1. Apply a kNN classifier to the iris.csv dataset (which is given to you). Use the file “knn.ipynb” to complete missing parts of the code.

In [126…
```
import numpy as np

# makes printing more human-friendly
np.set_printoptions(precision=3,suppress=True)
```

In [127…
```
# Load the data
colab=False
if colab:
    from google.colab import drive
    drive.mount('/content/drive')
    with open('/content/drive/539/data/iris.csv', 'r') as f:
      data = np.genfromtxt(f,delimiter=',')
else:
    with open('iris.csv', 'r') as f:
        data = np.genfromtxt(f,delimiter=',')

X = data[:, :-1]
y = data[:, -1]
print('num_samples, num_features', X.shape)
print('labels', np.unique(y))
```

```
num_samples, num_features (150, 4)
labels [1. 2. 3.]
```

a) Perform stratified data partition at a 70/15/15 ratio to yield a training/validation/testing partition: $X_{\text{train}}, y_{\text{train}}$ (label), $X_{\text{val}}, y_{\text{val}}$, and X_{test}, and y_{test} using scikit-learn's package `train test split()`.

In [128…
```
# Exercise 1
# 1a) Perform stratified data partition at a 70/30 ratio to yield Xtrain, ytrain (label), Xtest, and test.
from sklearn.model_selection import train_test_split

# X_temp = X_val + X_test
X_train, X_temp, y_train, y_temp = train_test_split(X, y, train_size=0.7, random_state=123)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, train_size=0.5, random_state=123)

print('train: ', X_train.shape)
print('val:   ', X_val.shape)
print('test:  ', X_test.shape)
```

```
train:  (105, 4)
val:    (22, 4)
test:   (23, 4)
```

(b) Let the number of neighbors be either 1, 3, 5, 10, 20, 30, 40, 50, 75, or 100. For each number of neighbors, train a kNN model using scikit-learn package `NeighborsNeighbors`. Evaluate the correct classification rate of each model on the validation set.

In [129…
```
# 1b) # of neighbors vary 1-9, knn model for each
from sklearn.neighbors import KNeighborsClassifier

models = {}
for i in [1,3,5,10,20,30,40,50,75,100]:
  print(f"Training kNN with {i} neighbors")
  model = KNeighborsClassifier(n_neighbors = i)
  model.fit(X_train, y_train)
  models[i] = model
```

```
Training kNN with 1 neighbors
Training kNN with 3 neighbors
Training kNN with 5 neighbors
Training kNN with 10 neighbors
Training kNN with 20 neighbors
Training kNN with 30 neighbors
Training kNN with 40 neighbors
Training kNN with 50 neighbors
Training kNN with 75 neighbors
Training kNN with 100 neighbors
```

In [130…
```
# 1b) accuracy
from sklearn.metrics import accuracy_score

for i in models:
  y_predict = models[i].predict(X_test)  # Predict class using models[i]
  # print(f'Predicted labels for {i} neighbors: {y_predict}')
```

```python
    # print(f'True labels: {y_test}')
    acc = accuracy_score(y_test, y_predict)  # Compute accuracy of the predictions
    print(f'Classification Rate using {i} neighbors: {acc*100:.2f}%')
```

```
Classification Rate using 1 neighbors: 86.96%
Classification Rate using 3 neighbors: 95.65%
Classification Rate using 5 neighbors: 100.00%
Classification Rate using 10 neighbors: 95.65%
Classification Rate using 20 neighbors: 86.96%
Classification Rate using 30 neighbors: 86.96%
Classification Rate using 40 neighbors: 78.26%
Classification Rate using 50 neighbors: 78.26%
Classification Rate using 75 neighbors: 21.74%
Classification Rate using 100 neighbors: 21.74%
```

(c) What model parameter (# of neighbors) yields the highest classification rate in part (b)? Which one do you choose for the final (optimal) model?

According to the result above, 5 neighbors yields the highset accuracy.

In [131…

```python
# 1c) Best number of neighbors
nneigs = 5
```

(d) Train a new kNN model with the number of neighbors selected in part (c). Use both training and validation data to fit the new model. Apply the model to the test set X_{test} and compute the corresponding classification rate and the confusion matrix.

In [132…

```python
# 1d) Train and evaluate final model on X_train and X_val
from sklearn.metrics import confusion_matrix

X_trainval = np.concatenate((X_train, X_val), axis=0)
y_trainval = np.concatenate((y_train, y_val), axis=0)   # Compose trainval dataset

model = models[nneigs].fit(X_trainval, y_trainval)      # Train kNN model on X_trainval, y_trainval with nneigs n
y_predict = model.predict(X_test)                       # Predict classes of X_test

acc = accuracy_score(y_test, y_predict)                 # Evaluate accuracy of predictions
cm = confusion_matrix(y_test, y_predict)                # Compute confusion matrix of predictions

print(f'\nClassification Rate of {nneigs} neighbors: {acc*100:.2f}%')
print(f'Confusion Matrix of {nneigs} neighbors:')
print(cm)
```

```
Classification Rate of 5 neighbors: 95.65%
Confusion Matrix of 5 neighbors:
[[8 0 0]
 [0 5 0]
 [0 1 9]]
```

2. Use the starter code in “knn.ipynb” to re-implement the kNN classifier. Both `fit()` and `predict()` need to be updated. After reimplementing the classifier, train a model similar to that of (1d) and apply it to the test set. Confirm that you obtain the same results as the sklearn version.

In [133…

```python
# Exercise 2

from scipy import stats

class myKNeighborsClassifier:
    def __init__(self, n_neighbors):
        self.n_neighbors = n_neighbors

    def fit(self, X, y):
        # No training necessary. Just store the training dataset
        self.X_train = X
        self.y_train = y
        return self

    def predict(self, X):
        n = X.shape[0]
        y = np.zeros(n)
        for i in range(n):
            d = np.linalg.norm(self.X_train - X[i], axis=1)   # Compute distances between X[i] and all self.X_tra
            neig_idx = np.argsort(d)[:self.n_neighbors]       # Find closest neigbors using np.argsort
            neig_y = self.y_train[neig_idx]                   # Collect labels of closest neighbors
            y[i] = stats.mode(neig_y, axis=None)[0]           # Find most likely label using stats.mode
        return y
```

```
myknn = myKNeighborsClassifier(n_neighbors=5)
myknn.fit(X_trainval, y_trainval)
# myknn.fit(X_train, y_train)
# myknn.fit(X_val, y_val)
mypredict = myknn.predict(X_test)

print(f"Predicted labels: {mypredict}")
print(f"Real labels:      {y_test}")

accuracy = accuracy_score(y_test, mypredict)
print(f'\nClassification Rate of {nneigs} neighbors: {accuracy*100:.2f}%')
```

```
Predicted labels: [1. 1. 3. 3. 2. 3. 1. 3. 3. 3. 2. 1. 2. 1. 2. 2. 1. 3. 3. 1. 1. 3. 2.]
Real labels:      [1. 1. 3. 3. 2. 3. 1. 3. 3. 3. 2. 1. 2. 1. 2. 3. 1. 3. 3. 1. 1. 3. 2.]

Classification Rate of 5 neighbors: 95.65%
```

3. Perform decision tree classification on the dataset `winequality-red.csv`. Use the file "DecisionTreeStarter.ipynb". Print the unique class labels. Use 80/20 stratified data partitioning. Provide a figure of the resulting decision tree, the classification accuracy rate, and the confusion matrix when tested with the testing dataset.

In [ ]:
```
import numpy as np

# makes printing more human-friendly
np.set_printoptions(precision=3, suppress=True)
```

In [ ]:
```
# Load the data
colab=False
if colab:
    from google.colab import drive
    drive.mount('/content/drive')
    with open('/content/drive/539/data/winequality-red.csv', 'r') as f:
      data = np.genfromtxt(f,delimiter=',')
else:
    with open('winequality-red.csv', 'r') as f:
        data = np.genfromtxt(f,delimiter=',')

X = data[1:,:-1]
y = data[1:,-1]-3
print('num_samples, num_features', X.shape)
print('labels', np.unique(y))
```

```
num_samples, num_features (1599, 11)
labels [0. 1. 2. 3. 4. 5.]
```

In [ ]:
```
# Partition the data into Training and Testing (80:20 split)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=123)

print(f'Size of training dataset: {X_train.shape}')
print(f'Size of testing dataset:  {X_test.shape}')
```

```
Size of training dataset: (1279, 11)
Size of testing dataset:  (320, 11)
```

In [ ]:
```
# Train the classification tree.
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion="gini", max_depth=3, random_state=123)
model.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=123)
```

In [ ]:
```
# Test the trained decision tree
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

y_hat = model.predict(X_test)
acc = accuracy_score(y_test, y_hat)
cm = confusion_matrix(y_test, y_hat)

print("Accuracy: " + str(format(acc*100,'.2f')) + '%')
```

```python
print("Confusion Matrix: ")
print(cm)
```

```
Accuracy: 53.44%
Confusion Matrix:
[[ 0  0  1  0  0  0]
 [ 0  0  7  6  0  0]
 [ 0  0 85 46  0  0]
 [ 0  0 48 75  8  0]
 [ 0  0  1 27 11  0]
 [ 0  0  1  2  2  0]]
```

In [ ]:

```python
# Plot a graph of the first trained classification tree.
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

plot_tree(model)
plt.show()
```