In [1]:
```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive')

# TODO: Enter the foldername in your Drive where you have saved the unzip
# assignment folder, e.g. 'cs6353/assignments/assignment1/'
FOLDERNAME = 'MS/CS6353/assignment1/'
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My\ Drive/{}'.format(FOLDERNAME))

# This downloads the CIFAR-10 dataset to your Drive
# if it doesn't already exist.
%cd /content/drive/My\ Drive/$FOLDERNAME/cs6353/datasets/
!bash get_datasets.sh
%cd /content/drive/My\ Drive/$FOLDERNAME

# Install requirements from colab_requirements.txt
# TODO: Please change your path below to the colab_requirements.txt file
#! python -m pip install -r /content/drive/My\ Drive/$FOLDERNAME/colab_re
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount,
call drive.mount("/content/drive", force_remount=True).
/content/drive/My Drive/MS/CS6353/assignment1/cs6353/datasets
--2024-09-08 05:09:59--  http://www.cs.toronto.edu/~kriz/cifar-10-python.
tar.gz
Resolving www.cs.toronto.edu (www.cs.toronto.edu)... 128.100.3.30
Connecting to www.cs.toronto.edu (www.cs.toronto.edu)|128.100.3.30|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 170498071 (163M) [application/x-gzip]
Saving to: 'cifar-10-python.tar.gz'

cifar-10-python.tar 100%[===================>] 162.60M  31.2MB/s    in 5.
1s

2024-09-08 05:10:04 (31.7 MB/s) - 'cifar-10-python.tar.gz' saved [1704980
71/170498071]

cifar-10-batches-py/
cifar-10-batches-py/data_batch_4
cifar-10-batches-py/readme.html
cifar-10-batches-py/test_batch
cifar-10-batches-py/data_batch_3
cifar-10-batches-py/batches.meta
cifar-10-batches-py/data_batch_2
cifar-10-batches-py/data_batch_5
cifar-10-batches-py/data_batch_1
/content/drive/My Drive/MS/CS6353/assignment1
```

k-Nearest Neighbor (kNN) exercise

Complete and hand in this completed worksheet (including its outputs and any supporting code outside of the worksheet) with your assignment submission. For more

details see the assignments page on the course website.

The kNN classifier consists of two stages:

- During training, the classifier takes the training data and simply remembers it
- During testing, kNN classifies every test image by comparing to all training images and transferring the labels of the k most similar training examples
- The value of k is cross-validated

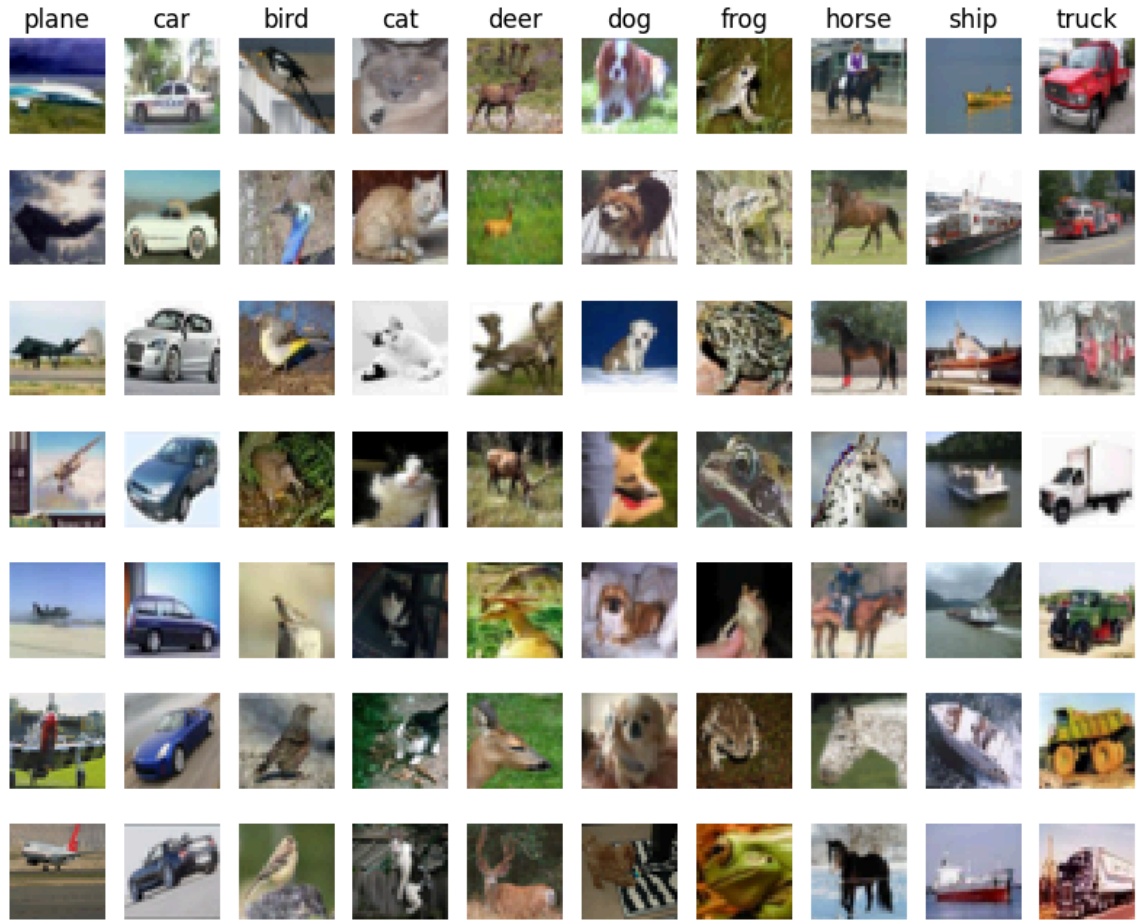
In this exercise you will implement these steps and understand the basic Image Classification pipeline, cross-validation, and gain proficiency in writing efficient, vectorized code.

In [2]:
```
# Run some setup code for this notebook.
from __future__ import print_function

import random
import numpy as np
from cs6353.data_utils import load_CIFAR10
import matplotlib.pyplot as plt
import copy

# This is a bit of magic to make matplotlib figures appear inline in the
# rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# Some more magic so that the notebook will reload external python module
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in
%load_ext autoreload
%autoreload 2
```

In [3]:
```
# Load the raw CIFAR-10 data.
cifar10_dir = 'cs6353/datasets/cifar-10-batches-py'

# Cleaning up variables to prevent loading data multiple times (which may
try:
   del X_train, y_train
   del X_test, y_test
   print('Clear previously loaded data.')
except:
   pass

X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

# As a sanity check, we print out the size of the training and test data.
print('Training data shape: ', X_train.shape)
print('Training labels shape: ', y_train.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Training data shape:  (50000, 32, 32, 3)
Training labels shape:  (50000,)
Test data shape:  (10000, 32, 32, 3)
Test labels shape:  (10000,)
```

In [4]:
```
# Visualize some examples from the dataset.
# We show a few examples of training images from each class.
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
num_classes = len(classes)
samples_per_class = 7
for y, cls in enumerate(classes):
    idxs = np.flatnonzero(y_train == y)
    idxs = np.random.choice(idxs, samples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(X_train[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls)
plt.show()
```



In [5]:
```
# Subsample the data for more efficient code execution in this exercise
num_training = 5000
mask = list(range(num_training))
X_train = X_train[mask]
y_train = y_train[mask]

num_test = 500
mask = list(range(num_test))
X_test = X_test[mask]
y_test = y_test[mask]
```

In [6]:
```
# Reshape the image data into rows
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))
print(X_train.shape, X_test.shape)
```

```
(5000, 3072) (500, 3072)
```

In [7]:
```
from cs6353.classifiers import KNearestNeighbor

# Create a kNN classifier instance.
# Remember that training a kNN classifier is a noop:
# the Classifier simply remembers the data and does no further processing
classifier = KNearestNeighbor()
classifier.train(X_train, y_train)
```

We would now like to classify the test data with the kNN classifier. Recall that we can break down this process into two steps:

1. First we must compute the distances between all test examples and all train examples.
2. Given these distances, for each test example we find the k nearest examples and have them vote for the label

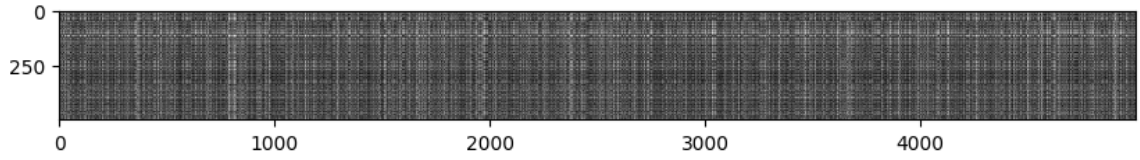
Lets begin with computing the distance matrix between all training and test examples. For example, if there are **Ntr** training examples and **Nte** test examples, this stage should result in a **Nte x Ntr** matrix where each element (i,j) is the distance between the i-th test and j-th train example.

First, open `cs6353/classifiers/k_nearest_neighbor.py` and implement the function `compute_distances_two_loops` that uses a (very inefficient) double loop over all pairs of (test, train) examples and computes the distance matrix one element at a time.

In [8]:
```
# Open cs6353/classifiers/k_nearest_neighbor.py and implement
# compute_distances_two_loops.

# Test your implementation:
dists = classifier.compute_distances_two_loops(X_test)
print(dists.shape)
```

```
(500, 5000)
```

In [9]:
```
# We can visualize the distance matrix: each row is a single test example
# its distances to training examples
plt.imshow(dists, interpolation='none')
plt.show()
```



Inline Question #1: Notice the structured patterns in the distance matrix, where some rows or columns are visible brighter. (Note that with the default color scheme black indicates low distances while white indicates high distances.)

- What in the data is the cause behind the distinctly bright rows?
- What causes the columns?

Your Answer:

- For brighter rows, it indicates that the test image distance is relatively high from all the training images. It suggests that there is low similarity between all the classes in the training labels to the test image.
- For brighter columns, it indicates that that particular training image has a relatively high distance to all test images. It suggests that there is low similarity between all the test images in the testing set to that training image.

In [10]:
```
# Now implement the function predict_labels and run the code below:
# We use k = 1 (which is Nearest Neighbor).
y_test_pred = classifier.predict_labels(dists, k=1)

# Compute and print the fraction of correctly predicted examples
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('Got %d / %d correct => accuracy: %f' % (num_correct, num_test, acc
```

```
Got 137 / 500 correct => accuracy: 0.274000
```

You should expect to see approximately `27%` accuracy. Now lets try out a larger `k`, say `k = 5`:

In [11]:
```
y_test_pred = classifier.predict_labels(dists, k=5)
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('Got %d / %d correct => accuracy: %f' % (num_correct, num_test, acc
```

```
Got 139 / 500 correct => accuracy: 0.278000
```

You should expect to see a slightly better performance than with `k = 1`.

Inline Question 2 We can also other distance metrics such as L1 distance. The performance of a Nearest Neighbor classifier that uses L1 distance will not change if (Select all that apply.):

1. The data is preprocessed by subtracting the mean.
2. The data is preprocessed by subtracting the mean and dividing by the standard deviation.
3. The coordinate axes for the data are rotated.
4. None of the above. (Mean and standard deviation in (1) and (2) are vectors and can be different across dimensions)

Your Answer:

1 and 2

Your explanation:

1. The mean will cancel out when computing distance as shown in the equation below because it is the same constant. By subtracting the mean to each data points, we are just shifting the data to center around the origin. Hence, no change in the distance.

|(p_i - mean) - (q_i - mean)| = | p_i - mean - q_i + mean| = |p_i - q_i|.

2. The standard deviation (sd) will only scale the distance by some constant. Each data point is scaled by the same sd constant, hence there is no difference in the distance comparison.

   |[(p_i - mean)]/sd - [(q_i - mean)]/sd| = |p_i - q_i|/sd.

3. L1 distance computation considers the axis orientation. Hence, if the data matrix is rotated, then the data matrix is transformed and it will more likely to produce different value for the distance.

In [12]:
```
# Now lets speed up distance matrix computation by using partial vectoriz
# with one loop. Implement the function compute_distances_one_loop and ru
# code below:
dists_one = classifier.compute_distances_one_loop(X_test)

# To ensure that our vectorized implementation is correct, we make sure t
# agrees with the naive implementation. There are many ways to decide whe
# two matrices are similar; one of the simplest is the Frobenius norm. In
# you haven't seen it before, the Frobenius norm of two matrices is the s
# root of the squared sum of differences of all elements; in other words,
# the matrices into vectors and compute the Euclidean distance between th
difference = np.linalg.norm(dists - dists_one, ord='fro')
print('Difference was: %f' % (difference, ))
if difference < 0.001:
    print('Good! The distance matrices are the same')
else:
    print('Uh-oh! The distance matrices are different')
```

```
Difference was: 0.000000
Good! The distance matrices are the same
```

In [13]:
```
# Now implement the fully vectorized version inside compute_distances_no_
# and run the code
dists_two = classifier.compute_distances_no_loops(X_test)

# check that the distance matrix agrees with the one we computed before:
difference = np.linalg.norm(dists - dists_two, ord='fro')
print('Difference was: %f' % (difference, ))
if difference < 0.001:
    print('Good! The distance matrices are the same')
else:
    print('Uh-oh! The distance matrices are different')
```

```
Difference was: 0.000000
Good! The distance matrices are the same
```

In [14]:
```
# Let's compare how fast the implementations are
def time_function(f, *args):
    """
    Call a function f with args and return the time (in seconds) that it
    """
    import time
    tic = time.time()
    f(*args)
    toc = time.time()
    return toc - tic

two_loop_time = time_function(classifier.compute_distances_two_loops, X_t
```

```
print('Two loop version took %f seconds' % two_loop_time)

one_loop_time = time_function(classifier.compute_distances_one_loop, X_te
print('One loop version took %f seconds' % one_loop_time)

no_loop_time = time_function(classifier.compute_distances_no_loops, X_tes
print('No loop version took %f seconds' % no_loop_time)

# you should see significantly faster performance with the fully vectoriz
```

```
Two loop version took 36.838056 seconds
One loop version took 37.831194 seconds
No loop version took 0.562591 seconds
```

Cross-validation

We have implemented the k-Nearest Neighbor classifier but we set the value k = 5 arbitrarily. We will now determine the best value of this hyperparameter with cross-validation.

In [15]:
```
num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

X_train_folds = []
y_train_folds = []
################################################################################
# TODO:
# Split up the training data into folds. After splitting, X_train_folds a
# y_train_folds should each be lists of length num_folds, where
# y_train_folds[i] is the label vector for the points in X_train_folds[i]
# Hint: Look up the numpy array_split function.
################################################################################
# Your code
X_train_folds = np.array_split(X_train, num_folds)
y_train_folds = np.array_split(y_train, num_folds)
################################################################################
#                                 END OF YOUR CODE
################################################################################

# A dictionary holding the accuracies for different values of k that we f
# when running cross-validation. After running cross-validation,
# k_to_accuracies[k] should be a list of length num_folds giving the diff
# accuracy values that we found when using that value of k.
k_to_accuracies = {}


################################################################################
# TODO:
# Perform k-fold cross validation to find the best value of k. For each
# possible value of k, run the k-nearest-neighbor algorithm num_folds tim
# where in each case you use all but one of the folds as training data an
# last fold as a validation set. Store the accuracies for all fold and al
# values of k in the k_to_accuracies dictionary.
################################################################################
# Your code
for k in k_choices:
  accuracies = []
  for i in range(num_folds):
      classifier = KNearestNeighbor()

      X_train_without_valid_set = np.copy(X_train_folds)
```

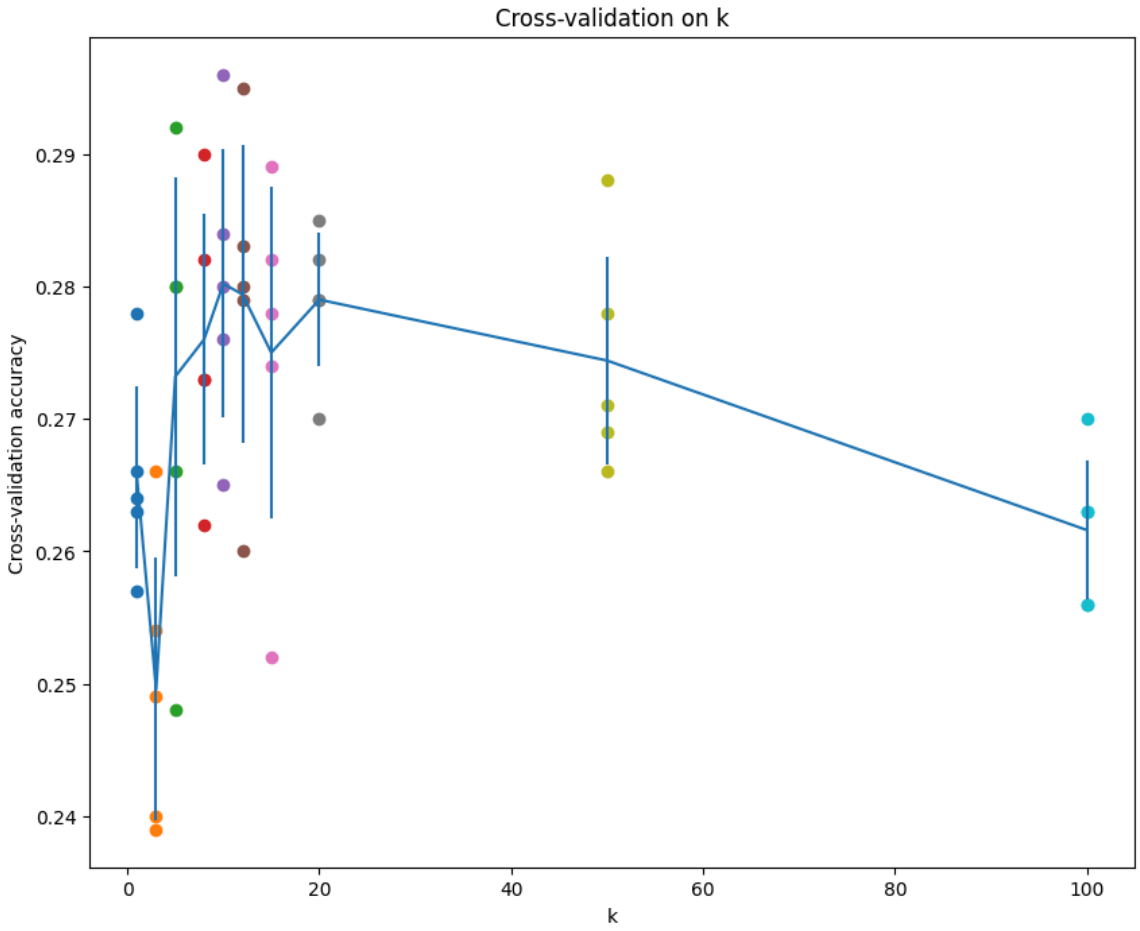
```
      X_train_without_valid_set = np.concatenate(np.delete(X_train_withou
      y_train_without_valid_set = np.copy(y_train_folds)
      y_train_without_valid_set = np.concatenate(np.delete(y_train_withou

      classifier.train(X_train_without_valid_set, y_train_without_valid_s

      y_pred = classifier.predict(X_train_folds[i], k=k)

      acc = np.mean(y_train_folds[i] == y_pred)
      accuracies.append(acc)
  k_to_accuracies[k] = accuracies
################################################################################
#                                  END OF YOUR CODE
################################################################################

# Print out the computed accuracies
for k in sorted(k_to_accuracies):
    for accuracy in k_to_accuracies[k]:
        print('k = %d, accuracy = %f' % (k, accuracy))
```

```
k = 1, accuracy = 0.263000
k = 1, accuracy = 0.257000
k = 1, accuracy = 0.264000
k = 1, accuracy = 0.278000
k = 1, accuracy = 0.266000
k = 3, accuracy = 0.239000
k = 3, accuracy = 0.249000
k = 3, accuracy = 0.240000
k = 3, accuracy = 0.266000
k = 3, accuracy = 0.254000
k = 5, accuracy = 0.248000
k = 5, accuracy = 0.266000
k = 5, accuracy = 0.280000
k = 5, accuracy = 0.292000
k = 5, accuracy = 0.280000
k = 8, accuracy = 0.262000
k = 8, accuracy = 0.282000
k = 8, accuracy = 0.273000
k = 8, accuracy = 0.290000
k = 8, accuracy = 0.273000
k = 10, accuracy = 0.265000
k = 10, accuracy = 0.296000
k = 10, accuracy = 0.276000
k = 10, accuracy = 0.284000
k = 10, accuracy = 0.280000
k = 12, accuracy = 0.260000
k = 12, accuracy = 0.295000
k = 12, accuracy = 0.279000
k = 12, accuracy = 0.283000
k = 12, accuracy = 0.280000
k = 15, accuracy = 0.252000
k = 15, accuracy = 0.289000
k = 15, accuracy = 0.278000
k = 15, accuracy = 0.282000
k = 15, accuracy = 0.274000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.279000
k = 20, accuracy = 0.279000
k = 20, accuracy = 0.282000
k = 20, accuracy = 0.285000
k = 50, accuracy = 0.271000
k = 50, accuracy = 0.288000
k = 50, accuracy = 0.278000
k = 50, accuracy = 0.269000
k = 50, accuracy = 0.266000
k = 100, accuracy = 0.256000
k = 100, accuracy = 0.270000
k = 100, accuracy = 0.263000
k = 100, accuracy = 0.256000
k = 100, accuracy = 0.263000
```

In [16]:

```
# plot the raw observations
for k in k_choices:
    accuracies = k_to_accuracies[k]
    plt.scatter([k] * len(accuracies), accuracies)

# plot the trend line with error bars that correspond to standard deviati
accuracies_mean = np.array([np.mean(v) for k,v in sorted(k_to_accuracies.
accuracies_std = np.array([np.std(v) for k,v in sorted(k_to_accuracies.it
plt.errorbar(k_choices, accuracies_mean, yerr=accuracies_std)
plt.title('Cross-validation on k')
plt.xlabel('k')
```

```
plt.ylabel('Cross-validation accuracy')
plt.show()
```



In [17]:
```
# Based on the cross-validation results above, choose the best value for
# retrain the classifier using all the training data, and test it on the
# data. You should be able to get above 28% accuracy on the test data.
best_k = k_choices[accuracies_mean.argmax()]

classifier = KNearestNeighbor()
classifier.train(X_train, y_train)
y_test_pred = classifier.predict(X_test, k=best_k)

# Compute and display the accuracy
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('Got %d / %d correct => accuracy: %f' % (num_correct, num_test, acc
```

```
Got 141 / 500 correct => accuracy: 0.282000
```

Inline Question 3 Which of the following statements about k-Nearest Neighbor (k-NN) are true in a classification setting, and for all k? Select all that apply.

1. The training accuracy of a 1-NN (nearest neighbor) will always be better than or equal to that of 5-NN.
2. The test accuracy of a 1-NN will always be better than that of a 5-NN.
3. The decision boundary of the k-NN classifier is linear.
4. The time needed to classify a test example with the k-NN classifier grows with the size of the training set.
5. It remembers all of the training data.
6. remembers all of the test data.

7. KNN cannot be applied for distance metrics other than L1 and L2.
8. KNN does more computation during test time than training time.
9. KNN does more computation during training time than test time.
10. KNN does equal computation during training time and test time.
11. KNN does equal computation during training time and test time.

Your Answer:

True : 1, 4, 5, 8

Your explanation:

1. *True* : Because in 1-NN all data points in training set are classified to the single closest neighbour which is itself. Hence, there is no error.
2. *False* : Because there is a chance of misclassification in the testing phase. 1-NN has a higher possibility of misclassification than 5-NN. Because in 5-NN it has more options to consider so it might get the wrong label for the closest image but correct class from the other closest images.
3. *False* : Because the data is not always linearly seperable. Since it is n-dimensional space, it may have a complex space and can not be represented by a hyperplane.
4. *True* : Because when kNN model is classifying for a test example, it went through all the training examples and do the computation to classify. Hence, more data in training set, more time for computation.
5. *True* : Because in the training phase, it stores all the data but no computation.
6. *False* : Because the test data is not used in the training phase and in testing phase, it just do the computation to classify the images but not storing them.
7. *False* : Because L1 and L2 are just two techniques for measuring distance. Other techniques for measuring distance can still be applied to kNN algorithm concept.
8. *True* : Because all computation is done in testing phase.
9. *False* : Because there is no computation in training phase.
10. *False* : Because all computation is done in testing phase and there is no computation in training phase.

In [24]:
```
!jupyter nbconvert --to html /content/drive/MyDrive/MS/CS6353/assignment1
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/MS/CS6353/assig
nment1/knn.ipynb to html
[NbConvertApp] Writing 1165688 bytes to /content/drive/MyDrive/MS/CS6353/
assignment1/knn.html
```