Upload files in Google Colab

If you are running this Jupyter Notebook on Google Colab, run this cell to upload the data files (train_inputs.csv, train_labels.csv, test_inputs.csv, test_labels.csv) in the Colab virtual machine. You will be prompted to select files that you would like to upload.

If you are running this Jupyter Notebook on your computer, you do not need to run this cell.

In [ ]:
```
from google.colab import files
uploaded = files.upload()
%ls
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving test_inputs.csv to test_inputs (2).csv
Saving test_labels.csv to test_labels (2).csv
Saving train_inputs.csv to train_inputs (2).csv
Saving train_labels.csv to train_labels (2).csv
 sample_data/          'test_labels (2).csv'   'train_labels (1).csv'
'test_inputs (1).csv'   test_labels.csv        'train_labels (2).csv'
'test_inputs (2).csv'  'train_inputs (1).csv'   train_labels.csv
 test_inputs.csv       'train_inputs (2).csv'
'test_labels (1).csv'   train_inputs.csv
```

Import libraries

Do not use any other Python library.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Function: load_knn_data

This function loads the data for KNN from a local drive into RAM

Outputs:

- **train_inputs**: numpy array of N training data points x M features
- **train_labels**: numpy array of N training labels
- **test_inputs**: numpy array of N' test data points x M features
- **test_labels**: numpy array of N' test labels

In [ ]:
```
def load_knn_data():
  test_inputs = np.genfromtxt('test_inputs.csv', delimiter=',')
  test_labels = np.genfromtxt('test_labels.csv', delimiter=',')
  train_inputs = np.genfromtxt('train_inputs.csv', delimiter=',')
  train_labels = np.genfromtxt('train_labels.csv', delimiter=',')
  return train_inputs, train_labels, test_inputs, test_labels
```

Function: predict_knn

This function implements the KNN classifier to predict the label of a data point. Measure distances with the Euclidean norm (L2 norm). When there is a tie between two (or more) labels, break the tie by choosing any label.

Inputs:

- **x**: input data point for which we want to predict the label (numpy array of M features)
- **inputs**: matrix of data points in which neighbours will be found (numpy array of N data points x M features)
- **labels**: vector of labels associated with the data points (numpy array of N labels)
- **k_neighbours**: # of nearest neighbours that will be used

Output:

- **predicted_label**: predicted label (integer)

In [ ]:
```
def predict_knn(x, inputs, labels, k_neighbours):

  x=np.array(x)
  inputs=np.array(inputs)
  labels = np.array(labels, dtype=int)
```

```
  if k_neighbours == len(labels):
    predicted_label= np.bincount(labels).argmax()
  else:
    distance = np.sqrt(np.sum(((inputs-x)**2), axis = 1))
    idx = np.argpartition(distance, k_neighbours)

    minDistLabels = labels[idx[:k_neighbours]]
    predicted_label = np.bincount(minDistLabels).argmax()

  return predicted_label
```

Function: eval_knn

Function that evaluates the accuracy of the KNN classifier on a dataset. The dataset to be evaluated consists of (inputs, labels). The dataset used to find nearest neighbours consists of (train_inputs, train_labels).

Inputs:

- **inputs**: matrix of input data points to be evaluated (numpy array of N data points x M features)
- **labels**: vector of target labels for the inputs (numpy array of N labels)
- **train_inputs**: matrix of input data points in which neighbours will be found (numpy array of N' data points x M features)
- **train_labels**: vector of labels for the training inputs (numpy array of N' labels)
- **k_neighbours**: # of nearest neighbours to be used (integer)

Outputs:

- **accuracy**: percentage of correctly labeled data points (float)

In [ ]:
```
def eval_knn(inputs, labels, train_inputs, train_labels, k_neighbours):

  labels = (np.array(labels)).astype(int)
  pred_labels = np.zeros(len(inputs))

  for i in range(len(inputs)):
    pred_labels [i] = predict_knn(inputs[i], train_inputs, train_labels, k_neighbours)
  compare = np.sum(pred_labels == labels)
  accuracy = (compare/len(labels))*100
```

```
  return accuracy
```

Function: cross_validation_knn

This function performs k-fold cross validation to determine the best number of neighbours for KNN.

Inputs:

- **k_folds**: # of folds in cross-validation (integer)
- **hyperparameters**: list of hyperparameters where each hyperparameter is a different # of neighbours (list of integers)
- **inputs**: matrix of data points to be used when searching for neighbours (numpy array of N data points by M features)
- **labels**: vector of labels associated with the inputs (numpy array of N labels)

Outputs:

- **best_hyperparam**: best # of neighbours for KNN (integer)
- **best_accuracy**: accuracy achieved with best_hyperparam (float)
- **accuracies**: vector of accuracies for the corresponding hyperparameters (numpy array of floats)

In [ ]:

```
def cross_validation_knn(k_folds, hyperparameters, inputs, labels):

  inputs = np.array(inputs)
  rows, columns = inputs.shape
  step = rows//k_folds

  accuracies = np.zeros(len(hyperparameters))

  for i in range(len(hyperparameters)):
    accuracy = np.zeros(k_folds)
    for j in range(1,k_folds+1):
      val_inputs = inputs[step*(j-1):step*(j)]
      val_labels = labels[step*(j-1):step*(j)]
      train_inputs = np.concatenate((inputs[:step*(j-1)], inputs[step*(j):]), axis=0)
      train_labels = np.concatenate((labels[:step*(j-1)], labels[step*(j):]), axis=0)


      accuracy [j-1] = eval_knn(val_inputs, val_labels, train_inputs, train_labels, hyperparameters[i])
```

```
    accuracies[i] = sum(accuracy) / len(accuracy)

  best_hyperparam = hyperparameters[accuracies.argmax()]
  best_accuracy = max(accuracies)
  accuracies = np.array(accuracies)

  return best_hyperparam, best_accuracy, accuracies
```

Function: plot_knn_accuracies

Function that plots the KNN accuracies for different # of neighbours (hyperparameters) based on cross validation

Inputs:

- **accuracies**: vector of accuracies for the corresponding hyperparameters (numpy array of floats)
- **hyperparams**: list of hyperparameters where each hyperparameter is a different # of neighbours (list of integers)

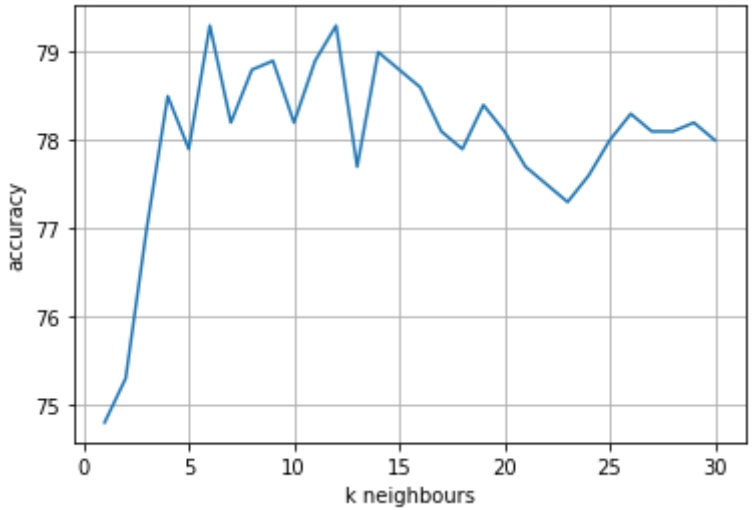
In [ ]:
```
def plot_knn_accuracies(accuracies,hyperparams):
  plt.plot(hyperparams,accuracies)
  plt.ylabel('accuracy')
  plt.xlabel('k neighbours')

  plt.grid() # added for better visualization

  plt.show()
```

Main KNN code

Load data. Use k-fold cross validation to find the best # of neighbours for KNN. Plot accuracies for different # of neighbours. Test KNN with the best # of neighbours.

In [ ]:
```
# load data
train_inputs, train_labels, test_inputs, test_labels = load_knn_data()

# number of neighbours to be evaluated by cross validation
```

```
hyperparams = range(1,31)

k_folds = 10
best_k_neighbours, best_accuracy, accuracies = cross_validation_knn(k_folds, hyperparams, train_inputs, train_labels)

# plot results
plot_knn_accuracies(accuracies, hyperparams)
print('best # of neighbours k: ' + str(best_k_neighbours))
print('best cross validation accuracy: ' + str(best_accuracy))

# evaluate with best # of neighbours
accuracy = eval_knn(test_inputs, test_labels, train_inputs, train_labels, best_k_neighbours)
print('test accuracy: '+ str(accuracy))
```




```
best # of neighbours k: 6
best cross validation accuracy: 79.3
test accuracy: 78.18181818181819
```