In [1]:
```python
import numpy as np

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

In [2]:
```python
# 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', encoding='utf-8') 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.]
```

In [3]:
```python
# 1a) Perform stratified data partition at a 70/15/15 ratio to yield Xtrain,
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, tra
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.
print('train: ', X_train.shape)
print('val: ', X_val.shape)
print('test: ', X_test.shape)
```

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

In [4]:
```python
# 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: ")
  models[i] = KNeighborsClassifier(n_neighbors=i).fit(X_train, y_train)
```

```
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 [5]:

```
# 1b) accuracy
from sklearn.metrics import accuracy_score

for i in models:
  y_predict = models[i].predict(X_val)  # Predict class using models[i]
  # acc = sum(y_predict == y_val) /  len(y_predict)
  acc = accuracy_score(y_val, y_predict)
  print(f'Classification Rate using {i} neighbors: {acc*100:.2f}%')
```

```
Classification Rate using 1 neighbors: 91.30%
Classification Rate using 3 neighbors: 95.65%
Classification Rate using 5 neighbors: 95.65%
Classification Rate using 10 neighbors: 95.65%
Classification Rate using 20 neighbors: 95.65%
Classification Rate using 30 neighbors: 95.65%
Classification Rate using 40 neighbors: 100.00%
Classification Rate using 50 neighbors: 86.96%
Classification Rate using 75 neighbors: 17.39%
Classification Rate using 100 neighbors: 13.04%
```

In [6]:

```
# 1c) Best number of neighbors
nneigs = 10
```

In [7]:

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

X_trainval = np.concatenate((X_train, X_val), axis=0)
y_trainval = np.concatenate((y_train, y_val), axis=0)
model = KNeighborsClassifier(n_neighbors=nneigs).fit(X_trainval, y_trainval)
y_predict = model.predict(X_test)

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

```
Classification Rate of 10 neighbors: 100.00%
Confusion Matrix of 10 neighbors:
[[11  0  0]
 [ 0  7  0]
 [ 0  0  4]]
```

In [8]:

```
# (Optional) Implement your own kNN classifier
# Retrain final classifier and apply it to test set to validate your impleme

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 = ...
        self.y_train = ...
        return self

    def predict(self, X):
        n = X.shape[0]
        y = np.zeros(n)
        for i in range(n):
            d = ...                           # Compute distances between X
            neig_idx = ...                    # Find closest neigbors using
            neig_y = self.y_train[neig_idx]   # Collect labels of closest n
            y[i] = ...                        # Find most likely label usin
        return y
```