

VIETNAMESE - GERMAN UNIVERSITY  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEER



## Lab 3 Report

*Student:* Le Thanh Hai - 10421016

*Student:* Trinh The Hao - 10421017

*Student:* Vu Ngoc Quang - 10421103

## 1 Exercise 2.1 & 2.2 & 2.3

---

```
1 from sklearn.datasets import load_diabetes
2 from sklearn.model_selection import train_test_split
3 import numpy as np
4
5 # load dataset
6 diabetes = load_diabetes()
7
8 # describe dataset
9 print(diabetes.DESCR)
10
11 # divide dataset into features and targets
12 features = diabetes.data
13 targets = diabetes.target
14
15 # shuffle the dataset
16 randNum = np.arange(features.shape[0])
17 np.random.shuffle(randNum)
18 features = features[randNum]
19 targets = targets[randNum]
20
21 # divide dataset into training and testing sets
22 features_train, features_test, targets_train, targets_test = \
    train_test_split(features, targets, test_size=0.2)
```

---

### Explanation:

The diabetes dataset consists of ten baseline variables, such as age, sex, body mass index, average blood pressure, and six blood serum measurements. The target variable is a quantitative measure of disease progression one year after baseline.

The load\_diabetes function loads the dataset into memory as a Bunch object, which is similar to a dictionary. The DESCR attribute of the Bunch object contains a description of the dataset.

The train\_test\_split function from the model\_selection module is used to split the data into training and testing sets. The test\_size parameter is set to 0.2, which means that twenty percent of the data is used for testing and 80 percent is used for training.

## 2 Exercise 2.4 & 2.5

---

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.metrics import mean_squared_error
3
4 # create model
```

```

.. _diabetes_dataset:
Diabetes dataset
-----
Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

: Number of Instances: 442

: Number of Attributes: First 10 columns are numeric predictive values

: Target: Column 11 is a quantitative measure of disease progression one year after baseline

: Attribute Information:
  - age      age in years
  - sex      sex
  - bmi      body mass index
  - bp       average blood pressure
  - s1       tc, total serum cholesterol
  - s2       ldl, low-density lipoproteins
  - s3       hdl, high-density lipoproteins
  - s4       tch, total cholesterol / HDL
  - s5       ltg, possibly log of serum triglycerides level
  - s6       glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times the square root of 'n_samples' (i.e. the sum of squares of each column totals 1).

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics (with discussion), 487-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)

```

Figure 1: Dataset Description

```

5 model = LinearRegression()
6 model.fit(features_train, targets_train)
7 pred = model.predict(features_test)
8
9 # calculate mean squared error
10 print("Error of the Linear Regression: ", mean_squared_error(↵
    targets_test, pred))

```

**Explanation:**

We used the scikit-learn's LinearRegression class to create a linear regression model, fits it to the training data, and then uses the model to make predictions on the testing data. The mean squared error between the predicted values and the actual target values is then calculated using the meansquarederror function from the metrics module.

The LinearRegression class is a simple linear regression model that fits a linear equation to the training data. The fit method is called on the model object to train the model using the training data. Once the model is trained, the predict method is called on the model object to make predictions on the testing data.

The mean squared error is a commonly used metric for evaluating regression models. It measures the average squared difference between the predicted values and the actual target values. A lower mean squared error indicates that the model is better at predicting the target values.

Error of the Linear Regression: 3151.2643307454923

haib1 lab3 main 3.9.0 1.565s

Figure 2: Error of the Linear Regression model

### 3 Exercise 2.6 & 2.7 & 2.8

---

```
1 import matplotlib.pyplot as plt
2 def create_model(features_train, targets_train):
3     # create model
4     model = LinearRegression()
5     model.fit(features_train, targets_train)
6     return model
7
8 def predict(model, features_test):
9     pred = model.predict(features_test)
10    return pred
11
12 # training dataset in each feature
13 plt.figure(figsize=(16, 10)) # set the size of the figure
14 for i in range(0, 10):
15     feature = features[:, i] # get the feature
16
17     # divide dataset into training and testing sets
18     feature_train, feature_test, target_train, target_test = ↵
        train_test_split(feature, targets, test_size=0.2)
19
20     # reshape the data (1D -> 2D) (353,) -> (353,1) to fit the model
21     feature_train = feature_train.reshape(-1, 1)
22     feature_test = feature_test.reshape(-1, 1)
23
24     # create model
25     model = create_model(feature_train, target_train)
26
27     # predict
28     pred = predict(model, feature_test)
29
30     # calculate mean squared error
31     print(f"Error of the Linear Regresson for feature {i+1}: ", ↵
        mean_squared_error(target_test, pred))
32
33     # plot the data
34     plt.subplot(2,5,i+1)
35     plt.title(f"Feature {i+1}: {feature_names[i]}") # set the title of ↵
        the figure
36     plt.scatter(feature_test, target_test, color=colors[i], label='↵
        Data') # plot the data
37     plt.xlabel('Feature') # set the label of the x-axis
38     plt.ylabel('Target') # set the label of the y-axis
```

```

39     plt.plot(feature_test, pred, color='black', linewidth=3) # plot ←
        the line
40     plt.tight_layout()
41
42 plt.show() # show the figure

```

### Explanation:

We created a linear regression model for each of the ten features in the diabetes dataset, trains the model on a subset of the feature data, and then evaluates the model's performance by plotting the predicted targets against the actual targets and calculating the mean squared error.

The createmodel function creates a linear regression model using scikit-learn's LinearRegression class and trains the model on the training data. The predict function uses the trained model to make predictions on the testing data.

```

Error of the Linear Regresson for feature 1: 5604.866564725574
Error of the Linear Regresson for feature 2: 6610.79333138954
Error of the Linear Regresson for feature 3: 3981.4390198229917
Error of the Linear Regresson for feature 4: 5006.221573434579
Error of the Linear Regresson for feature 5: 5478.813415133981
Error of the Linear Regresson for feature 6: 5543.282052989554
Error of the Linear Regresson for feature 7: 5643.004187105904
Error of the Linear Regresson for feature 8: 5586.285729154551
Error of the Linear Regresson for feature 9: 3879.454014753615
Error of the Linear Regresson for feature 10: 4486.184178121869

```

Figure 3: Error of Linear Regression for 10 features

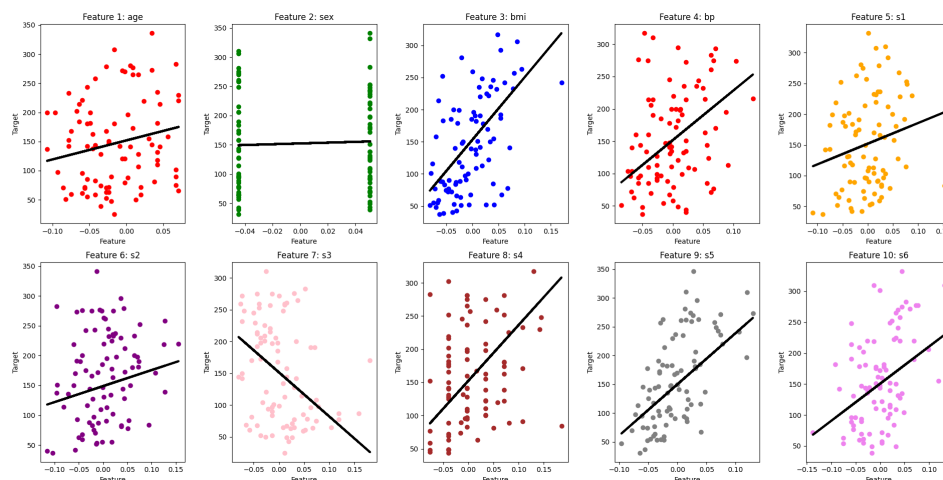


Figure 4: Linear Regression Visualization

## 4 Exercise 2.9 & 2.10

---

```
1 # Add a column of ones to the feature matrix for the bias term
2 X_train = np.hstack((np.ones((feature_train.shape[0], 1)), ↵
    feature_train))
3
4 # Calculate the manual inverse matrix
5 XtX_inv = np.linalg.inv(np.dot(X_train.T, X_train))
6 Xty = np.dot(X_train.T, target_train)
7
8 # Calculate the weights using the manual inverse matrix
9 weights = np.dot(XtX_inv, Xty)
10
11 # Print the weights
12 print("Weights:")
13 print(weights)
14
15 # Predict on the training set
16 y_pred_train = np.dot(X_train, weights)
17
18 # Predict on the testing set
19 X_test = np.hstack((np.ones((feature_test.shape[0], 1)), feature_test)↵
    )
20 y_pred_test = np.dot(X_test, weights)
21
22 # calculate mean squared error
23 print("MSE: " ,mean_squared_error(target_test, y_pred_test))
```

---

### Explanation:

This code provides a way to perform linear regression manually using matrix operations and to compare the results with those obtained using scikit-learn's LinearRegression class. By calculating the mean squared error, the code also provides a metric for evaluating the performance of the manually implemented linear regression model.

## 5 Exercise 2.11 & 2.12

---

```
1 from sklearn.datasets import load_iris
2
3 # load dataset
4 iris = load_iris()
5
6 # describe dataset
```

```
7 print(iris.DESCR)
```

### Explanation:

This code loads the iris dataset from the scikit-learn library and prints a description of the dataset.

```
.. _iris_dataset:
- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
  - Iris-Setosa
  - Iris-Versicolour
  - Iris-Virginica

:Summary Statistics:
=====
      Min   Max   Mean   SD   Class Correlation
=====
sepal length:  4.3   7.9   5.84   0.83   0.7826
sepal width:   2.0   4.4   3.05   0.43  -0.4194
petal length:  1.0   6.9   3.76   1.76   0.9490 (high!)
petal width:   0.1   2.5   1.20   0.76   0.9565 (high!)
=====

:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988
```

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper. Note that it's the same as in R, but not as in the UCI Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

.. topic:: References

- Fisher, R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).
- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis. (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II conceptual clustering system finds 3 classes in the data.
- Many, many more ...

## 6 Exercise 2.13 & 2.14 & 2.15 & 2.16

```
1 from sklearn.tree import DecisionTreeClassifier
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import accuracy_score
4 import numpy as np
5
6 # Filter the data for two specific classes
7 X = iris.data
8 y = iris.target
9
10 # shuffle the dataset
11 randNum = np.arange(X.shape[0])
12 np.random.shuffle(randNum)
13 X = X[randNum]
14 y = y[randNum]
15
16 # divide dataset into training and testing sets with a ratio of 80/20
17 features_train, features_test, targets_train, targets_test = \
    train_test_split(X,y, test_size=0.2)
18
19 # create model
20 clf = DecisionTreeClassifier()
```

```
21 clf.fit(features_train, targets_train) # Train the model using the ↵  
    training set  
22  
23 # predict the class labels for the test set  
24 prediction = clf.predict(features_test)  
25  
26 # calculate the accuracy of the trained model on the testing set  
27 accuracy = accuracy_score(targets_test, prediction)  
28 print(f"Accuracy of the Decision Tree Classifier: {accuracy*100}%")
```

---

**Explanation:**

We used scikit-learn's DecisionTreeClassifier class to create a decision tree model, fits it to the training data, and then uses the model to make predictions on the testing data. The accuracy of the model is then calculated using the accuracyscore function from the metrics module.

The traintestsplit function from the modelselection module is used to split the data into training and testing sets with a test size of 20 percent. The resulting training and testing sets are then used to fit the decision tree model and make predictions on the testing data, respectively.

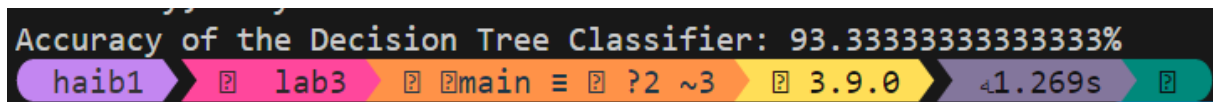


Figure 5: Accuracy of Decision Tree Classifier

## 7 Exercise 2.17 & 2.18 & 2.19

---

```
1 from sklearn.neighbors import KNeighborsClassifier  
2  
3 # create model  
4 n_neighbors = 3  
5 knn = KNeighborsClassifier(n_neighbors=n_neighbors)  
6  
7 # Train the model using the training set  
8 knn.fit(features_train, targets_train)  
9  
10 # predict the class labels for the test set  
11 prediction = knn.predict(features_test)  
12  
13 # calculate the accuracy of the trained model on the testing set  
14 accuracy = accuracy_score(targets_test, prediction)  
15 print(f"Accuracy of the KNN Classifier: {accuracy*100}%")
```

---

**Explanation:**



We used scikit-learn's KNeighborsClassifier class to create a k-nearest neighbors (KNN) classification model, fits it to the training data, and then uses the model to make predictions on the testing data. The accuracy of the model is then calculated using the accuracyscore function from the metrics module.

The KNeighborsClassifier class is a simple classification model that assigns the class label of a test sample to be the majority class label among its k-nearest neighbors in the training set. The nneighbors parameter specifies the number of nearest neighbors to consider when making predictions.

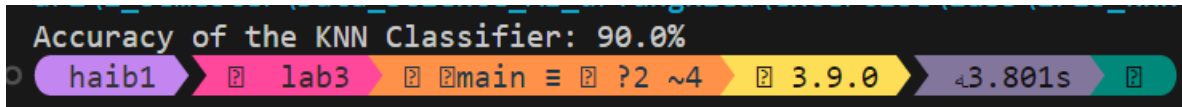


Figure 6: Accuracy of KNN

## 8 Exercise 2.20 & 2.21

```

1  n_neighbors = [2,3,4,5,6,7,8,9,10] # default is 5
2  for n in n_neighbors:
3      knn = create_model(features_train, targets_train,n)
4      prediction = knn.predict(features_test)
5      accuracy = accuracy_score(targets_test, prediction)
6      print(f"Accuracy of the KNN Classifier with n_neighbor = {n}: {←
          accuracy*100}%")
7
8  metrics = ['euclidean', 'manhattan', 'chebyshev', 'minkowski'] # default ←
          is minkowski
9  for m in metrics:
10     knn = create_model(features_train, targets_train,3,m)
11     prediction = knn.predict(features_test)
12     accuracy = accuracy_score(targets_test, prediction)
13     print(f"Accuracy of the KNN Classifier with metric = {m}: {←
          accuracy*100}%")
14
15  weights = ['uniform', 'distance'] # default is uniform
16  for w in weights:
17     knn = create_model(features_train, targets_train,3, 'minkowski',w)
18     prediction = knn.predict(features_test)
19     accuracy = accuracy_score(targets_test, prediction)
20     print(f"Accuracy of the KNN Classifier with weights = {w}: {←
          accuracy*100}%")
21

```

```
22 algorithm = ['auto', 'ball_tree', 'kd_tree', 'brute'] # this ←
    hyperparameters is ignored when p = 1
23 for a in algorithm:
24     knn = create_model(features_train, targets_train, 3, 'minkowski', '←
        uniform', a)
25     prediction = knn.predict(features_test)
26     accuracy = accuracy_score(targets_test, prediction)
27     print(f"Accuracy of the KNN Classifier with algorithm = {a}: {←
        accuracy*100}%")
28
29 leaf_size = [10, 20, 30, 40, 50] # leaf_size should be less than the ←
    number of samples
30 for l in leaf_size:
31     knn = create_model(features_train, targets_train, 3, 'minkowski', '←
        uniform', 'auto', l)
32     prediction = knn.predict(features_test)
33     accuracy = accuracy_score(targets_test, prediction)
34     print(f"Accuracy of the KNN Classifier with leaf_size = {l}: {←
        accuracy*100}%")
35
36 p = [1, 2, 3, 4, 5] # p = 1 is equivalent to using manhattan_distance (l1)←
    , and euclidean_distance (l2) for p = 2
37 for p_value in p:
38     knn = create_model(features_train, targets_train, 3, 'minkowski', '←
        uniform', 'auto', 30, p_value)
39     prediction = knn.predict(features_test)
40     accuracy = accuracy_score(targets_test, prediction)
41     print(f"Accuracy of the KNN Classifier with p = {p_value}: {←
        accuracy*100}%")
```

---

**Explanation:**

We performed hyperparameter tuning for a KNN classifier on the iris dataset, using various values for the nneighbors, metric, weights, algorithm, leafsize, and p hyperparameters. For each combination of hyperparameters, the code trains a KNN model on the training data, makes predictions on the testing data, and evaluates the accuracy of the model using the accuracyscore function.

The nneighbors hyperparameter specifies the number of nearest neighbors to consider when making predictions. The metric hyperparameter specifies the distance metric used to calculate distances between samples. The weights hyperparameter specifies the weight function used in prediction. The algorithm hyperparameter specifies the algorithm used to compute the nearest neighbors. The leafsize hyperparameter specifies the size of the leaf node in the KD-tree. The p hyperparameter specifies the power parameter for the Minkowski distance metric.

```
Accuracy of the KNN Classifier with n_neighbor = 2: 93.33333333333333%
Accuracy of the KNN Classifier with n_neighbor = 3: 96.66666666666667%
Accuracy of the KNN Classifier with n_neighbor = 4: 96.66666666666667%
Accuracy of the KNN Classifier with n_neighbor = 5: 100.0%
Accuracy of the KNN Classifier with n_neighbor = 6: 96.66666666666667%
Accuracy of the KNN Classifier with n_neighbor = 7: 100.0%
Accuracy of the KNN Classifier with n_neighbor = 8: 96.66666666666667%
Accuracy of the KNN Classifier with n_neighbor = 9: 100.0%
Accuracy of the KNN Classifier with n_neighbor = 10: 96.66666666666667%
Accuracy of the KNN Classifier with metric = euclidean: 96.66666666666667%
Accuracy of the KNN Classifier with metric = manhattan: 96.66666666666667%
Accuracy of the KNN Classifier with metric = chebyshev: 96.66666666666667%
Accuracy of the KNN Classifier with metric = minkowski: 96.66666666666667%
Accuracy of the KNN Classifier with weights = uniform: 96.66666666666667%
Accuracy of the KNN Classifier with weights = distance: 96.66666666666667%
Accuracy of the KNN Classifier with algorithm = auto: 96.66666666666667%
Accuracy of the KNN Classifier with algorithm = ball_tree: 96.66666666666667%
Accuracy of the KNN Classifier with algorithm = kd_tree: 96.66666666666667%
Accuracy of the KNN Classifier with algorithm = brute: 96.66666666666667%
Accuracy of the KNN Classifier with leaf_size = 10: 96.66666666666667%
Accuracy of the KNN Classifier with leaf_size = 20: 96.66666666666667%
Accuracy of the KNN Classifier with leaf_size = 30: 96.66666666666667%
Accuracy of the KNN Classifier with leaf_size = 40: 96.66666666666667%
Accuracy of the KNN Classifier with leaf_size = 50: 96.66666666666667%
Accuracy of the KNN Classifier with p = 1: 96.66666666666667%
Accuracy of the KNN Classifier with p = 2: 96.66666666666667%
Accuracy of the KNN Classifier with p = 3: 96.66666666666667%
Accuracy of the KNN Classifier with p = 4: 96.66666666666667%
Accuracy of the KNN Classifier with p = 5: 96.66666666666667%
```

Figure 7: Accuracy of KNN with different hyperparameters

## 9 Exercise 2.22

```
1 # create model
2 clf = GaussianNB()
3 clf.fit(features_train, targets_train) # Train the model using the ←
    training set
4
5 # predict the class labels for the test set
6 prediction = clf.predict(features_test)
7
8 # calculate the accuracy of the trained model on the testing set
9 accuracy = accuracy_score(targets_test, prediction)
10 print(f"Accuracy of the Naive Bayes Classifier: {accuracy*100}%")
```

### Explanation:

We used scikit-learn's GaussianNB class to create a Naive Bayes classification model, fits it to the training data, and then uses the model to make predictions on the testing data. The accuracy of the model is then calculated using the accuracyscore function from the metrics module.

The Naive Bayes classifier is a probabilistic model that uses Bayes' theorem to predict the class label of a test sample based on its features. The GaussianNB class assumes that the distribution of the features is Gaussian, and estimates the parameters of the distribution from the training data.

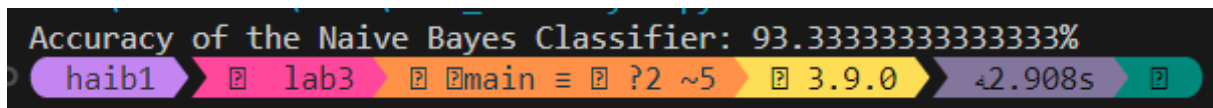


Figure 8: Accuracy of Naive Bayes Classifier

## 10 Exercise 2.23 & 2.24

```
1 # create naivebayes with different parameters
2 priors = [None←
    , [0.1,0.2,0.3,0.4], [0.2,0.2,0.3,0.3], [0.3,0.3,0.2,0.2], [0.4,0.3,0.2,0.1]] ←
3 for p in priors:
4     clf = create_model(features_train, targets_train,p)
5     # predict the class labels for the test set
6     prediction = clf.predict(features_test)
7     # calculate the accuracy of the trained model on the testing set
8     accuracy = accuracy_score(targets_test, prediction)
```

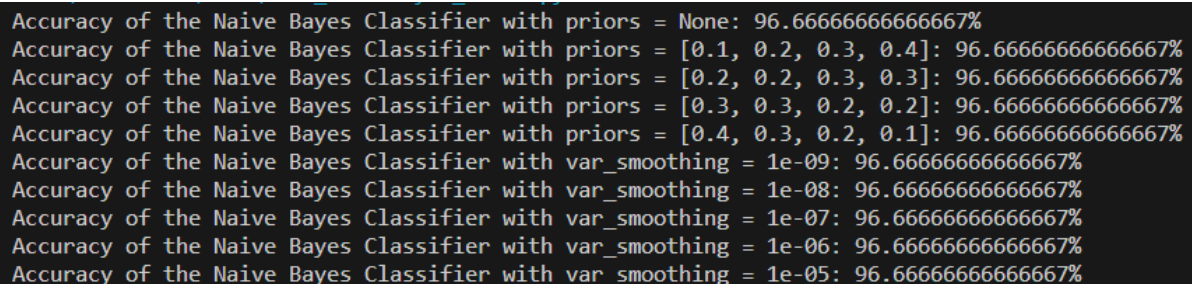
```
9     print(f"Accuracy of the Naive Bayes Classifier with priors = {p}: ←  
        {accuracy*100}%")  
10  
11  var_smoothing = [1e-9,1e-8,1e-7,1e-6,1e-5]  
12  for v in var_smoothing:  
13      clf = create_model(features_train, targets_train, None, v)  
14      # predict the class labels for the test set  
15      prediction = clf.predict(features_test)  
16      # calculate the accuracy of the trained model on the testing set  
17      accuracy = accuracy_score(targets_test, prediction)  
18      print(f"Accuracy of the Naive Bayes Classifier with var_smoothing ←  
            = {v}: {accuracy*100}%")
```

---

**Explanation:**

We performed hyperparameter tuning for a Naive Bayes classifier on the iris dataset, using various values for the priors and varsmoothing hyperparameters. For each combination of hyperparameters, the code trains a Naive Bayes model on the training data, makes predictions on the testing data, and evaluates the accuracy of the model using the accuracyscore function.

The priors hyperparameter specifies the prior probabilities of the classes. The varsmoothing hyperparameter specifies the portion of the largest variance of all features that is added to variances for calculation stability.



```
Accuracy of the Naive Bayes Classifier with priors = None: 96.66666666666667%  
Accuracy of the Naive Bayes Classifier with priors = [0.1, 0.2, 0.3, 0.4]: 96.66666666666667%  
Accuracy of the Naive Bayes Classifier with priors = [0.2, 0.2, 0.3, 0.3]: 96.66666666666667%  
Accuracy of the Naive Bayes Classifier with priors = [0.3, 0.3, 0.2, 0.2]: 96.66666666666667%  
Accuracy of the Naive Bayes Classifier with priors = [0.4, 0.3, 0.2, 0.1]: 96.66666666666667%  
Accuracy of the Naive Bayes Classifier with var_smoothing = 1e-09: 96.66666666666667%  
Accuracy of the Naive Bayes Classifier with var_smoothing = 1e-08: 96.66666666666667%  
Accuracy of the Naive Bayes Classifier with var_smoothing = 1e-07: 96.66666666666667%  
Accuracy of the Naive Bayes Classifier with var_smoothing = 1e-06: 96.66666666666667%  
Accuracy of the Naive Bayes Classifier with var_smoothing = 1e-05: 96.66666666666667%
```

Figure 9: Accuracy of Naive Bayes Classifier with different hyperparameters

## 11 Exercise 2.25

---

```
1  # create model  
2  clf = DecisionTreeClassifier()  
3  clf.fit(features_train, targets_train) # Train the model using the ←  
    training set  
4  
5  # predict the class labels for the test set  
6  prediction = clf.predict(features_test)  
7  
8  # calculate the accuracy of the trained model on the testing set
```

```

9 accuracy = accuracy_score(targets_test, prediction)
10 print(f"Accuracy of the Decision Tree Classifier: {accuracy*100}%")

```

### Explanation:

We used scikit-learn's DecisionTreeClassifier class to create a decision tree classification model, fits it to the training data, and then uses the model to make predictions on the testing data. The accuracy of the model is then calculated using the accuracyscore function from the metrics module.

The decision tree classifier is a non-parametric model that works by recursively splitting the data into subsets based on the values of the input features, with the goal of maximizing the information gain at each split. The DecisionTreeClassifier class is used to create the decision tree model.

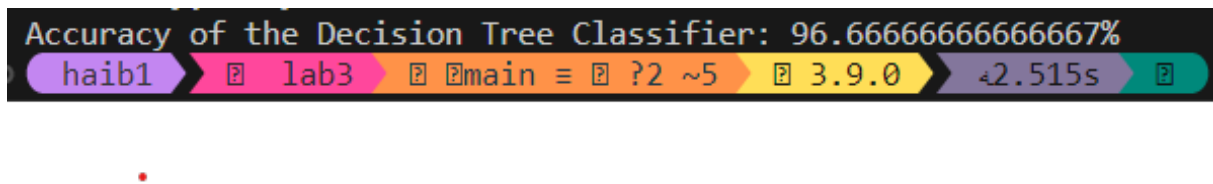


Figure 10: Accuracy of Decision Tree Classifier

## 12 Exercise 2.26 & 2.27

```

1 # create model with differnet hyperparameters
2 criteria = ['gini', 'entropy', 'log_loss']
3 for c in criteria:
4     clf = create_model(features_train, targets_train, criterion=c)
5     # predict the class labels for the test set
6     prediction = clf.predict(features_test)
7     # calculate the accuracy of the trained model on the testing set
8     accuracy = accuracy_score(targets_test, prediction)
9     print(f"Accuracy of the Decision Tree Classifier with criteria = {c}: {accuracy*100}%")
10
11 splitter = ['best', 'random']
12 for s in splitter:
13     clf = create_model(features_train, targets_train, splitter=s)
14     # predict the class labels for the test set
15     prediction = clf.predict(features_test)
16     # calculate the accuracy of the trained model on the testing set
17     accuracy = accuracy_score(targets_test, prediction)
18     print(f"Accuracy of the Decision Tree Classifier with splitter = {s}: {accuracy*100}%")

```

```
19
20 max_depth = [None,1,2,3,4,5,6,7,8,9,10]
21 for m in max_depth:
22     clf = create_model(features_train, targets_train,max_depth=m)
23     # predict the class labels for the test set
24     prediction = clf.predict(features_test)
25     # calculate the accuracy of the trained model on the testing set
26     accuracy = accuracy_score(targets_test, prediction)
27     print(f"Accuracy of the Decision Tree Classifier with max_depth = {m}: {accuracy*100}%")
28
29 min_samples_split = [2,3,4,5,6,7,8,9,10]
30 for m in min_samples_split:
31     clf = create_model(features_train, targets_train,min_samples_split=
        =m)
32     # predict the class labels for the test set
33     prediction = clf.predict(features_test)
34     # calculate the accuracy of the trained model on the testing set
35     accuracy = accuracy_score(targets_test, prediction)
36     print(f"Accuracy of the Decision Tree Classifier with {m}: {accuracy*100}%")
37
38 min_samples_leaf = [1,2,3,4,5,6,7,8,9,10]
39 for m in min_samples_leaf:
40     clf = create_model(features_train, targets_train,min_samples_leaf=
        m)
41     # predict the class labels for the test set
42     prediction = clf.predict(features_test)
43     # calculate the accuracy of the trained model on the testing set
44     accuracy = accuracy_score(targets_test, prediction)
45     print(f"Accuracy of the Decision Tree Classifier with {m}: {accuracy*100}%")
46
47 min_weight_fraction_leaf = [0.0,0.1,0.2,0.3,0.4,0.5]
48 for m in min_weight_fraction_leaf:
49     clf = create_model(features_train, targets_train,
        min_weight_fraction_leaf=m)
50     # predict the class labels for the test set
51     prediction = clf.predict(features_test)
52     # calculate the accuracy of the trained model on the testing set
53     accuracy = accuracy_score(targets_test, prediction)
54     print(f"Accuracy of the Decision Tree Classifier with {m}: {accuracy*100}%")
55 max_features = [None,'sqrt','log2']
56 for m in max_features:
```

```
57     clf = create_model(features_train, targets_train,max_features=m)
58     # predict the class labels for the test set
59     prediction = clf.predict(features_test)
60     # calculate the accuracy of the trained model on the testing set
61     accuracy = accuracy_score(targets_test, prediction)
62     print(f"Accuracy of the Decision Tree Classifier with max_features←
        = {m}: {accuracy*100}%")
63
64 max_leaf_nodes = [2,3,4,5,6]
65 for m in max_leaf_nodes:
66     clf = create_model(features_train, targets_train,max_leaf_nodes=m)
67     # predict the class labels for the test set
68     prediction = clf.predict(features_test)
69     # calculate the accuracy of the trained model on the testing set
70     accuracy = accuracy_score(targets_test, prediction)
71     print(f"Accuracy of the Decision Tree Classifier with ←
        max_leaf_nodes = {m}: {accuracy*100}%")
72
73 min_impurity_decrease = [0.0,0.1,0.2,0.3,0.4,0.5]
74 for m in min_impurity_decrease:
75     clf = create_model(features_train, targets_train,←
        min_impurity_decrease=m)
76     # predict the class labels for the test set
77     prediction = clf.predict(features_test)
78     # calculate the accuracy of the trained model on the testing set
79     accuracy = accuracy_score(targets_test, prediction)
80     print(f"Accuracy of the Decision Tree Classifier with ←
        min_impurity_decrease = {m}: {accuracy*100}%")
81
82 ccp_alpha = [0.0,0.1,0.2,0.3,0.4,0.5]
83 for m in ccp_alpha:
84     clf = create_model(features_train, targets_train,ccp_alpha=m)
85     # predict the class labels for the test set
86     prediction = clf.predict(features_test)
87     # calculate the accuracy of the trained model on the testing set
88     accuracy = accuracy_score(targets_test, prediction)
89     print(f"Accuracy of the Decision Tree Classifier with ccp_alpha = ←
        {m}: {accuracy*100}%")
```

---

**Explanation:**

We performed hyperparameter tuning for a decision tree classifier on the iris dataset, using various values for different hyperparameters. For each combination of hyperparameters, the code trains a decision tree model on the training data, makes predictions on the testing data, and evaluates the accuracy of the model using the accuracyscore function.



We performed a grid search over a range of values for each hyperparameter, and reports the accuracy of the model for each combination of hyperparameters.

```
Accuracy of the Decision Tree Classifier with criteria = gini: 96.6666666666667%
Accuracy of the Decision Tree Classifier with criteria = entropy: 96.6666666666667%
Accuracy of the Decision Tree Classifier with criteria = log_loss: 96.6666666666667%
Accuracy of the Decision Tree Classifier with splitter = random: 96.6666666666667%
Accuracy of the Decision Tree Classifier with max_depth = None: 93.3333333333333%
Accuracy of the Decision Tree Classifier with max_depth = 1: 60.0%
Accuracy of the Decision Tree Classifier with max_depth = 2: 93.3333333333333%
Accuracy of the Decision Tree Classifier with max_depth = 3: 96.6666666666667%
Accuracy of the Decision Tree Classifier with max_depth = 4: 93.3333333333333%
Accuracy of the Decision Tree Classifier with max_depth = 5: 93.3333333333333%
Accuracy of the Decision Tree Classifier with max_depth = 6: 93.3333333333333%
Accuracy of the Decision Tree Classifier with max_depth = 7: 96.6666666666667%
Accuracy of the Decision Tree Classifier with max_depth = 8: 93.3333333333333%
Accuracy of the Decision Tree Classifier with max_depth = 9: 96.6666666666667%
Accuracy of the Decision Tree Classifier with max_depth = 10: 93.3333333333333%
Accuracy of the Decision Tree Classifier with min_samples_split = 2: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_split = 3: 93.3333333333333%
Accuracy of the Decision Tree Classifier with min_samples_split = 4: 93.3333333333333%
Accuracy of the Decision Tree Classifier with min_samples_split = 4: 93.3333333333333%
Accuracy of the Decision Tree Classifier with min_samples_split = 5: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_split = 6: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_split = 7: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_split = 8: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_split = 9: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_split = 10: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_split = 10: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_leaf = 1: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_leaf = 2: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_leaf = 3: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_leaf = 4: 96.6666666666667%
Accuracy of the Decision Tree Classifier with min_samples_leaf = 5: 93.3333333333333%
Accuracy of the Decision Tree Classifier with min_samples_leaf = 6: 93.3333333333333%
Accuracy of the Decision Tree Classifier with min_samples_leaf = 7: 93.3333333333333%
Accuracy of the Decision Tree Classifier with min_samples_leaf = 8: 93.3333333333333%
Accuracy of the Decision Tree Classifier with min_samples_leaf = 9: 93.3333333333333%
Accuracy of the Decision Tree Classifier with min_samples_leaf = 10: 93.3333333333333%
```

```
Accuracy of the Decision Tree Classifier with min_weight_fraction_leaf = 0.0: 96.66666666666667%
Accuracy of the Decision Tree Classifier with min_weight_fraction_leaf = 0.1: 93.33333333333333%
Accuracy of the Decision Tree Classifier with min_weight_fraction_leaf = 0.2: 93.33333333333333%
Accuracy of the Decision Tree Classifier with min_weight_fraction_leaf = 0.3: 93.33333333333333%
Accuracy of the Decision Tree Classifier with min_weight_fraction_leaf = 0.4: 66.66666666666666%
Accuracy of the Decision Tree Classifier with min_weight_fraction_leaf = 0.5: 66.66666666666666%
Accuracy of the Decision Tree Classifier with max_features = None: 93.33333333333333%
Accuracy of the Decision Tree Classifier with max_features = sqrt: 93.33333333333333%
Accuracy of the Decision Tree Classifier with max_features = log2: 96.66666666666667%
Accuracy of the Decision Tree Classifier with max_leaf_nodes = 2: 60.0%
Accuracy of the Decision Tree Classifier with max_leaf_nodes = 3: 93.33333333333333%
Accuracy of the Decision Tree Classifier with max_leaf_nodes = 4: 96.66666666666667%
Accuracy of the Decision Tree Classifier with max_leaf_nodes = 5: 96.66666666666667%
Accuracy of the Decision Tree Classifier with max_leaf_nodes = 6: 96.66666666666667%
Accuracy of the Decision Tree Classifier with min_impurity_decrease = 0.0: 93.33333333333333%
Accuracy of the Decision Tree Classifier with min_impurity_decrease = 0.1: 93.33333333333333%
Accuracy of the Decision Tree Classifier with min_impurity_decrease = 0.2: 93.33333333333333%
Accuracy of the Decision Tree Classifier with min_impurity_decrease = 0.3: 60.0%
Accuracy of the Decision Tree Classifier with min_impurity_decrease = 0.4: 26.66666666666668%
Accuracy of the Decision Tree Classifier with min_impurity_decrease = 0.5: 26.66666666666668%
Accuracy of the Decision Tree Classifier with ccp_alpha = 0.0: 96.66666666666667%
Accuracy of the Decision Tree Classifier with ccp_alpha = 0.1: 93.33333333333333%
Accuracy of the Decision Tree Classifier with ccp_alpha = 0.2: 93.33333333333333%
Accuracy of the Decision Tree Classifier with ccp_alpha = 0.3: 60.0%
Accuracy of the Decision Tree Classifier with ccp_alpha = 0.4: 26.66666666666668%
Accuracy of the Decision Tree Classifier with ccp_alpha = 0.5: 26.66666666666668%
```