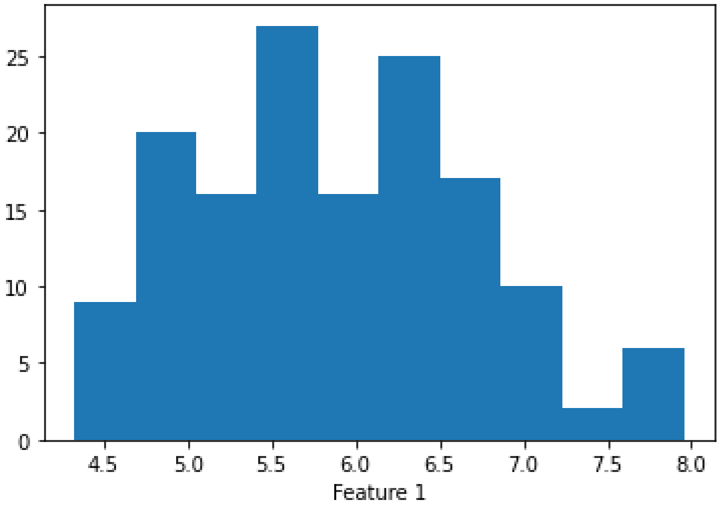
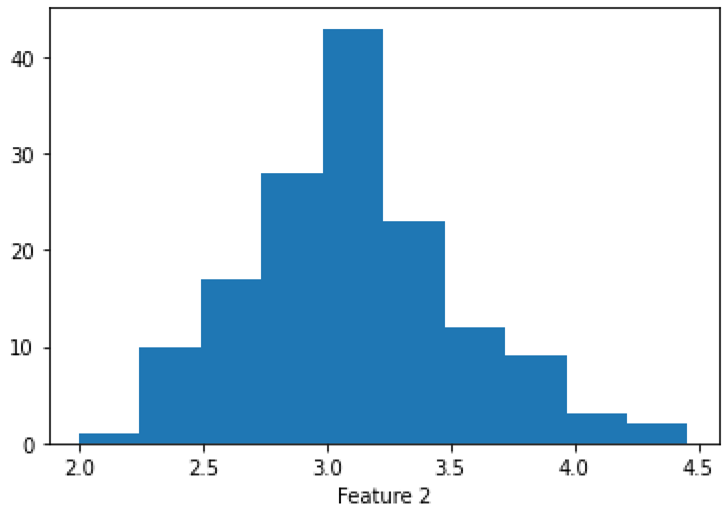
Problem 1: Python & Data Exploration

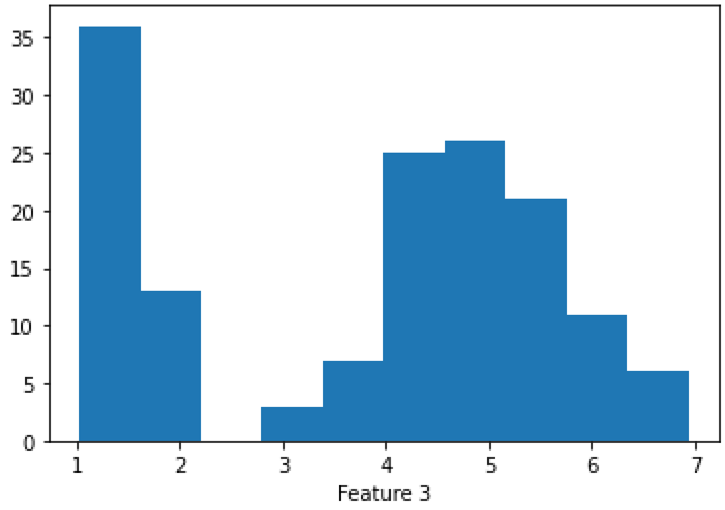
In [1]:

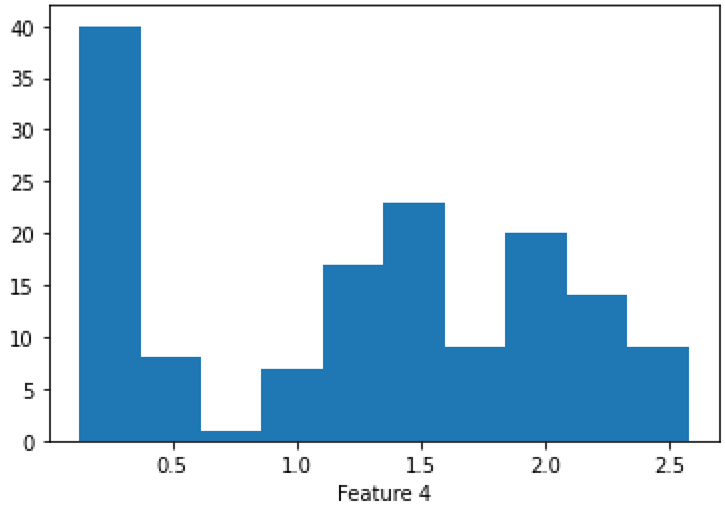
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

iris = np.genfromtxt("data/iris.txt", delimiter=None)
Y = iris[:,-1]
X = iris[:,0:-1]

print(X.shape)
print("Number of data points is", X.shape[0])
print("Number of features is", X.shape[1])

for i in range(X.shape[1]):
    plt.hist(X[:,i])
    f = "Feature " + str(i + 1)
    plt.xlabel(f)
    plt.show()

print("The mean of each feature is", np.mean(X,axis=0))
print("The standard deviation of each feature is", np.std(X,axis=0))


label = np.array([int(i) for i in list(Y)])
colors = np.array(["b", "g", "r"])

plt.scatter(X[:,0], X[:,1], c=colors[label])
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()

plt.scatter(X[:,0], X[:,2], c=colors[label])
plt.xlabel("Feature 1")
plt.ylabel("Feature 3")
plt.show()

plt.scatter(X[:,0], X[:,3], c=colors[label])
plt.xlabel("Feature 1")
plt.ylabel("Feature 4")
plt.show()
```

```
(148, 4)
Number of data points is 148
Number of features is 4
```

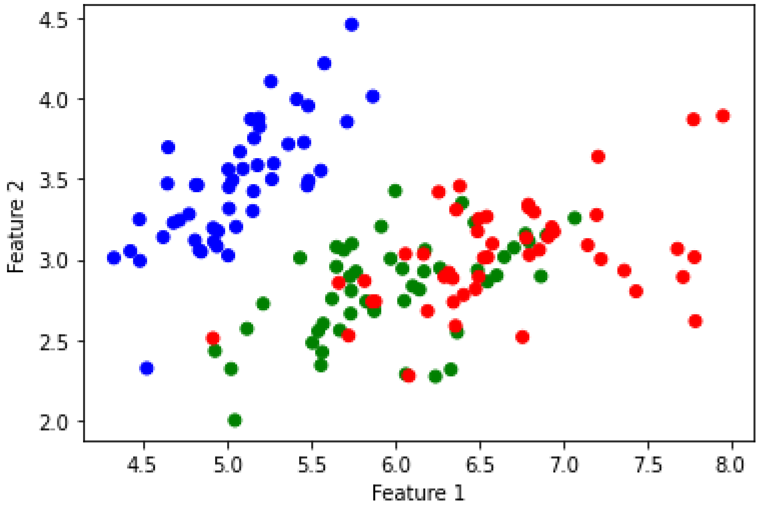


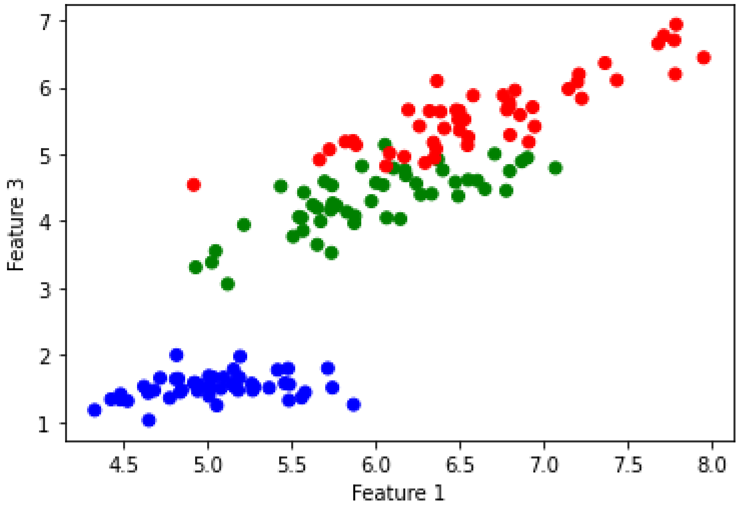


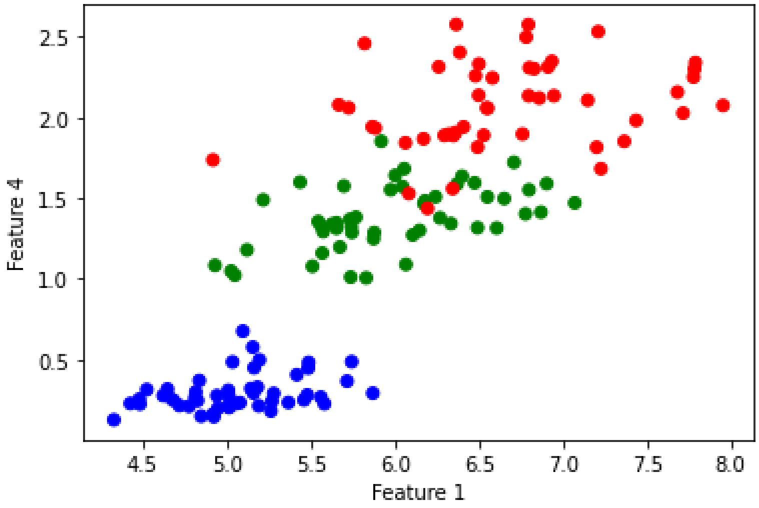




```
The mean of each feature is [5.90010376 3.09893092 3.81955484 1.25255548]
The standard deviation of each feature is [0.83340207 0.43629184 1.75405711 0.75877246]
```







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 [2]:

```
# Run some setup code for this notebook.

import random
import numpy as np
from cs178.data_utils import load_CIFAR10
import matplotlib.pyplot as plt

# This is a bit of magic to make matplotlib figures appear inline in the notebook
# 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 modules;
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
# %load_ext autoreload
# %autoreload 2
```

In [3]:

```
%cd cs178/datasets
!source get_datasets.sh
```

```
/Users/andytran/Desktop/hw1/cs178/datasets
get_datasets.sh:2: command not found: wget
tar: Error opening archive: Failed to open 'cifar-10-python.tar.gz'
rm: cifar-10-python.tar.gz: No such file or directory
```

In [4]:

```
%cd ../..
```

```
/Users/andytran/Desktop/hw1
```

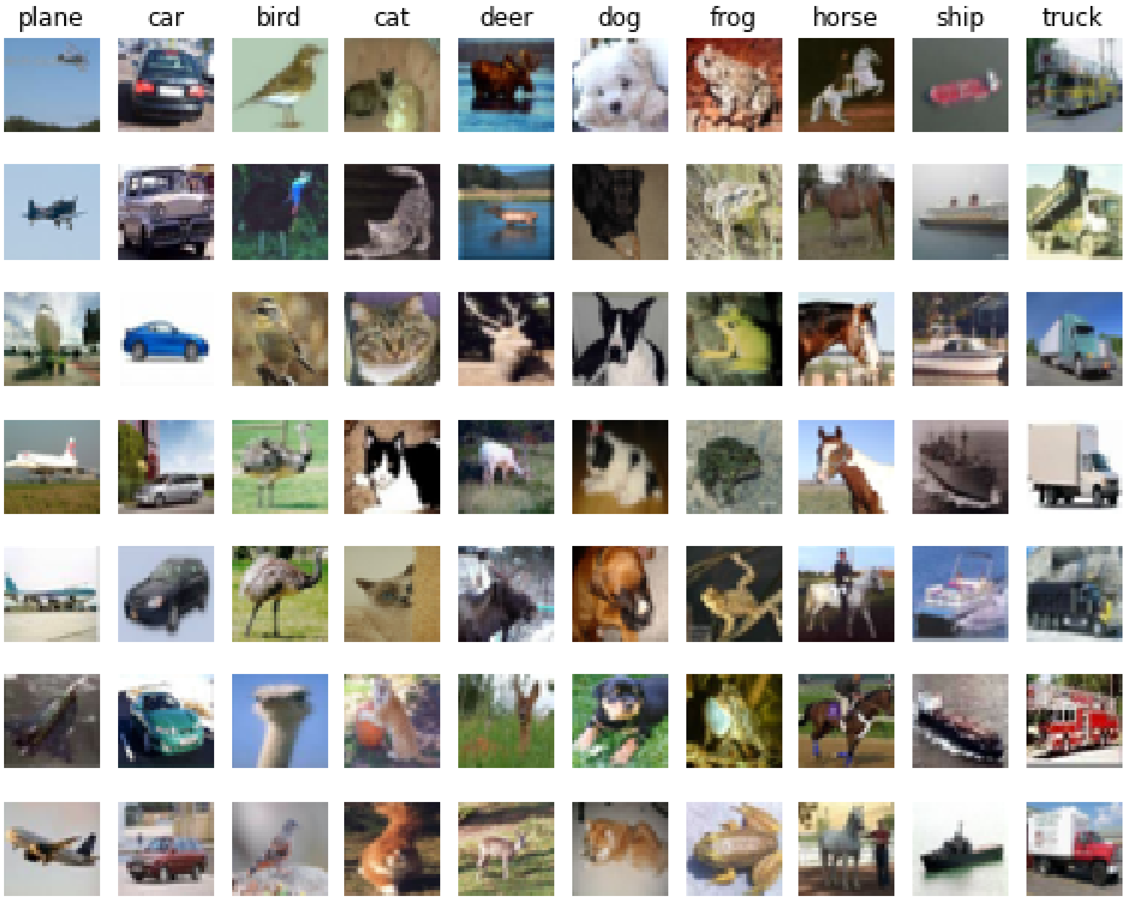
In [5]:

```
# Load the raw CIFAR-10 data.
cifar10_dir = './cs178/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 [6]:

```
# 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', 'ship', 'truck
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 [7]:

```python
# 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 [8]:

```python
# 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 [9]:

```python
from cs178.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 `cs178/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.

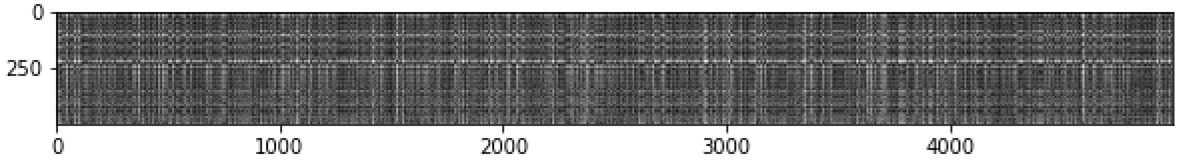
In [10]:

```
# Open cs178/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 [11]:

```
# We can visualize the distance matrix: each row is a single test example and
# 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: The reason for the rows to be distinctly bright is because the testing data is far from the training data. The reason for the columns to be distinctly bright is because the training data is far from the testing data.

In [12]:

```
# 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, accuracy))
```

```
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 [13]:

```
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, accuracy))
```

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

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

In [14]:

```
# Now lets speed up distance matrix computation by using partial vectorization
# with one loop. Implement the function compute_distances_one_loop and run the
# code below:
dists_one = classifier.compute_distances_one_loop(X_test)

# To ensure that our vectorized implementation is correct, we make sure that it
# agrees with the naive implementation. There are many ways to decide whether
# two matrices are similar; one of the simplest is the Frobenius norm. In case
# you haven't seen it before, the Frobenius norm of two matrices is the square
# root of the squared sum of differences of all elements; in other words, reshape
# the matrices into vectors and compute the Euclidean distance between them.
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 [15]:

```
# Now implement the fully vectorized version inside compute_distances_no_loops
# 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 [16]:

```
# 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 took to execute
    """
    import time
    tic = time.time()
    f(*args)
    toc = time.time()
    return toc - tic

two_loop_time = time_function(classifier.compute_distances_two_loops, X_test)
print('Two loop version took %f seconds' % two_loop_time)

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

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

# you should see significantly faster performance with the fully vectorized implementati
```

```
Two loop version took 28.183744 seconds
One loop version took 23.103608 seconds
No loop version took 0.130528 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 [17]:

```
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 and    #
# 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, axis=0)
y_train_folds = np.array_split(y_train, num_folds, axis=0)
################################################################################
#                                 END OF YOUR CODE                             #
################################################################################

# A dictionary holding the accuracies for different values of k that we find
# when running cross-validation. After running cross-validation,
# k_to_accuracies[k] should be a list of length num_folds giving the different
# 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 times,   #
# where in each case you use all but one of the folds as training data and the #
# last fold as a validation set. Store the accuracies for all fold and all     #
# values of k in the k_to_accuracies dictionary.                               #
################################################################################

for k_val in k_choices:

    data_for_k = []

    for i in range(num_folds):

        cur_x_train = np.concatenate(X_train_folds[:i] + X_train_folds[i+1:])
        cur_y_train = np.concatenate(y_train_folds[:i] + y_train_folds[i+1:])

        cur_x_test = X_train_folds[i]
        cur_y_test = y_train_folds[i]

        cur_classifier = KNearestNeighbor()
        cur_classifier.train(cur_x_train, cur_y_train)

        cur_dists = cur_classifier.compute_distances_no_loops(cur_x_test)
        cur_y_test_pred = cur_classifier.predict_labels(cur_dists, k=k_val)

        cur_num_correct = np.sum(cur_y_test_pred == cur_y_test)
        cur_num_test = cur_x_test.shape[0]
        cur_accuracy = float(cur_num_correct) / cur_num_test

        data_for_k.append(cur_accuracy)

    k_to_accuracies[k_val] = data_for_k

################################################################################
```

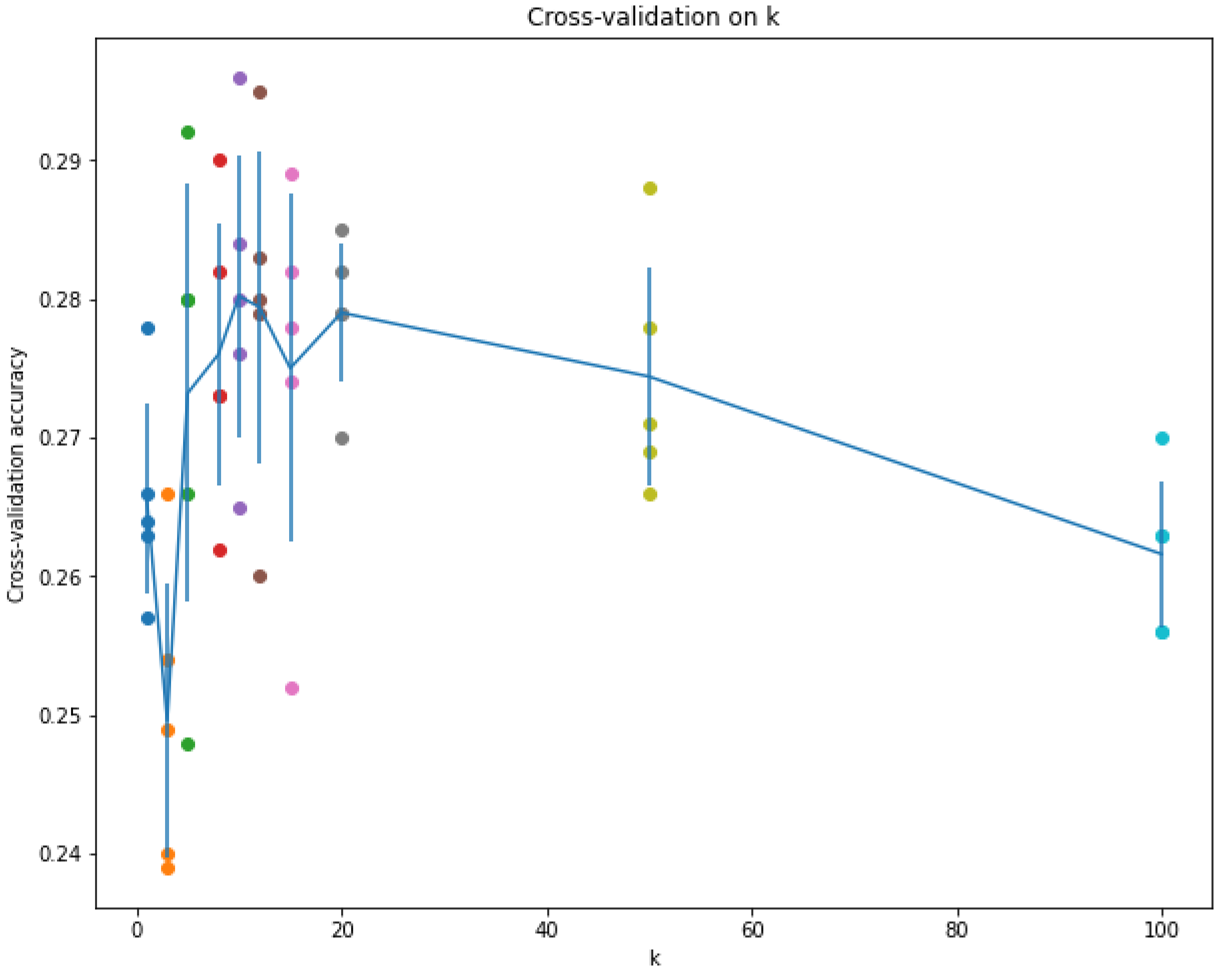
```
#                                 END OF YOUR CODE                              #
################################################################################

# 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 [18]:

```
# 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 deviation
accuracies_mean = np.array([np.mean(v) for k,v in sorted(k_to_accuracies.items())])
```

```
accuracies_std = np.array([np.std(v) for k,v in sorted(k_to_accuracies.items())])
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()
```



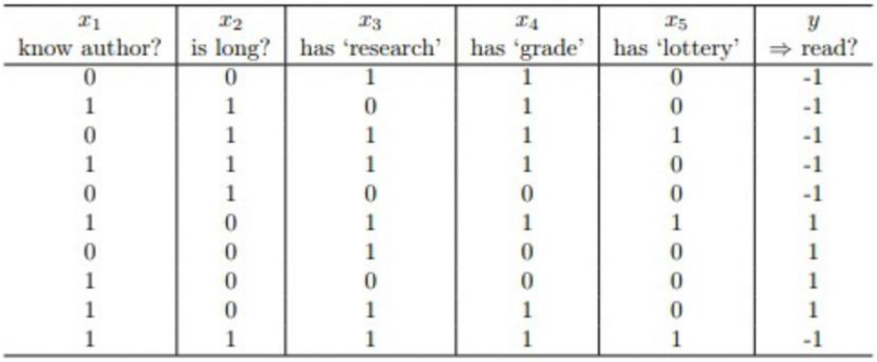
In [19]:
```
# Based on the cross-validation results above, choose the best value for k,
# retrain the classifier using all the training data, and test it on the test
# data. You should be able to get above 28% accuracy on the test data.
best_k = k_choices[np.argmax(accuracies_mean)]

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, accuracy))
```

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

Problem 3: Naïve Bayes Classifiers

HW1 Q3

1)



x_1 know author?	x_2 is long?	x_3 has 'research'	x_4 has 'grade'	x_5 has 'lottery'	y ⇒ 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

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

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

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

$P(x_1=0 \mid y=-1) = \frac{3}{6} = \frac{1}{2}$

$P(x_1=1 \mid y=1) = \frac{3}{4}$

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

$P(x_2=0 \mid y=1) = 1$

$P(x_2=0 \mid y=-1) = \frac{1}{6}$

$P(x_2=1 \mid y=1) = 0$

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

$P(x_3=0 \mid y=1) = \frac{1}{4}$

$P(x_3=0 \mid y=-1) = \frac{2}{6} = \frac{1}{3}$

$P(x_3=1 \mid y=1) = \frac{3}{4}$

$P(x_3=1 \mid y=-1) = \frac{4}{6} = \frac{2}{3}$

$P(x_4=0 \mid y=1) = \frac{2}{4} = \frac{1}{2}$

$P(x_4=0 \mid y=-1) = \frac{1}{6}$

$P(x_4=1 \mid y=1) = \frac{2}{4} = \frac{1}{2}$

$P(x_4=1 \mid y=-1) = \frac{5}{6}$

$P(x_5=0 \mid y=1) = \frac{3}{4}$

$P(x_5=0 \mid y=-1) = \frac{4}{6} = \frac{2}{3}$

$P(x_5=1 \mid y=1) = \frac{1}{4}$

$P(x_5=1 \mid y=-1) = \frac{2}{6} = \frac{1}{3}$

2) $P(x=0,0,0,0,0|y=1)=$

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

$= \frac{1}{4} \cdot 1 \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{3}{4} = \frac{3}{128}$

$P(x=0,0,0,0,0|y=-1)=$

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

$= \frac{1}{2} \cdot \frac{1}{6} \cdot \frac{1}{3} \cdot \frac{1}{6} \cdot \frac{2}{3} = \frac{1}{324}$

$P(x=0,0,0,0,0)=$

$=P(x=0,0,0,0,0|y=1)P(y=1) + P(x=0,0,0,0,0|y=-1)P(y=-1)$

$= \frac{3}{128}\left(\frac{2}{5}\right) + \left(\frac{1}{324}\right)\left(\frac{3}{5}\right) = \frac{3}{320} + \frac{1}{540}$

a) $P(y=1|x=0,0,0,0,0)$

$$=\frac{P(x=0,0,0,0,0|y=1)P(y=1)}{P(x=0,0,0,0,0)} = \frac{\frac{3}{128}\left(\frac{2}{5}\right)}{\frac{3}{320}+\frac{1}{540}} \approx .8351$$

$P(y=-1|x=0,0,0,0,0)$

$$=\frac{P(x=0,0,0,0,0|y=-1)P(y=-1)}{P(x=0,0,0,0,0)} = \frac{\frac{1}{324}\left(\frac{3}{5}\right)}{\frac{3}{320}+\frac{1}{540}} \approx .1649$$

Therefore the prediction for $x=0,0,0,0,0$ is $y=1$, so class = read

$P(x=1,1,0,1,0|y=1)=$

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

$= \frac{3}{4} \cdot 0 \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{3}{4} = 0$

$P(x=1,1,0,1,0|y=-1)=$

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

$= \frac{1}{2} \cdot \frac{5}{6} \cdot \frac{1}{3} \cdot \frac{5}{6} \cdot \frac{2}{3} = \frac{25}{324}$

$P(x=1,1,0,1,0)=$

$=P(x=1,1,0,1,0|y=1)P(y=1) + P(x=1,1,0,1,0|y=-1)P(y=-1)$

$= 0\left(\frac{2}{5}\right)+\left(\frac{25}{324}\right)\left(\frac{3}{5}\right) = \frac{5}{108}$

b) $P(y=1|x=1,1,0,1,0)$

$$= \frac{P(x=1,1,0,1,0|y=1)\,P(y=1)}{P(x=1,1,0,1,0)} = \frac{0\left(\frac{2}{5}\right)}{\left(\frac{5}{108}\right)} = 0$$

$P(y=-1|x=1,1,0,1,0)$

$$= \frac{P(x=1,1,0,1,0|y=-1)P(y=-1)}{P(x=1,1,0,1,0)} = \frac{\frac{25}{324}\left(\frac{3}{5}\right)}{\left(\frac{5}{108}\right)} = 1$$

Therefore the prediction for $x=1,1,0,1,0$ is $y=-1$, so class = discard

3) $P(x=1,1,0,1,0|y=1)=$

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

$= \frac{3}{4}\cdot 0\cdot\frac{1}{4}\cdot\frac{1}{2}\cdot\frac{3}{4}=0$

$P(x=1,1,0,1,0|y=-1)=$

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

$=\frac{1}{2}\cdot\frac{5}{6}\cdot\frac{1}{3}\cdot\frac{5}{6}\cdot\frac{2}{3}=\frac{25}{324}$

$P(x=1,1,0,1,0)=$

$=P(x=1,1,0,1,0|y=1)P(y=1)+P(x=1,1,0,1,0|y=-1)P(y=-1)$

$=0\left(\frac{2}{5}\right)+\left(\frac{25}{324}\right)\left(\frac{3}{5}\right)=\frac{5}{108}$

$P(y=1|x=1,1,0,1,0)$

$=\frac{P(x=1,1,0,1,0|y=1)\,P(y=1)}{P(x=1,1,0,1,0)}=\frac{0\left(\frac{2}{5}\right)}{\left(\frac{5}{108}\right)}=0$

So $P(y=1\mid x=1,1,0,1,0)=0$

4) For a "joint" Bayes classifier we would have to count the number of occurences and divide by total number of datapoints. But if we have no occurnce then it would be 0, but by using naive we rely on each individual featre appearing instead of appearing together. So with joint Bayes, the probability can be 0 in cases that do not appeur, but with naive those cases can be represented with actual values. So we want to use naive Bayes classifier instead of joint Bayes classifier because data points that do not appeur in given data can still be accounted for.

5) We should retrain the data to only use the other 4 features ($x_2 \dots x_5$) because it may change predictions and the total possible data sets decreases by factor of 2, so 2^4 instead of 2^5.

Statement of Collaboration

I, Andy Tran, did this assignment by myself.