Using Machine Learning to Predict Diabetes

1. Introduction

According to the CDC's studies, over 37 million US citizens suffer from diabetes and 1 out of 5 of them have it without even knowing it (Center for Disease Control and Prevention, 2022). In this project we are using machine learning to predict occurrence of diabetes. This project could be applied in medical diagnosis or health surveys to give directional information about whether a person has diabetes or not.

This report will discuss problem formulation, compare different machine learning methods and analyze the results. Section 2 focuses on problem formulation and information about the data used in this project. Section 3 then discusses two possible machine learning methods, Logistic Regression and Decision Tree, that can be used to solve our problem. Section 4 presents the results, and in Section 5 we made conclusions based on them.

2. Problem Formulation

2.1 The Problem

The problem we are solving is whether it would be possible to predict a person's risk of having diabetes by using machine learning. In this project we are using supervised machine learning methods to solve the problem.

2.2 Dataset

Our dataset is from the National Institute of Diabetes and Digestive and kidney diseases (obtained from Kaggle). In our dataset, one datapoint represents one subject. One datapoint contains a few factors that could affect person's probability to have diabetes and the information whether they have diabetes. The different factors are in the form of numeric data, either floats or integers, and whether they have diabetes is binarized data. The dataset includes columns:

- "Pregnancies": number of pregnancies
- "Glucose": plasma glucose concentration from an oral glucose tolerance test
- "BloodPressure": diastolic blood pressure measured in mmHg

- “SkinThickness”: skin thickness measured from triceps in mm,
- “Insulin”: 2-hour serum insulin in muU/ml,
- “BMI”: body mass index in kg/m^2
- “DiabetesPedigreeFunction”: diabetes pedigree function, likelihood of diabetes based on family history
- “Age”: age in years
- “Outcome”: a label column binarized cases of diabetes, 1 meaning the person has diabetes and 0 meaning not having

2.3 Feature Engineering

Throughout this project we use label vector (y) and feature matrix (X). Label vector consists of “Outcome” values. The data has 8 possible features from which we chose to focus on “BMI”, “Age”, “Glucose” and “BloodPressure” as the feature matrix. We selected these four features, because according to THL, these are the biggest contributing factors for diabetes out of the dataset’s possible features (Terveyden ja hyvinvoinnin laitos, 2022). We also chose not to use the diabetes pedigree function and number of pregnancies, because we found them difficult to binarize for the decision tree model.

3. Methods

3.1 Data

The dataset consists of 768 over 21-year-old women of Pima Indian heritage (Kaggle, 2020). Since there is some information missing, the total number of usable datapoints is 724. As mentioned earlier, our feature matrix is made of the four chosen risk factors: BMI, age, blood pressure and glucose. Since the column “Outcome” is already binarized into ones and zeros, we can use it as the label vector as it is. About one third of the outcome values are diabetes positive. The methods we will use in this project are logistic regression and decision tree.

We formed the training set of 70% of the datapoints chosen at random. The rest of the data was further divided in half between validation and training sets. We chose such design, because it is a commonly used ratio (V7 labs, 2022).

3.2 Logistic regression

Since we want to predict occurrence of diabetes, we can use the binary data 0 meaning “no diabetes” and 1 meaning “diabetes” and use logistic regression model as the machine learning method. As a model logistic regression seems like a good option, because it is a binary classification method (A. Jung, 2022).

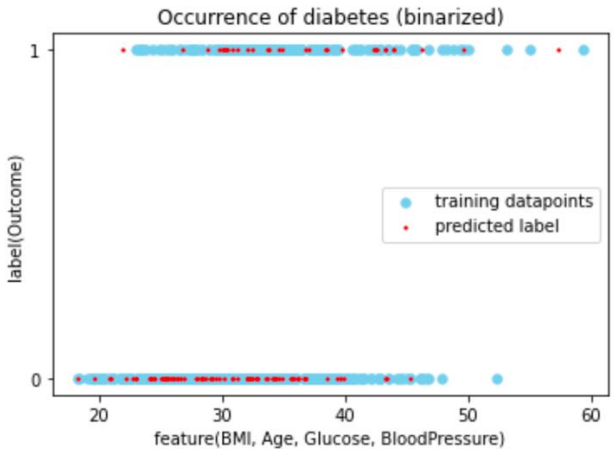


Figure 1: Logistic Regression

We also computed a confusion matrix to examine the performance of the regression.

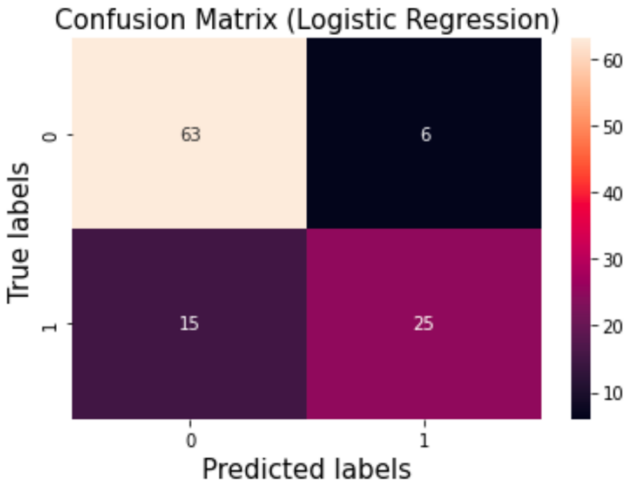


Figure 2: The confusion matrix of logistic regression

As the loss function, we used the logistic loss, later discussed in the report.

3.3 Decision tree

As the other machine learning model, we decided to use the decision tree method. We chose this model because it is suitable for classification problems. Also, the logic behind decision tree is easy to understand because decision tree tends to mimic human thinking ability (Javatpoint, n.d.).

To avoid overfitting, we decided to binarize all data. We binarized the data by turning all values over the risk limit into ones and under the limit into zeros. The limit values are defined as follows:

- BMI: overweight limit is 25.0 (Centers for Disease Control and Prevention, 2022)
- Glucose: values over 140 indicate pre-diabetes or diabetes (Diabetes Education Online, 2010)
- Blood pressure: the limit of high blood pressure is 80 (Centers for Disease Control and Prevention, 2021)
- Age: the limit of risk age is 40 (Diabetesliitto, 2022)

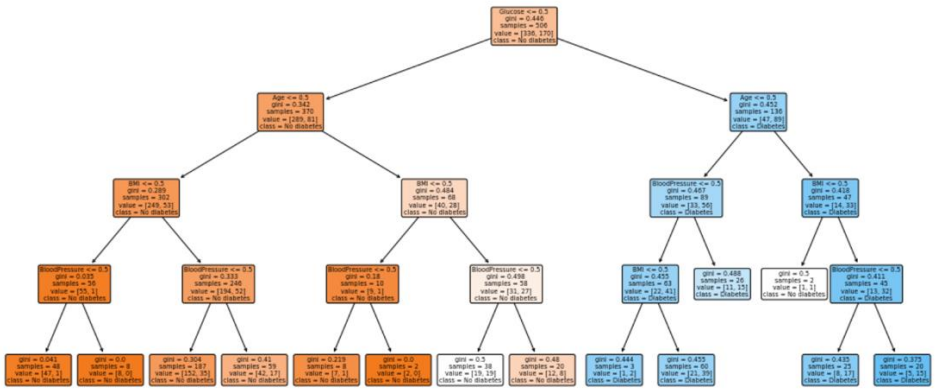


Figure 3: Decision Tree

For decision tree we also computed a confusion matrix to examine the model's performance.

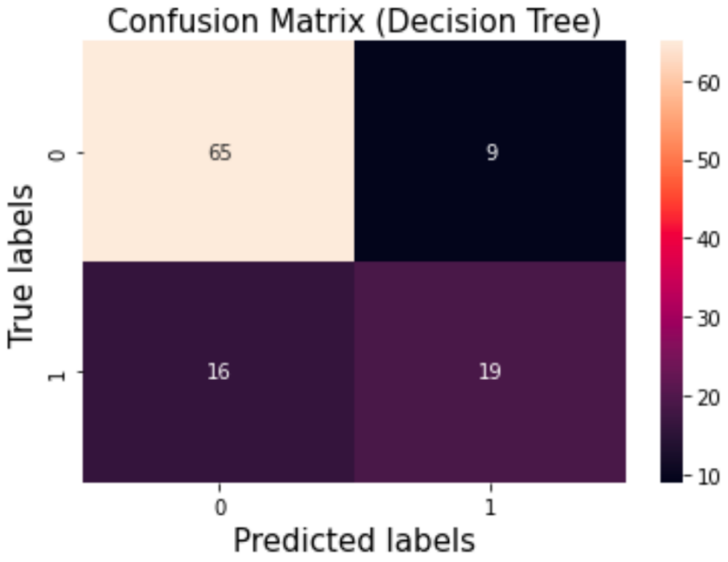


Figure 4: the Confusion Matrix of Decision Tree Classifier

As the loss function, we used the logistic loss, later discussed in the report.

4. Results

4.1 Choosing between the methods

To choose between the machine learning methods, logistic regression and decision tree classifier, we are comparing the training and validation errors. We calculated the errors by using accuracy scores: error = 1 – accuracy.

Figure 5: Comparison of training and validation errors

As shown in the figure above, both models seem to perform quite similarly. For both models the training error is around 25% and validation error around 20%. The accuracy varies between each run, because the sets are chosen randomly. However, the logistic regression seems to outperform decision tree model in every execution of the code. Because the differences between training and validation errors are not significant, we also compared the loss functions.

4.3 Loss Functions

As the loss function, we used logistic loss, because it is suitable for both logistic regression and decision tree classifier, since they are both used in classification problems (analytics Vidhya, 2022). Logistic loss is calculated with the following formula (Towards Data Science, 2019):

$$LogLoss = -\frac{1}{n}\sum_{k=0}^{n}[y_i log(y_i) + (1 - y_i) log(1 - y_i)]$$

Logistic loss function is already defined in the sklearn library. Therefore, it is also convenient to use it as the loss function.

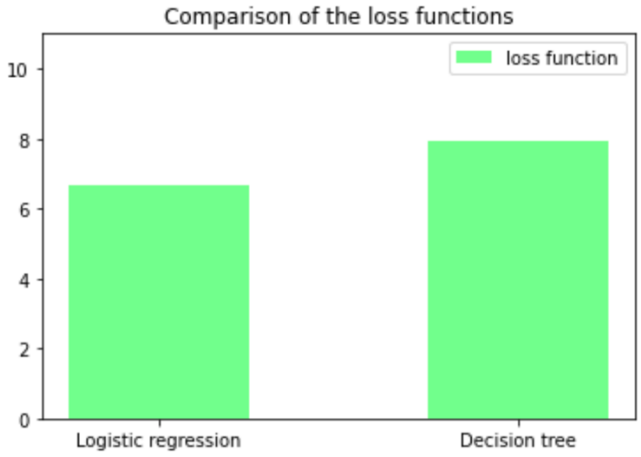


Figure 6: Comparison of the loss functions

As seen from the figure above, logistic regression's loss function appears to indicate a more accurate outcome. The models perform very similarly, but logistic regression seems to perform a little better than decision tree.

All things considered, we decided to choose logistic regression as our final method.

4.4 A test set

To test the performance of our chosen method, we are using a separate set of the data called the test set. As mentioned earlier in the report, the test set consist of 15% of randomly chosen values excluding values in training and validation sets.

Since the test error is the average loss on a test set, we calculated it by using logistic loss function with the test set. The test error of the logistic regression model is 8,24.

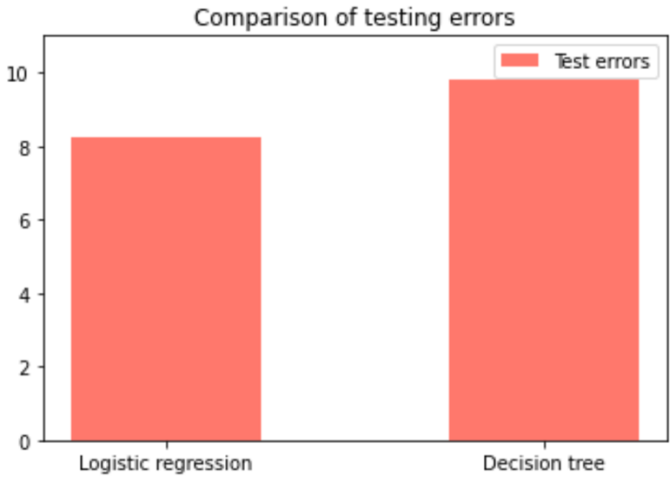


Figure 7: Comparison of testing errors

5. Conclusion

5.1 Report summary

This report applied two classification methods, logistic regression and decision tree model, to predict occurrence of diabetes. According to Figure 5 and Figure 6, logistic regression seems to outperform the decision tree classifier. The errors and loss function values are lower in logistic regression model, meaning that this model makes better predictions.

The results don't imply at overfitting, because the training error was higher than the validation error. There is no reason to assume that the training accuracy of our model is too high.

Figures 2 and 4 are visualizations of the models' confusion matrices. We can see, that both models tend to predict quite a lot of false negative results. However, as the matrices show, the predictions

are accurate. This indicates, that when the prediction is off, both models tend to predict a false negative result rather than false positive result.

5.2 Future improvements

Because of the relatively high test error value, shown in the figure 7, the results may not be very optimal and there is some room for improvement. To further improve the method, we would collect more training data and use more features of datapoints, as in the impact of genetics.

Also, the dataset used in this project never explicitly separates type 1 and type 2 diabetes. This might be reason for inaccuracies of our model, because the application of this project would be more suitable for predicting type 2 diabetes, since type 2 mainly depends on things one can control, as in for example lifestyle.

6. References

Center for Disease Control and Prevention (2022) Diabetes Fast Facts. Available at: https://www.cdc.gov/diabetes/basics/quick-facts.html (Accessed: 23 September 2022)

Kaggle (2020) Diabetes Dataset. Available at: https://www.kaggle.com/datasets/mathchi/diabetes-data-set (Accessed: 22 September 2022)

A. Jung, "Machine Learning: The Basics," Springer, Singapore, 2022. Available at: https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf (Accessed: 22 September 2022)

Terveyden ja hyvinvoinnin laitos (2022) Tyypin 2 diabeteksen riskitekijät. Available at: https://thl.fi/fi/web/kansantaudit/diabetes/tyypin-2-diabeteksen-riskitekijat (Accessed: 4 October 2022)

Centers for Disease Control and Prevention (2022) Defining Adult Overweight & Obesity. Available at: https://www.cdc.gov/obesity/basics/adult-defining.html (Accessed: 4 October 2022)

Diabetes Education Online (2010) Diagnosing Diabetes. Available at: https://dtc.ucsf.edu/types-of-diabetes/type2/understanding-type-2-diabetes/basic-facts/diagnosing-diabetes/ (Accessed: 4 October 2022)

Centers for Disease Control and Prevention (2021) High Blood Pressure Symptoms and Causes. Available at: https://www.cdc.gov/bloodpressure/about.htm (Accessed: 4 October 2022)

Diabetesliitto (2022) Tyypin 2 diabetes. Available at: https://www.diabetes.fi/diabetes/tyypin_2_diabetes (Accessed: 4 October 2022)

Wikipedia (2022) Decision Tree. Available at: https://en.wikipedia.org/wiki/Decision_tree (Accessed: 4 October 2022)

Javapoint (n. d.) Decision Tree Classification Algorithm. Available at: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm (Accessed: 4 October 2022)

V7 labs (2022) Train Test Validation Split: How To & Best Practices [2022]. https://www.v7labs.com/blog/train-validation-test-set (Accessed: 6 October 2022)

Towards Data Science (2020) What are Loss Functions?. Available at: https://towardsdatascience.com/what-is-loss-function-1e2605aeb904 (Accessed: 8 October 2022)

Analytics Vidhya (2022) Binary Cross Entropy aka Log Loss-The cost function used in Logistic Regression. Available at: https://www.analyticsvidhya.com/blog/2020/11/binary-cross-entropy-aka-log-loss-the-cost-function-used-in-logistic-regression/ (Accessed: 9 October 2022)

Towards Data Science (2019) The proper way to use Machine Learning Metrics. Available at: https://towardsdatascience.com/the-proper-way-to-use-machine-learning-metrics-4803247a2578 (Accessed: 9 October 2022)

7. Appendix

Machine Learning Project

October 9, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, log_loss
from sklearn.model_selection import train_test_split
from sklearn import tree
```

[2]:
```
# function to filter out missing information
def drop_empty_info(df):
    df = df[df['BMI'] != 0]
    df = df[df['Age'] != 0]
    df = df[df['BloodPressure'] != 0]
    df = df[df['Glucose'] != 0]
    return df
```

[3]:
```
def bar_height(y):
    neg = 0
    pos = 0
    for i in y:
        if i == 1:
            pos += 1
        else:
            neg +=1
    return neg, pos
```

[4]:
```
# converting the data into a dataframe and dropping unnecessary columns
diabetesData = pd.read_csv("diabetes.csv")
diabetesData = diabetesData.drop(columns=['Pregnancies', 'SkinThickness',␣
 ↪'Insulin', 'DiabetesPedigreeFunction'])
diabetesData = drop_empty_info(diabetesData)

# initializing lists to help with histograms
val_errors = []
test_errors = []
train_errors = []
```

```
loss = []
```

[5]:
```
def logistic_regression(diabetesData):
    X = diabetesData[['BMI', 'Age', 'Glucose', 'BloodPressure']].to_numpy()
    y = diabetesData['Outcome'].to_numpy()

    # separating the data into training data 70% and testing and validation␣
 ↪data overall 30%
    X_train, X_test_val, y_train, y_test_val = train_test_split(X, y,␣
 ↪train_size=0.7)
    X_test, X_val, y_test, y_val = train_test_split(X_test_val, y_test_val,␣
 ↪train_size=0.5)

    # using logistic regression method to examine the data
    clf = LogisticRegression().fit(X_train, y_train)

    # computing predicted labels with training, validation and testing sets
    y_pred_val = clf.predict(X_val)
    y_pred_test = clf.predict(X_test)
    y_pred_train = clf.predict(X_train)

    # calculationg the accuracy of the predicted labels
    test_accuracy = accuracy_score(y_test, y_pred_test)
    val_accuracy = accuracy_score(y_val, y_pred_val)
    train_accuracy = accuracy_score(y_train, y_pred_train)

    # calculating and saving the prediction errors
    train_errors.append(1 - train_accuracy)
    val_errors.append(1 - val_accuracy)
    test_loss = log_loss(y_test, y_pred_test)
    test_errors.append(test_loss)

    print("Logistic Regression:")
    print("Training accuracy of the logistic regression is {:.0f} %".
 ↪format(train_accuracy * 100))
    print("Validation accuracy of the logistic regression is {:.0f} %".
 ↪format(val_accuracy*100))
    print("Training error of the logistic regression is {:.0f} %".format((1 -␣
 ↪train_accuracy) * 100))
    print("Validation error of the logistic regression is {:.0f} %".format((1 -␣
 ↪val_accuracy) * 100))

    # visualizing the logistic regression
    ax1 = plt.subplot()

    ax1.set_xlabel("feature(BMI, Age, Glucose, BloodPressure)")
```

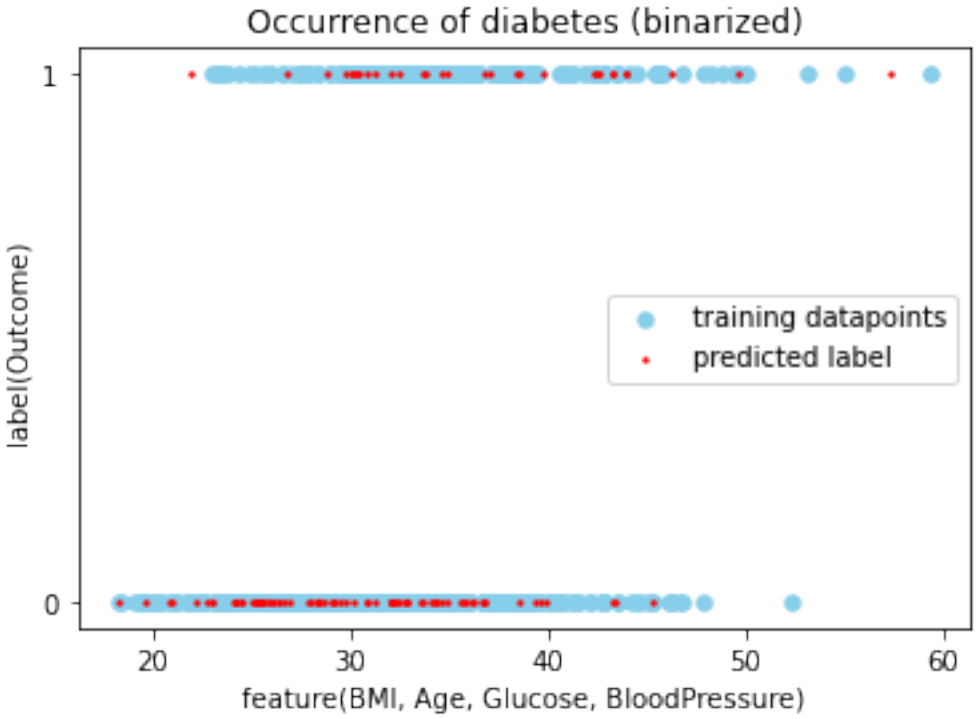
```
    ax1.set_yticks([0, 1])
    ax1.set_ylabel('label(Outcome)')
    ax1.set_title("Occurrence of diabetes (binarized)")
    ax1.scatter(X_train[:, 0], y_train, color='skyblue', s=30, label='training␣
↪datapoints')
    ax1.scatter(X_val[:, 0], y_pred_val, color='r', s=2, label='predicted␣
↪label')
    ax1.legend()
    plt.show()

    # computing a confusion matrix to examine the accuracy
    ax2 = plt.subplot()
    conf_mat = confusion_matrix(y_val, y_pred_val)
    sns.heatmap(conf_mat, annot=True, fmt='g', ax=ax2)
    ax2.set_xlabel('Predicted labels', fontsize=15)
    ax2.set_ylabel('True labels', fontsize=15)
    ax2.set_title('Confusion Matrix (Logistic Regression)', fontsize=15)
    plt.show()

    # logistic regression's loss function
    loss_val = log_loss(y_val, y_pred_val)
    loss.append(loss_val)
```

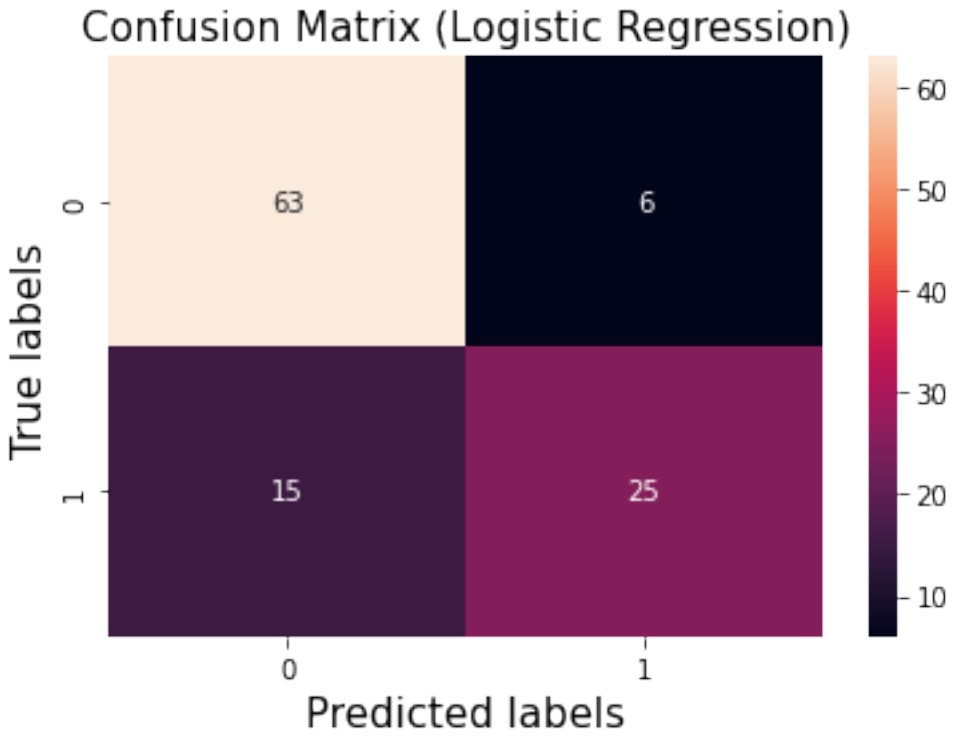
[6]:
```
def decision_tree(diabetesData):
    binarized_BMI = np.where(diabetesData['BMI'] >= 25, 1, 0)
    binarized_Age = np.where(diabetesData['Age'] > 40, 1, 0)
    binarized_Glucose = np.where(diabetesData['Glucose'] >= 140, 1, 0)
    binarized_BloodPressure = np.where(diabetesData['BloodPressure'] >= 80, 1,␣
↪0)

    diabetesData.insert(5, 'BinarizedBMI', binarized_BMI)
    diabetesData.insert(6, 'BinarizedAge', binarized_Age)
    diabetesData.insert(7, 'BinarizedGlucose', binarized_Glucose)
    diabetesData.insert(8, 'BinarizedBloodPressure', binarized_BloodPressure)

    X = diabetesData[['BinarizedBMI', 'BinarizedAge', 'BinarizedGlucose',␣
↪'BinarizedBloodPressure']]
    y = diabetesData['Outcome']
    X_train, X_test_val, y_train, y_test_val = train_test_split(X, y,␣
↪train_size=0.7)
    X_test, X_val, y_test, y_val = train_test_split(X_test_val, y_test_val,␣
↪train_size=0.5)

    clf = tree.DecisionTreeClassifier().fit(X_train, y_train)

    # computing predicted labels with training, validation and testing sets
```

```
    y_pred_val = clf.predict(X_val)
    y_pred_test = clf.predict(X_test)
    y_pred_train = clf.predict(X_train)

    # calculationg the accuracy of the predicted labels
    test_accuracy = accuracy_score(y_test, y_pred_test)
    val_accuracy = accuracy_score(y_val, y_pred_val)
    train_accuracy = accuracy_score(y_train, y_pred_train)

    # calculating and saving the prediction errors
    train_errors.append(1 - train_accuracy)
    val_errors.append(1 - val_accuracy)
    test_loss = log_loss(y_test, y_pred_test)
    test_errors.append(test_loss)

    print("\nDecision Tree:")
    print("Training accuracy of the decision tree is {:.0f} %".
↪format(train_accuracy * 100))
    print("Validation accuracy of the decision tree is {:.0f} %".
↪format(val_accuracy * 100))
    print("Training error of the decision tree is {:.0f} %".format((1 -␣
↪train_accuracy) * 100))
    print("Validation error of the decision tree is {:.0f} %".
↪format((1-val_accuracy) * 100))

    # Visualizing the decision tree model
    feature_names = ["BMI", "Age", "Glucose", "BloodPressure"]
    class_names = ["No diabetes", "Diabetes"]

    plt.subplots(figsize=(17, 8))
    tree.plot_tree(clf, feature_names=feature_names, filled=True, rounded=True,␣
↪class_names=class_names)
    plt.show()

    # computing a confusion matrix to examine the accuracy
    conf_mat = confusion_matrix(y_val, y_pred_val)

    ax = plt.subplot()
    sns.heatmap(conf_mat, annot=True, fmt='g', ax=ax)
    ax.set_xlabel('Predicted labels', fontsize=15)
    ax.set_ylabel('True labels', fontsize=15)
    ax.set_title('Confusion Matrix (Decision Tree)', fontsize=15)
    plt.show()

    # decision tree's loss function
    loss_val = log_loss(y_val, y_pred_val)
    loss.append(loss_val)
```

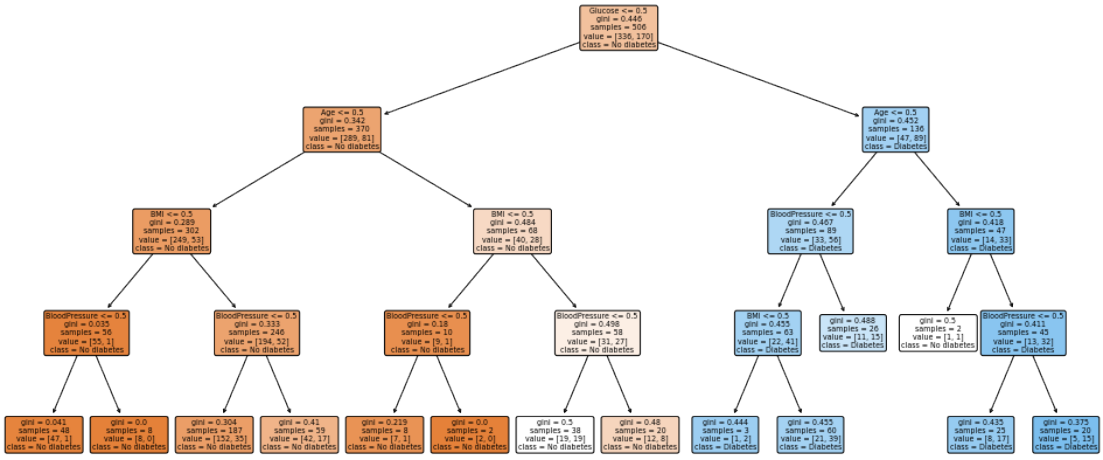
[7]:
```
def main():
    logistic_regression(diabetesData)
    decision_tree(diabetesData)

    height = bar_height(diabetesData['Outcome'])

    # Comparing logistic regression's and decision tree's training and␣
 ↪validation errors with histograms
    ax3 = plt.subplot()
    ax3.bar(np.arange(2) + 0.2, val_errors, 0.4, color='cornflowerblue',␣
 ↪label='validation error')
    ax3.bar(np.arange(2) - 0.2, train_errors, 0.4, color='violet',␣
 ↪label='training error')
    ax3.set_ylim(0, 0.35)
    ax3.set_xticks(np.arange(2))
    ax3.set_xticklabels(['Logistic regression', 'Decision tree'])
    ax3.set_title('Comparison of training and validation errors')
    ax3.legend()
    plt.show()

    # Comparing logistic regression's and decision tree's calculated logistic␣
 ↪loss functions with histograms
    ax1 = plt.subplot()
    ax1.bar(np.arange(2), loss, 0.5, color='palegreen', label='loss function')
    ax1.set_ylim(0, 11)
    ax1.set_xticks(np.arange(2))
    ax1.set_xticklabels(['Logistic regression', 'Decision tree'])
    ax1.set_title('Comparison of the loss functions')
    ax1.legend()
    plt.show()

    # Comparing test errors
    ax5 = plt.subplot()
    ax5.bar(np.arange(2), test_errors, 0.5, color='salmon', label='Test errors')
    ax5.set_ylim(0, 11)
    ax5.set_xticks(np.arange(2))
    ax5.set_xticklabels(['Logistic regression', 'Decision tree'])
    ax5.set_title('Comparison of testing errors')
    ax5.legend()
    plt.show()

    # Printing the testing errors
    print("Testing error of the logistic regression is {:.2f}".
 ↪format((test_errors[0])))

main()
```

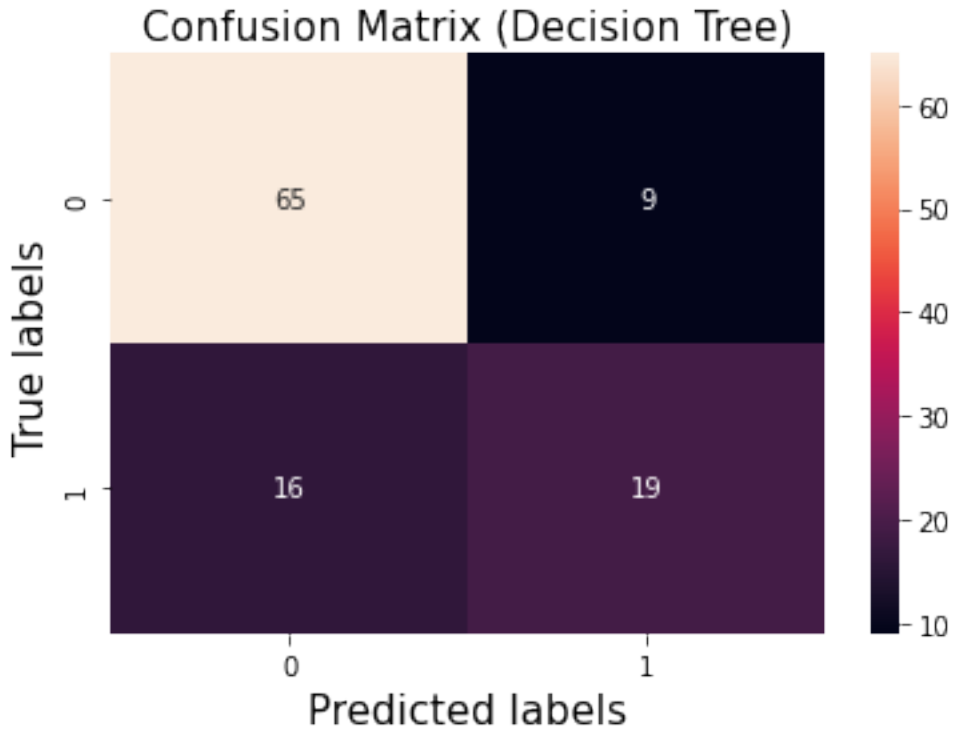
```
Logistic Regression:
Training accuracy of the logistic regression is 76 %
Validation accuracy of the logistic regression is 81 %
Training error of the logistic regression is 24 %
Validation error of the logistic regression is 19 %
```



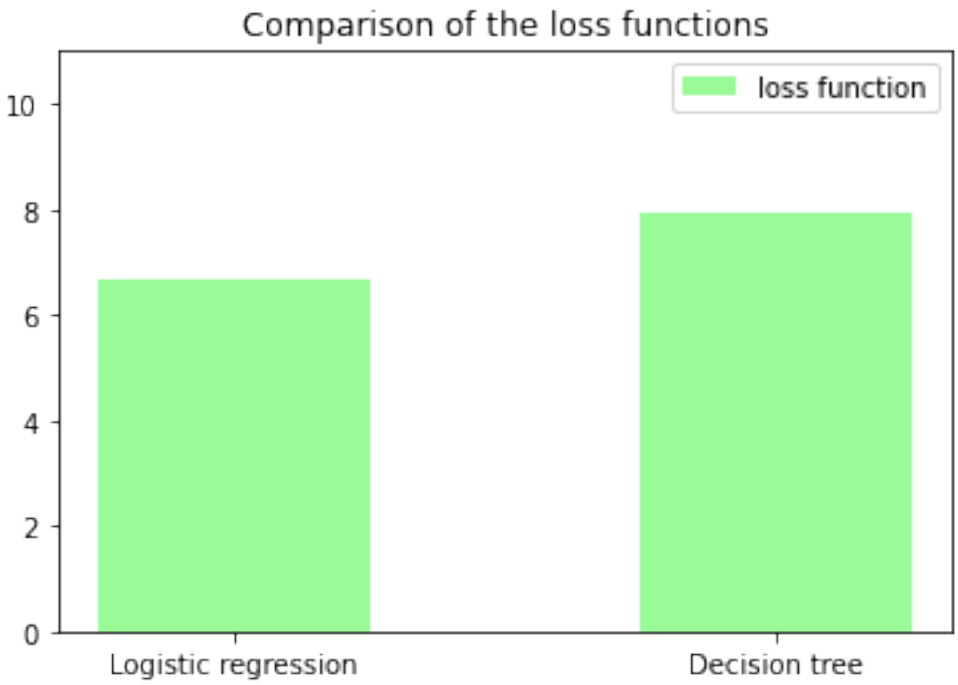


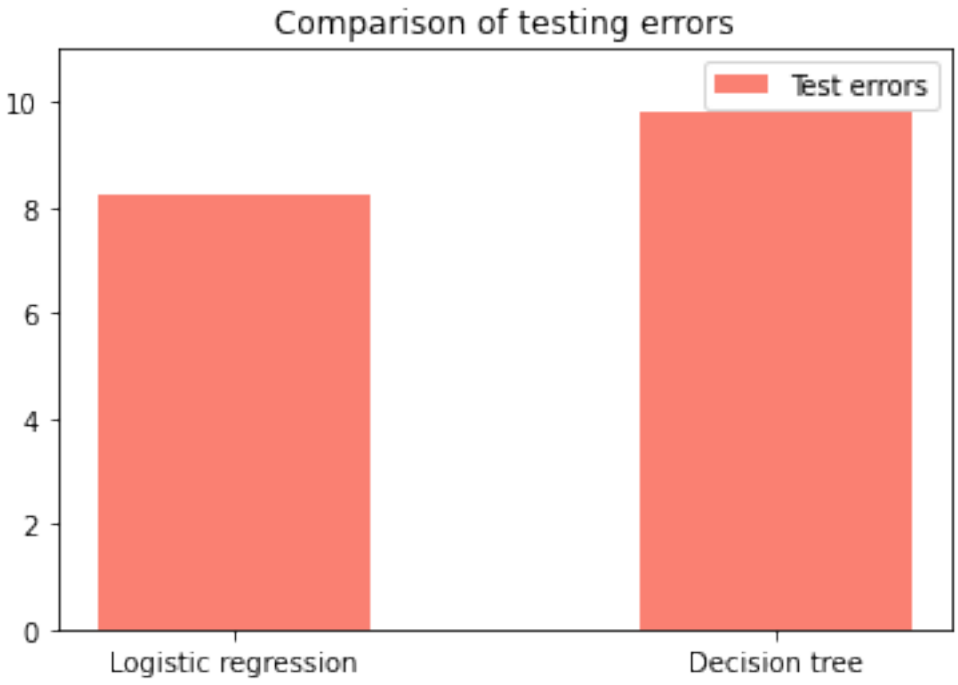
Decision Tree:
Training accuracy of the decision tree is 75 %
Validation accuracy of the decision tree is 77 %
Training error of the decision tree is 25 %
Validation error of the decision tree is 23 %




Confusion Matrix (Decision Tree)
True labels
Predicted labels
0
1
65
9
16
19
60
50
40
30
20
10


Comparison of training and validation errors
validation error
training error
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Logistic regression
Decision tree






```
Testing error of the logistic regression is 8.24
```

[ ]: