assignment-2-a-i-1

March 2, 2024

[8]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[9]:
```
import os
os.listdir()
```

[9]:
```
['.ipynb_checkpoints',
 'AIDS_ClinicalTrial_GroupStudy175.csv',
 'apple_quality.csv',
 'Untitled.ipynb']
```

[10]:
```
# names = ['time',         'trt',         'age,',␣
 ↪'wtkg',         'hemo',         'homo', 'drugs',␣
 ↪'karnof',         'oprior',         'z30',         'zprior',
#         ␣
 ↪'preanti',         'race',         'gender',         'str2',         'strat',␣
 ↪'symptom',         'treat',         'offtrt',         'cd40',         'cd420',
#          'cd80',         'cd820',         'label',]

dataset = pd.read_csv('apple_quality.csv')
dataset.head()
```

[10]:

	A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	\
0	0	-3.970049	-2.512336	5.346330	-1.012009	1.844900	0.329840	
1	1	-1.195217	-2.839257	3.664059	1.588232	0.853286	0.867530	
2	2	-0.292024	-1.351282	-1.738429	-0.342616	2.838636	-0.038033	
3	3	-0.657196	-2.271627	1.324874	-0.097875	3.637970	-3.413761	
4	4	1.364217	-1.296612	-0.384658	-0.553006	3.030874	-1.303849	

	Acidity	Quality
0	-0.491590	1
1	-0.722809	1
2	2.621636	0
3	0.790723	1
4	0.501984	1

[11]:
```
X = dataset.iloc[:, 1:8]  # iloc is used for integer-based indexing
Y = dataset.iloc[:, 8]
```

[12]:
```
print(X)
```

```
           Size    Weight  Sweetness  Crunchiness  Juiciness  Ripeness  \
0    -3.970049 -2.512336   5.346330    -1.012009   1.844900  0.329840
1    -1.195217 -2.839257   3.664059     1.588232   0.853286  0.867530
2    -0.292024 -1.351282  -1.738429    -0.342616   2.838636 -0.038033
3    -0.657196 -2.271627   1.324874    -0.097875   3.637970 -3.413761
4     1.364217 -1.296612  -0.384658    -0.553006   3.030874 -1.303849
…          …       …          …            …          …         …
3995  0.059386 -1.067408  -3.714549     0.473052   1.697986  2.244055
3996 -0.293118  1.949253  -0.204020    -0.640196   0.024523 -1.087900
3997 -2.634515 -2.138247  -2.440461     0.657223   2.199709  4.763859
3998 -4.008004 -1.779337   2.366397    -0.200329   2.161435  0.214488
3999  0.278540 -1.715505   0.121217    -1.154075   1.266677 -0.776571

       Acidity
0    -0.491590
1    -0.722809
2     2.621636
3     0.790723
4     0.501984
…          …
3995  0.137784
3996  1.854235
3997 -1.334611
3998 -2.229720
3999  1.599796

[4000 rows x 7 columns]
```

[13]:
```
print(Y)
```

```
0       1
1       1
2       0
3       1
4       1
       ..
3995    0
3996    1
3997    0
3998    1
3999    1
Name: Quality, Length: 4000, dtype: int64
```

[79]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.18,␣
↪random_state=40)
```

[80]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

[81]:
```
from sklearn.neighbors import KNeighborsClassifier

# param_grid = {'n_neighbors': [2,3, 5, 7, 9, 11]}
# grid_search = GridSearchCV(KNeighborsClassifier(), param_grid)
# grid_search.fit(X_train, y_train)
# best_classifier = grid_search.best_estimator_
classifier = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
classifier.fit(X_train, y_train)
```

[81]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=3)
```

[82]:
```
y_pred = classifier.predict(X_test)
# y_pred = best_classifier.predict(X_test)
```

[83]:
```
print(y_pred)
```

```
[1 0 0 0 1 0 0 1 0 0 0 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 0
 1 0 1 0 0 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 1 1 0 0
 1 1 1 0 0 1 0 0 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 1 1 1 0 1 1 0 1 0 0
 0 1 0 1 0 0 1 0 1 0 1 1 0 0 0 1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0 0 1 1 0 1 1
 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0
 1 0 1 1 1 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 0 0 1 0 0 1
 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 0 1 0 0 1 1 1 0 1 0 0 1 0 1 0 0 0 1
 1 0 0 0 1 1 0 1 0 0 1 0 0 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 0 1 0 0 1 1 0 1
 0 1 0 1 1 1 0 0 1 1 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 0 1 1 1 0
 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 0 1 1 0 0 1 0 1 1 1 1 0 0 0 0 0 1 0
 0 1 1 1 1 0 1 0 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0
 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 1 0 0
 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0 0 1 1 0 1 1 0 0 1 1 1 1
 0 1 0 0 0 1 0 0 0 0 1 1 1 0 1 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 1
 0 0 1 0 1 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 1 1 0 0 1 1 1 0 1
 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 0 1 0 1 0 0 0 1 1 1 1 0 0 1 1 1 0 1 1
 1 0 1 0 0 1 0 0 0 0 1 1 1 0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 1 0 1 0 0
 1 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 0 0
 1 1 0 1 1 1 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 1 0 0 1
 0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 1 1]
```

[84]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
# accuracy = accuracy_score(y_test, y_pred)
# print("Best parameters:", grid_search.best_params_)
# print("Accuracy:", accuracy)
```

```
0.9013888888888889
```

[85]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90       363
           1       0.91      0.89      0.90       357

    accuracy                           0.90       720
   macro avg       0.90      0.90      0.90       720
weighted avg       0.90      0.90      0.90       720

[[333  30]
 [ 41 316]]
```

Prepared by:

Artuz, Jhon Rey
Barbadillo, Carlo
Pielago, Carlo Nico
Ojastro, Von Carlo
Eltahir, Mohammed

Question 1:

List out 10 datasets which are suitable for classification and its respective link to download

#S.No	Name of the data set	Link to the dataset
1	Mobile Price Classification	https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification
2	Apple Quality	https://www.kaggle.com/datasets/nelgiriyewithana/apple-quality
3	Climate Change Indicators	https://www.kaggle.com/datasets/tarunrm09/climate-change-indicators
4	Global Covid Trend	https://www.kaggle.com/datasets/nelgiriyewithana/global-covid-trend
5	Liver Disorders	https://www.kaggle.com/datasets/fatemehmehrparvar/liver-disorders
6	Heart Attack Analysis	https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset
7	Titanic Dataset	https://www.kaggle.com/datasets/yasserh/titanic-dataset
8	Breast Cancer	https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset
9	Wine Quality	https://www.kaggle.com/datasets/rajyellow46/wine-quality
10	World Population	https://www.kaggle.com/datasets/whenamancodes/world-population-live-dataset

Question 2:

b. Ideal scenario of using KNN or when to use KNN?

KNN is ideal in situations where the data points are well-defined, involving small to medium-sized datasets, non-linear relationships, noisy data, and imbalanced classes.

d. What other algorithms are better than KNN and why?

Naive Bayes:
Naive Bayes is an eager learning classifier and it is much faster than K-NN. Thus, it could be used for prediction in real time. Typically, email spam filtering uses Naive Bayes classifier.

Decision Tree:
The Decision Tree algorithm is an easier algorithm when compared to KNN and it is more accurate also.

Decision trees and random forests can handle large datasets more efficiently than KNN, as they do not require computing distances between all data points.

Decision Tree is used to partition the data to find accurate result but in KNN it used to find similar values from the data.

Support Vector Machines (SVM):
Support Vector Machines (SVM) is more oriented on global data structures and performs well in high-dimensional environments.

SVM performs better on datasets where the number of features exceeds the number of observations, but KNN performs well with small numbers of input variables.

e. KNN is better than what other algorithms and why?

Linear regression:
In situations with a strong non-linear relationship, KNN regression outperforms linear regression. This is because KNN can adapt to complex relationships, providing more accurate predictions.

Support Vector Machines (SVM):
It depends on the data size for example KKN is better than SVM when the data is small and well defined. KNN is easy to implement and computationally less expensive, while SVM takes more time for training and testing.

- Non-linear Relationships: KNN naturally handles non-linear relationships between features and the target variable without requiring explicit feature engineering or transformation.

- Lazy Learning: KNN is a lazy learning algorithm, meaning it doesn't make any assumptions about the underlying data distribution and can adapt quickly to changes in the data.

- Robustness to Noise: KNN can be robust to noisy data, as outliers tend to have less influence on the final predictions when using a larger value of 'k'.