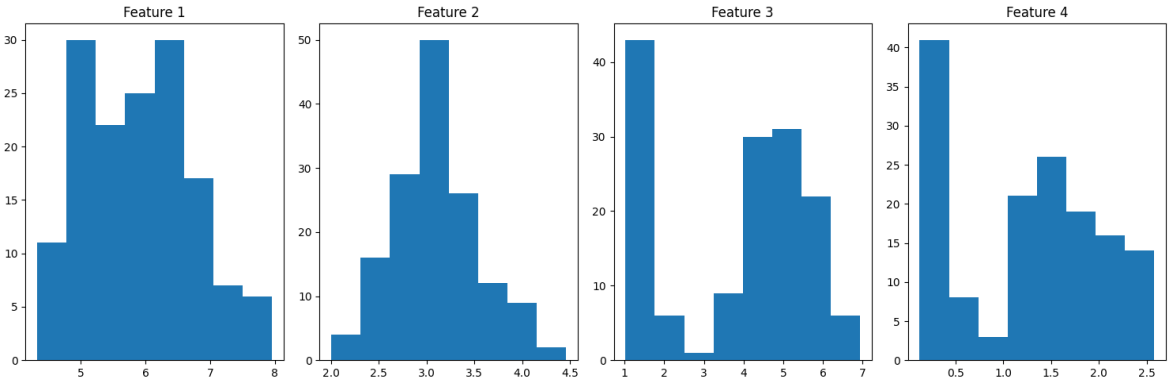
Problem 1: Python & Data Exploration

In [ ]:
```
# Write and run your code here
import numpy as np
import matplotlib.pyplot as plt

iris = np.genfromtxt("data/iris.txt",delimiter=None) # load the text file
Y = iris[:,-1] # target value is the last column
X = iris[:,0:-1] # features are the other columns
```

In [ ]:
```
# 1. Use X.shape to get the number of features and the data points. Repor
data_points, features = X.shape
print("Number of features: ", features)
print("Number of data points: ", data_points)
```

```
Number of features:  4
Number of data points:  148
```

In [ ]:
```
# 2. For each feature, plot a histogram ( plt.hist ) of the data values
bins = int(1 + np.log2(data_points))

plt.figure(figsize=(15, 5))

for i in range(features):
    plt.subplot(1, features, i+1)
    plt.hist(X[:, i], bins)
    plt.title(f'Feature {i+1}')

plt.tight_layout()
plt.show()
```

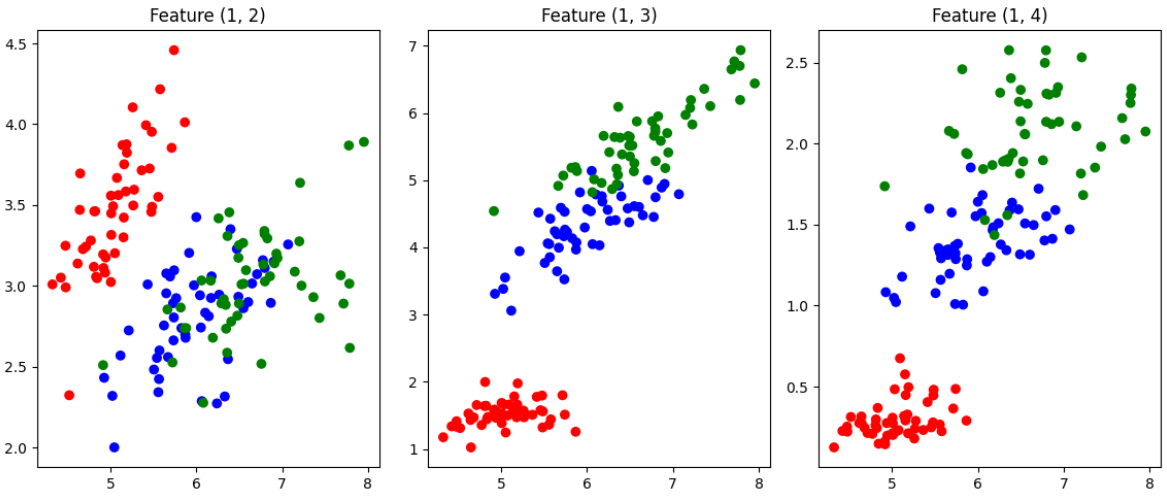


In [ ]:
```
# 3. Compute the mean & standard deviation of the data points for each fe
means = np.mean(X, axis=0)
standard_deviation = np.std(X, axis=0)

for i in range(4):
    print("Feature ", i+1)
    print("Mean: ", means[i])
    print("Standard Deviation: ", standard_deviation[i])
```

```
Feature  1
Mean:  5.900103764189187
Standard Deviation:  0.8334020667748939
Feature  2
Mean:  3.0989309168918915
Standard Deviation:  0.43629183800107685
Feature  3
Mean:  3.8195548405405413
Standard Deviation:  1.7540571093439352
Feature  4
Mean:  1.252555484594594
Standard Deviation:  0.7587724570263246
```

In [ ]:

```
# 4. For each pair of features (1,2), (1,3), and (1,4), plot a scatterplo

colors = []
for y in Y:
    if y == 0:
        colors.append('red')
    elif y == 1:
        colors.append('blue')
    else:
        colors.append('green')

plt.figure(figsize=(15, 5))

for i in range(1, features):
    plt.subplot(1, features, i+1)
    plt.scatter(X[:,0], X[:,i], c=colors)
    plt.title(f'Feature (1, {i+1})')

plt.tight_layout()
plt.show()
```



Problem 2: k-Nearest Neighbor (kNN) exercise

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 transfering 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 [ ]:
```
# Run some setup code for this notebook.

import random
import numpy as np
from cs273p.data_utils import load_CIFAR10
import matplotlib.pyplot as plt

# 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 [ ]:
```
%cd cs273p/datasets
!source get_datasets.sh
```

```
/Users/harryxiong24/Code/Study/grad-code-collection/ML/hw/hw1/cs273p/datas
ets
--2024-01-19 10:00:53--  http://www.cs.toronto.edu/~kriz/cifar-10-python.t
ar.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...
/Users/harryxiong24/Library/Python/3.10/lib/python/site-packages/IPython/c
ore/magics/osm.py:417: UserWarning: using dhist requires you to install th
e `pickleshare` library.
  self.shell.db['dhist'] = compress_dhist(dhist)[-100:]
```

```
200 OK
Length: 170498071 (163M) [application/x-gzip]
Saving to: ‘cifar-10-python.tar.gz’

cifar-10-python.tar 100%[===================>] 162.60M  13.7MB/s    in 18s

2024-01-19 10:01:12 (8.94 MB/s) - ‘cifar-10-python.tar.gz’ saved [17049807
1/170498071]

x cifar-10-batches-py/
x cifar-10-batches-py/data_batch_4
x cifar-10-batches-py/readme.html
x cifar-10-batches-py/test_batch
x cifar-10-batches-py/data_batch_3
x cifar-10-batches-py/batches.meta
x cifar-10-batches-py/data_batch_2
x cifar-10-batches-py/data_batch_5
x cifar-10-batches-py/data_batch_1
```

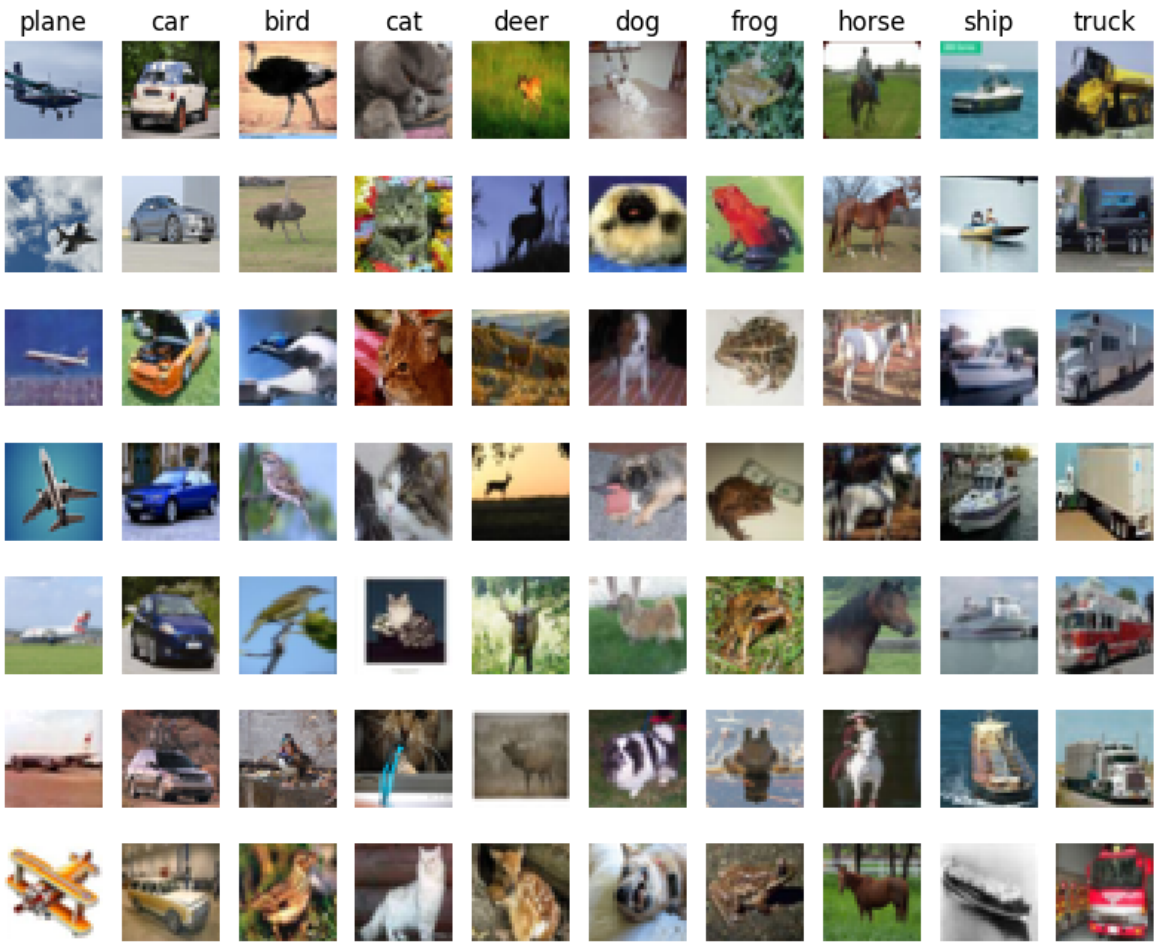
In [ ]:
```
%cd ../..
```

```
/Users/harryxiong24/Code/Study/grad-code-collection/ML/hw/hw1
```

In [ ]:
```
# Load the raw CIFAR-10 data.
cifar10_dir = './cs273p/datasets/cifar-10-batches-py'
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 [ ]:
```
# 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 [ ]:
```
# 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 [ ]:
```
# 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 [ ]:
```
from cs273p.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

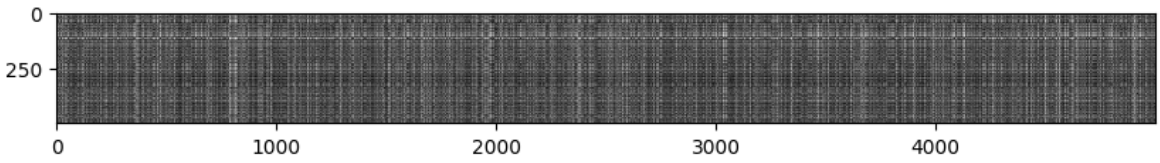
Lets begin with computing the distance matrix between all training and test examples. For example, if there are **Ntr** training examples and **Nte** test examples, this stage should result in a **Nte x Ntr** matrix where each element (i,j) is the distance between the i-th test and j-th train example.

First, open `cs273p/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 [ ]:
```
# Open cs273p/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 [ ]:
```
# 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: *fill this in.*

1. A distinctly bright row in the distance matrix indicates that a specific test sample has a high distance to almost all training samples. The reason for this may be because of the outlier test sample.

2. A distinctly bright column in the distance matrix indicates that a specific training sample has a high distance to almost all test samples. The reason for this may be because of the outlier training sample.

In [ ]:
```
# 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 [ ]:
```
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 145 / 500 correct => accuracy: 0.290000
```

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

In [ ]:
```
# 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: 551335735.545143
Uh-oh! The distance matrices are different
```

In [ ]:
```
# 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 [ ]:
```
# 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 14.648581 seconds
One loop version took 13.580517 seconds
No loop version took 0.129227 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.

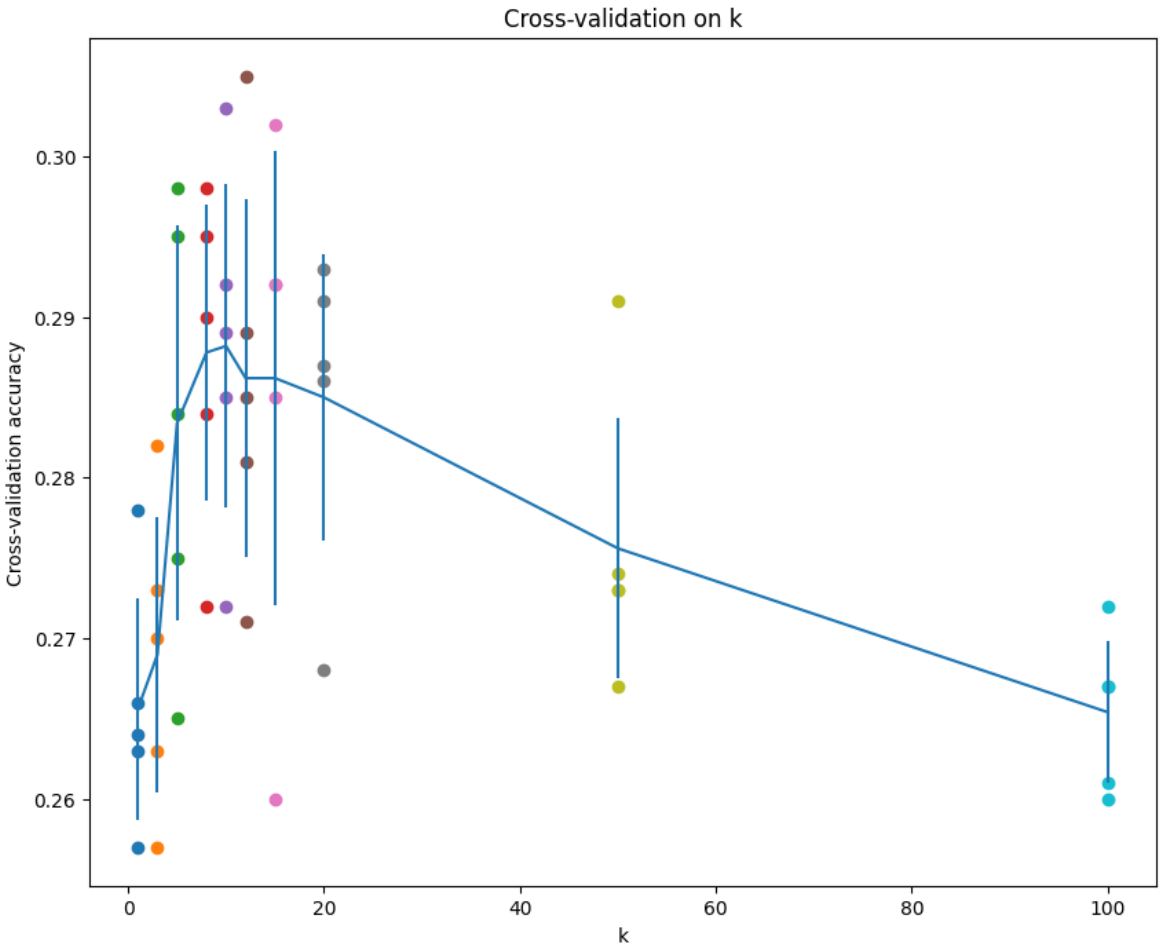
In [ ]:
```
num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

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.
################################################################################
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.
################################################################################
for k in k_choices:
    accuracies = []
    for i in range(num_folds):
        # Use the ith fold as the validation set and the rest as the trai
        X_val_fold = X_train_folds[i]
        y_val_fold = y_train_folds[i]
        X_train_fold = np.concatenate([X_train_folds[j] for j in range(nu
        y_train_fold = np.concatenate([y_train_folds[j] for j in range(nu

        # Create a k-NN classifier and train it
        classifier = KNearestNeighbor()
        classifier.train(X_train_fold, y_train_fold)

        # Predict labels for the validation set
        y_val_pred = classifier.predict(X_val_fold, k=k, num_loops=0)

        # Compute and store the accuracy
        num_correct = np.sum(y_val_pred == y_val_fold)
        accuracy = float(num_correct) / len(y_val_fold)
        accuracies.append(accuracy)

    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 = 0.257000
k = 3, accuracy = 0.263000
k = 3, accuracy = 0.273000
k = 3, accuracy = 0.282000
k = 3, accuracy = 0.270000
k = 5, accuracy = 0.265000
k = 5, accuracy = 0.275000
k = 5, accuracy = 0.295000
k = 5, accuracy = 0.298000
k = 5, accuracy = 0.284000
k = 8, accuracy = 0.272000
k = 8, accuracy = 0.295000
k = 8, accuracy = 0.284000
k = 8, accuracy = 0.298000
k = 8, accuracy = 0.290000
k = 10, accuracy = 0.272000
k = 10, accuracy = 0.303000
k = 10, accuracy = 0.289000
k = 10, accuracy = 0.292000
k = 10, accuracy = 0.285000
k = 12, accuracy = 0.271000
k = 12, accuracy = 0.305000
k = 12, accuracy = 0.285000
k = 12, accuracy = 0.289000
k = 12, accuracy = 0.281000
k = 15, accuracy = 0.260000
k = 15, accuracy = 0.302000
k = 15, accuracy = 0.292000
k = 15, accuracy = 0.292000
k = 15, accuracy = 0.285000
k = 20, accuracy = 0.268000
k = 20, accuracy = 0.293000
k = 20, accuracy = 0.291000
k = 20, accuracy = 0.287000
k = 20, accuracy = 0.286000
k = 50, accuracy = 0.273000
k = 50, accuracy = 0.291000
k = 50, accuracy = 0.274000
k = 50, accuracy = 0.267000
k = 50, accuracy = 0.273000
k = 100, accuracy = 0.261000
k = 100, accuracy = 0.272000
k = 100, accuracy = 0.267000
k = 100, accuracy = 0.260000
k = 100, accuracy = 0.267000
```

In [ ]:
```
# 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 [ ]:
```
# 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 = 10

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 144 / 500 correct => accuracy: 0.288000
```

Problem 3: Naïve Bayes Classifiers

You don't need to code this question. You can either type your answer or attach an image of hand written solution here.

Answer:

In order to reduce my email load, I decide to implement a machine learning algorithm to decide whether or not I should read an email, or simply file it away instead. To train my model, I obtain the following data set of binary-valued features about each email, including whether I know the author or not, whether the email is long or short, and whether it has any of several key words, along with my final decision about whether to read it ($y = +1$ for "read", $y = -1$ for "discard").

x_1 know author?	x_2 is long?	x_3 has 'research'	x_4 has 'grade'	x_5 has 'lottery'	y $\Rightarrow$ read?
0	0	1	1	0	-1
1	1	0	1	0	-1
0	1	1	1	1	-1
1	1	1	1	0	-1
0	1	0	0	0	-1
1	0	1	1	1	1
0	0	1	0	0	1
1	0	0	0	0	1
1	0	1	1	0	1
1	1	1	1	1	-1

In the case of any ties, we will prefer to predict class +1.
I decide to try a naïve Bayes classifier to make my decisions and compute my uncertainty.

1. Compute all the probabilities necessary for a naïve Bayes classifier, i.e., the class probability $p(y)$ and all the individual feature probabilities $p(x_i|y)$, for each class y and feature x_i `(10 points)`

1. $P(y=1) = \frac{4}{10} = \frac{2}{5}$

$P(y=-1) = \frac{6}{10} = \frac{3}{5}$

$P(x_1=1 \mid y=1) = \frac{P(x_1 \cap y=1)}{P(y=1)} = \frac{\frac{3}{10}}{\frac{2}{5}} = \frac{3}{4}$

$P(x_1=1 \mid y=-1) = \frac{3}{6} = \frac{1}{2}$

$P(x_1=0 \mid y=1) = \frac{1}{4}$

$P(x_1=0 \mid y=-1) = \frac{1}{2}$

$P(x_2=1 \mid y=-1) = \frac{0}{4} = 0$

$P(x_2=1 \mid y=-1) = \frac{5}{6}$

$P(X_2=0 \mid y=1) = 1$

$P(X_2=0 \mid y=-1) = \frac{1}{6}$

$P(X_3=1 \mid y=1) = \frac{3}{4}$

$P(X_3=1 \mid y=-1) = \frac{4}{6} = \frac{2}{3}$

$P(X_3=0 \mid y=1) = \frac{1}{4}$

$P(X_3=0 \mid y=1) = \frac{1}{3}$

$P(X_4=1 \mid y=1) = \frac{2}{4} = \frac{1}{2}$

$P(X_4=1 \mid y=-1) = \frac{5}{6}$

$P(X_4=0 \mid y=1) = \frac{1}{2}$

$P(X_4=0 \mid y=-1) = \frac{1}{6}$

$P(X_5=1 \mid y=1) = \frac{1}{4}$

$P(X_5=1 \mid y=-1) = \frac{2}{6} = \frac{1}{3}$

$P(X_5=0 \mid y=1) = \frac{3}{4}$

$P(X_5=0 \mid y=-1) = \frac{2}{3}$

2. Which class would be predicted for $\underline{x} = (0\ 0\ 0\ 0\ 0)$? What about for $\underline{x} = (1\ 1\ 0\ 1\ 0)$? (10 points)

① $X = (0\ 0\ 0\ 0\ 0)$

$P(y=1|X) = P(y=1) \times P(x_1=0|y=1) \times P(x_2=0|y=1) \times P(x_3=0|y=1) \times P(x_4=0|y=1) \times P(x_5=0|y=1)$

$= \frac{2}{5} \times \frac{1}{4} \times 1 \times \frac{1}{4} \times \frac{1}{2} \times \frac{3}{4}$

$= \frac{3}{320} = 0.0094$

$P(y=-1|X) = P(y=-1) P(x_1=0|y=-1) P(x_2=0|y=-1) P(x_3=0|y=-1) P(x_4=0|y=-1) P(x_5=0|y=-1)$

$= \frac{3}{5} \times \frac{1}{2} \times \frac{1}{6} \times \frac{1}{3} \times \frac{1}{6} \times \frac{2}{3}$

$= \frac{1}{540} = 0.0019$

$P(y=1|X) > P(y=1|X)$

Predicted for $x=(00000)$ is 1

② $x=(11010)$

$P(y=1|x) = P(y=1)P(x_1=1|y=1)$
$P(x_2=1|y=1)P(x_3=0|y=1)$
$P(x_4=1|y=1)P(x_5=0|y=1)$
$= \frac{2}{5} \times \frac{3}{4} \times 0 \times \frac{1}{4} \times \frac{1}{2} \times \frac{3}{4}$
$= 0$

$P(y=-1|x) = P(y=-1)P(x_1=1|y=-1)$
$P(x_2=1|y=-1)P(x_3=0|y=-1)$
$P(x_4=1|y=-1)P(x_5=0|y=-1)$
$= \frac{3}{5} \times \frac{1}{2} \times \frac{5}{6} \times \frac{1}{3} \times \frac{5}{6} \times \frac{2}{3}$
$= \frac{5}{108} = 0.047$

$P(y=-1|x) > P(y=1|x)$

Predict for $x=(1\,1\,0\,1\,0)$ is -1

3. Compute the posterior probability that $y = +1$ given the observation $\underline{x} = (1\ 1\ 0\ 1\ 0)$. `(5 points)`

$$P(y=1|X) = P(x_1=1|y=1)P(x_2=1|y=1)$$
$$P(x_3=0|y=1)P(x_4=1|y=1)P(x_5=0|y=1)$$
$$= \frac{3}{4} \times 0 \times \frac{1}{4} \times \frac{1}{2} \times \frac{3}{4}$$
$$= \frac{9}{128}$$

4. Why should we probably not use a "joint" Bayes classifier (using the joint probability of the features x, as opposed to a naïve Bayes classifier) for these data? `(10 points)`

① High computational consumption

② As the number of features increases, the amount of data needed to reliably estimate the joint probability

distribution grows exponentially.

5. Suppose that, before we make our predictions, we lose access to my address book, so that we cannot tell whether the email author is known. Should we re-train the model, and if so, how? (e.g.: how does the model, and its parameters, change in this new situation?) Hint: what will the naïve Bayes model over only features $x_2 \dots x_5$ look like, and what will its parameters be? `(10 points)`

We need to retrain.

Because we lose X1's data. But original model was trained with X1 would be available.

And parameters will be the probabilities associated with features x2 to x5.

Statement of Collaboration

I do this homework by myself.