Opdrachten Dag 5

Deze opdracht gaat over het classificeren van verschillende soorten pinguïns op basis van verschillende kenmerken zoals gewicht, de lengte van de flippers en de lengte en diepte van de snavel. In de opdracht worden de verschillende stappen door lopen vanaf data acquisitie, data exploratie en data preparatie tot het maken van een classificatie model en het evalueren daarvan.

Stap 1: Importeer de benodigde bibliotheken:
Tip:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

Stap 2: Lees de penguïns dataset in in een dataframe. Deze is opgenomen in de datasets van seaborn.
Tip: **df = sns.load_dataset('penguins')**

Stap 3: Exploreer de dataframe. Welke kolommen zijn er? Hoeveel verschillende species zijn er? En hoeveel rijen per specie?
Tip: **df['species'].value_counts()**

Step 4: Maak, om meteen een indruk te krijgen van de dataset, een matrix van scatter plots met de seaborn functie pairplot.
Tip: **sns.pairplot(df, hue='species')**

Stap 5: Verwijder eventuele incomplete observaties.
Tip: **df.dropna(inplace = True)**

Stap 6: In de dataset zijn verschillend categoriale kenmerken opgenomen. Zet deze om naar categoriën. Ook species kan in een categorie worden omgezet.
Tip: **categorical_features = ['species','island','sex']**
en **df[categorical_features] = df[categorical_features].astype('category')**

Stap 7: Onderzoek de verdeling van de observaties over de verschillende categorieën van deze van features met behulp van een bar plot.
Tip: **df['island'].value_counts().plot(kind = 'bar')**

Stap 8: Voor het gebruik van deze categoriale features in het nodig dat deze worden gecodeerd met behulp van one-hot encoded dummy variabelen. Voeg deze toe aan de dataframe.
Tip:
enc = OneHotEncoder()
enc.fit(df[categorical_features])
onehotlabels = enc.transform(df[categorical_features]).toarray()
encoded_columns = list(enc.get_feature_names_out(['island', 'sex']))
df[encoded_columns] = onehotlabels

Stap 9: Identificeer de feature en de target kolommen. Selecteer eventueel de kolommen die je zou willen gebruiken bij het maken van het model.
Tip:

```
feature_names = [
   'bill_length_mm',
   'bill_depth_mm',
   'flipper_length_mm',
   'body_mass_g',
   'sex_Female', 'sex_Male',
   'island_Biscoe', 'island_Dream', 'island_Torgersen']
target_name = 'species'
```

Stap 10: Splits de data in een training set en een test set.
Tip:

```
from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(df, test_size=0.3)
```

Stap 11: Verdeel de features en de target in verschillende datasets.
Tip:

```
df_train_features = df_train[feature_names]
df_train_target = df_train[target_name]
df_test_features = df_test[feature_names]
df_test_target = df_test[target_name]
```

Stap 12: Schaal de numerieke kolommen naar waarden tussen 0 en 1.
Tip:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df_train_features)
train_features_scaled = scaler.transform(df_train_features)
test_features_scaled = scaler.transform(df_test_features)
```

Stap 13: Je bent nu klaar om aan de daadwerkelijke classificatie te beginnen. Er zullen verschillende algoritmes worden gebruikt. Begin met een Naïve Bayes Classificatie.
Tip:

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(df_train_features, df_train_target)
```

Step 14: Gebruik deze classifier op de test set en bestudeer hoeveel van deze voorspellingen overeen met de bekende target waarden in de dataset. Sla deze resultaten op in de test dataframe.
Tip:

```
predicted = classifier.predict(df_test_features)
correct = predicted == df_test_target
correct.value_counts()
```

df_test['NB_predicted'] = predicted
df_test['NB_correct'] = correct

Stap 15: Bepaal de accuracy van het model. Let op wat er gebeurt als alles voorspelling correct zijn.
Tip: **accuracy = sum(correct) / len(df_test_target)**

Herhaal de stappen 13 t/m 15 nu ook voor een aantal andere classificatie algoritmes

Stap 16: Gebruik een k Nearest Neighbor algoritme met k = 3. Gebruik hiervoor de geschaalde features en vergelijk dit met de uitkomst als de niet geschaalde features worden gebruikt.
Tip:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 3)
classifier.fit(df_train_features_scaled, df_train_target)
predicted = classifier.predict(df_test_features_scaled)
correct = predicted == df_test_target
correct.value_counts()
df_test['3NN_predicted'] = predicted
df_test['3NN_correct'] = correct
accuracy = sum(correct) / len(df_test_target)
```

Stap 17: Gebruik een k nearest neigbor algoritme met k = 5.
Tip:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 3)
```

Stap 18: En een decision tree.
Tip:

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(max_depth=4)
```

Stap 19: Van een decision tree classifier kan de tree worden getekend.
Tip:

```
from sklearn.tree import plot_tree
   plot_tree(classifier,
   feature_names = feature_names,
   class_names = target_name,
   filled = True,
   rounded = True,
   impurity = False)
```

Stap 20: Een random forest.
Tip:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier()
```

Stap 21: Een support vector machine. Er kunnen verschillende kernel worden gebruikt zoals linear, poly, rbf of sigmoid.
Tip:

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear')
```

Stap 22: Een logistic regression classificatie.
Tip:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0, max_iter=10000)
```

Stap 23: En tenslotte een classificatie met een neural network. Een multi-layer perceptron.
Tip:

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(
  solver='lbfgs',
  activation='relu',    #'logistic',
  alpha=1e-5,
  hidden_layer_sizes=(5, 5),
  random_state=1,
  max_iter=10000)
```

Stap 24: Om de verschillende classificatie modellen te beoordelen kunnen verschillende performance metrics worden bepaald. Naast de accuracy zijn dit de precision, de recall en de f-score. Deze zijn allen gebaseerd op de confusion matrix. Bepaal van de de verschilende classifers deze metrics.
Tip:

```
prediction = '5NN_predicted'
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(df_test_target, df_test[prediction])
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(df_test_target, df_test[prediction])
from sklearn.metrics import precision_recall_fscore_support
precision, recall, f_score, _ = \
  precision_recall_fscore_support(df_test_target, df_test[prediction])
```

★ Stap 25: Met deze dataset kan ook een lineaire regressie worden gedaan om bijvoorbeeld het gewicht van een penguïn te schatten op basis van andere kenmerken.
Tip:

```
feature_names = ['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm',
'sex_Female', 'sex_Male']
target_name = 'body_mass_g'
```

en

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(df_train_features, df_train_target)
```

predicted = regressor.predict(df_test_features)
r2_score = regressor.score(df_test_features, df_test_target)

★ Stap 26: Bij penguins is het bepalen van het geslacht niet eenvoudig en belastend voor de penguin. Is het mogelijk om op basis van andere kenmerken het geslacht te schatten? Hoe nauwkeurig is deze schatting? Vergelijk de performance van verschillende algoritmes.
Tip:
feature_names = ['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']
target_name = 'sex'