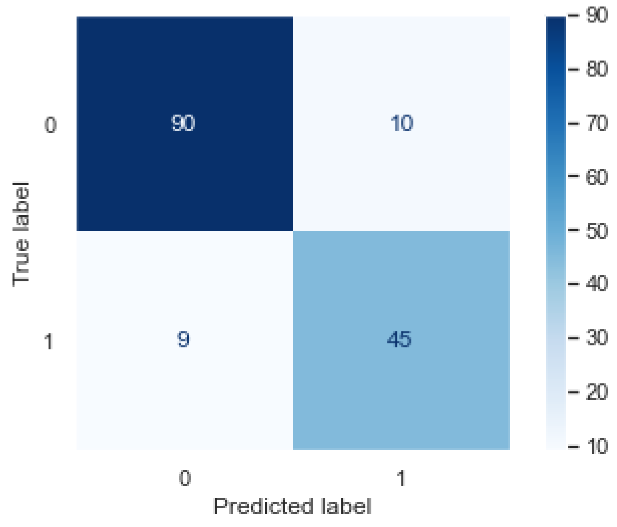
```
Confusion matrix for KNN model
```



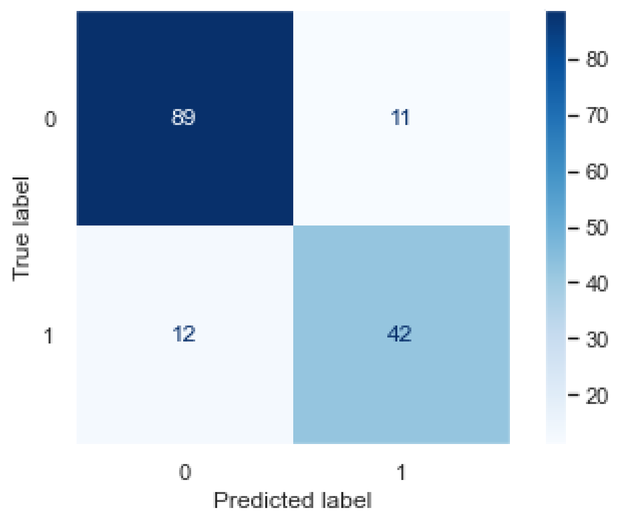
```
Confusion matrix for Decision Tree model
```



```
Confusion matrix for Random Forest model
```



```
Confusion matrix for SVM model
```



```
Confusion matrix for ANN model
                 Predicted Negative  Predicted Positive
Actual Negative                  87                  13
Actual Positive                  17                  37
```

	Model	Accuracy	10-folds cv	Recall
0	Logistic Regression	0.79	0.78	0.67
1	KNN	0.89	0.87	0.78
2	Decision Tree	0.88	0.89	0.76
3	Random Forest	0.88	0.89	0.83
4	SVM	0.85	0.84	0.78
5	ANN	0.81	NA	0.69

From the summary result table above, we can see that the KNN model gives the best accuracy by using the normal prediction of the test set. Whilst with the 10-fold cross-validation method, the model that presents the best performance is the Random Forest. But since the label of the dataset is unbalanced, it is important to check the Recall or Sensitivity or true positive rate. Because the goal of this study is to predict diabetic patients as much as we can, so they can process for further treatment. In this case, Random Forest is the best candidate as it has the highest recall value. The Logistic Regression which is the most simple one shows the least accuracy and recall here. Even the 10-fold cv is not calculated here but from the accuracy and recall rate, we can say that the only deep learning method here (ANN) does not represent a good performance compared with others. It might be because the dataset is not big enough to train the model. Among all model with parameter adjusting in this study, giving the result that the Random Forest is the model which fit this dataset best.

IV. Conclusion

Not just only for diabetes, it is important to early diagnose all diseases so we can get early treatment and increase the chance of recovering. And from the study result, we can ensure that the predictors which are Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, and Age are influenced the output variable which is Outcome.The best model in this study is Random Forest which give about 89% of accuracy when evaluate withe 10-fold cross validation and shows the highest recall rate at 85%. There are several methodologies to solve the binary classification problem and many methods to measure their performance. This report only demonstrates some of them. The factor to make the model accurate is not just only the model creation process, but exploratory data analysis, data cleaning, feature engineering, and parameter tuning, all processes are important to make a better model. The model performance getting in this report is also not the best performance we can get. We can invest more time to prepare the data or adjust the parameters to improve the model but we should weigh between what we will gain and what to lose.

References

International Diabetes Federation. (2021). IDF Diabetes Atlas | Tenth Edition. Diabetesatlas.org. https://diabetesatlas.org/

Mehmet Akturk. (2020). Diabetes Dataset. Kaggle.com. https://www.kaggle.com/datasets/mathchi/diabetes-data-set

The British Diabetic Association operating as Diabetes UK. (2022). Diabetes UK - Know diabetes. Fight diabetes. | Diabetes UK. Diabetes.org.uk. https://www.diabetes.org.uk/

Notebook by Khwanchanok Chumkhun, Registration Number : 2110866, **University of Essex**