Lab 5: Clustering and Linear Regression

Name: Matthew Wei (mswei2)

Due: October 9, 2017 11:59 PM

Logistics ¶

See Canvas

What You Will Need to Know For This Lab

- K-means clustering
- Vector Quantization
- Nearest Neighbors Classification
- Linear Regression

Preamble (don't change this)

In [1]:
```
%pylab inline
import numpy as np
from sklearn import neighbors
from numpy import genfromtxt
import scipy.spatial.distance as dist
import random
from sklearn.cluster import KMeans
from PIL import Image
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
%pylab is deprecated, use %matplotlib inline and import the required librarie
s.
Populating the interactive namespace from numpy and matplotlib
```

Problem 1: Selecting the number of clusters (30 points)

Write a function which implements K-means clustering.

You will be given as input:

- A (N, d) numpy.ndarray of unlabeled data (with each row as a feature vector), data
- A scalar K which indicates the number of clusters
- A scalar representing the number of iterations, niter (this is your stopping criterion/criterion for convergence)

Your output will be a tuple consisting of a vector of length N containing which cluster ($0, \ldots, K-1$) a feature vector is in and a (K, d) matrix with the rows containing the cluster centers.

Do not use scikit-learn or similar for implement K-means clustering. You may use `scipy.spatial.distance.cdist` to calculate distances. Initialize the centers randomly without replacement with points from the data set. `random.sample` may be useful for this. **(10 points)**

In [42]:
```python
def kMeans(data,K,niter):
    #Put your code here
    init = data[np.random.choice(data.shape[0], K, replace=False),:]
    centers = init
    for i in range(niter):
        distance = dist.cdist(data,centers,'euclidean')
        labels = np.argmin(distance, axis=1)
        for j in range(K):
            centers[j,:] = data[labels==j].mean(axis=0)
    return (labels, centers)
```

The K-means clustering problem tries to minimize the following quantity by selecting $\{z_i\}_{i=1}^N$ and $\{\mu_k\}_{k=1}^K$:

$$J_K(\{z_i\}_{i=1}^N, \{\mu_k\}_{k=1}^K) = \sum_{i=1}^N \|\mathbf{x}_i - \mu_{z_i}\|^2$$

where μ_{z_i} is the center of the cluster to which $\mathbf{x}_i$ is assigned.

One visual heuristic to choose the number of clusters from the data (where the number of clusters is not known a priori) is to estimate the optimal value of $J_K(\{z_i\}_{i=1}^N, \{\mu_k\}_{k=1}^K)$, $J^*(K)$, for different values of K and look for an "elbow" or "knee" in the curve of J^* versus K and choose that value of K.

In this part of the problem, you will run K-means 100 times for each $K = 2, \ldots, 10$ and calculate $J_K(\{z_i\}_{i=1}^N, \{\mu_k\}_{k=1}^K)$ for the clustering given by K-means. Use the smallest value of $J_K(\{z_i\}_{i=1}^N, \{\mu_k\}_{k=1}^K)$ in the runs of K-means for each value of K to form an estimate of $J^*(K)$. Plot this estimate versus K. Which K should you pick by this heuristic? Use niter=100 for each run of K-means.

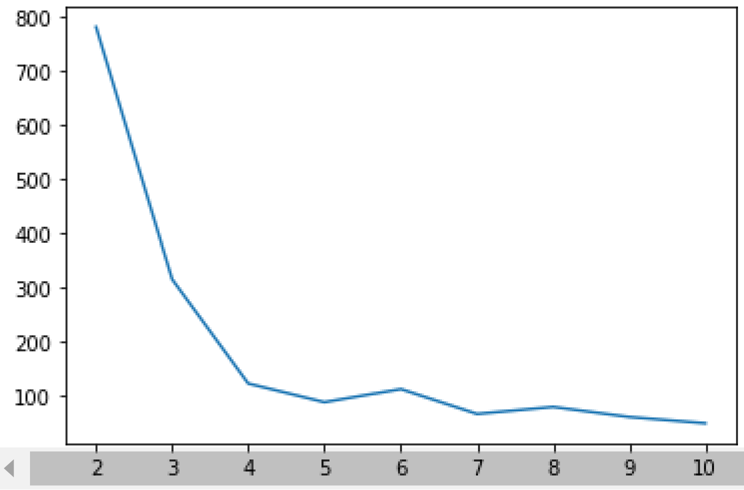
For an attempt to formalize this heuristic, see Tibshirani, Robert, Guenther Walther, and Trevor Hastie. "Estimating the number of clusters in a data set via the gap statistic." Journal of the Royal Statistical Society: Series B (Statistical Methodology) 63.2 (2001): 411-423. Sometimes, an elbow does not exist in the curve or there are multiple elbows or the K value of an elbow cannot be unambiguously identified. Further material can be found on Wikipedia (http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set#The_Elbow_M as well.

Note: Your code should be relatively quick -- a few minutes, at worst. **(10 points)**

```
In [43]: # Load up some data, which we will store in a variable called problem1
         problem1= genfromtxt('problem1.csv', delimiter=',')
```

```
In [46]: # Put your code here
         niter = 100
         jk = np.zeros(9)
         for K in range(2,11):
             (labels,centers) = kMeans(problem1, K, niter)
             for j in range(K):
                 jk[K-2]+=((problem1[labels==j]-centers[j])**2).sum()
         plot(range(2,11), jk)
```

```
Out[46]: [<matplotlib.lines.Line2D at 0x2e04c9d11e0>]
```

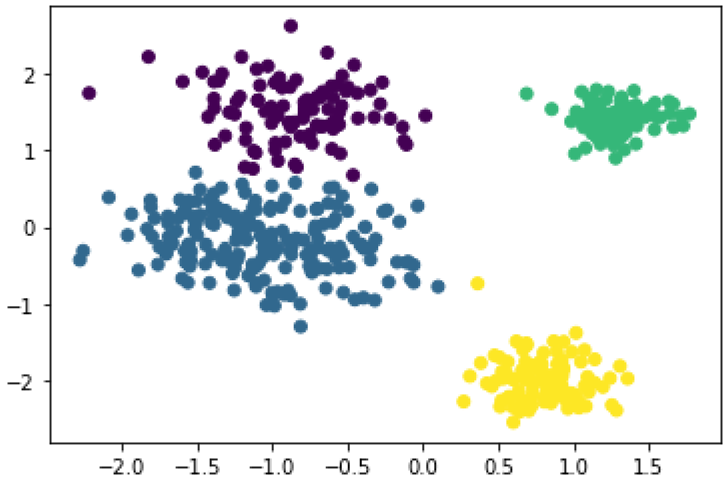


The "elbow" or "knee" seems to occur at K = 4.

Using the value of K you determined from the elbow, perform K-means clustering on the data. Plot it as a scatter plot with the colors given by the labels. **(5 points)**

In [47]:
```
#Put your code here
labels, centers = kMeans(problem1, 4, 100)
scatter(problem1[:,0], problem1[:,1], c=labels)
```

Out[47]:
```
<matplotlib.collections.PathCollection at 0x2e04ca384f0>
```



Should you pick the K such that $J^*(K)$ is minimized? Why or why not? **(5 points)**

No becuase in this case as K increases, J(K) will also decrease. Realisitcally, this will result in more clusters. Having more clusters may end up categorizing data incorrectly by having too many groups than needed. This is one reason why K should not be picked based on J(K) minimization.

Problem 2: Vector Quantization (40 points)

In this problem, you will implement vector quantization. You will use `sklearn.cluster.KMeans` for the K-means implementation and use k-means++ as the initialization method.

Write a function to generate a codebook for vector quantization. You will be given inputs:

- A (N, M) numpy.ndarray representing a greyscale image, called image. (If we want to generate our codebook from multiple images, we can concatenate the images before running them through this function).
- A scalar B, for which you will use $B \times B$ blocks for vector quantization. You may assume N and M are divisible by B.
- A scalar K, which is the size of your codebook

You will return:

- The codebook as a (K, B^2) numpy.ndarray. **(10 points)**

In [59]:
```python
def trainVQ(image,B,K):
    # Put your code here
    N, M = image.shape
    img = np.zeros((N*M//B**2,B**2))
    matrix = (image.reshape(N//B, B, -1, B).swapaxes(1,2).reshape(-1, B, B))
    for i in range(int((N*M)/(B*B))):
        img[i] = matrix[i].flatten()
    kmeans = KMeans(n_clusters=K,init='k-means++').fit(img)
    codebook = kmeans.cluster_centers_
    return (codebook, kmeans)
```

Write a function which compresses an image against a given codebook. You will be given inputs:

- A (N, M) numpy.ndarray representing a greyscale image, called image. You may assume N and M are divisible by B.
- A (K, B^2) codebook called codebook
- B

You will return:

- A $(N/B, M/B)$ numpy.ndarray consisting of the indices in the codebook used to approximate the image.

You can use the nearest neighbor classifier from scikit-learn if you want (though it is not necessary) to map blocks to their nearest codeword. **(10 points)**

In [60]:
```python
from sklearn.neighbors import NearestNeighbors

def compressImg(image, codebook,B):
    #Put your code here
    classifier = NearestNeighbors(n_neighbors=1, algorithm='auto')
    classifier.fit(codebook)
    N, M = image.shape
    matrix = np.zeros((N*M//B**2,B**2))
    for i in range(N):
        for j in range (M):
            matrix[i//B*M//B+j//B][i%B*B+j%B]=image[i][j]
    idx = classifier.kneighbors(matrix, return_distance=False)
    return np.resize(idx,(N//B,M//B))
```

Write a function to reconstruct an image from its codebook. You will be given inputs:

- A $(N/B, M/B)$ numpy.ndarray containing the indices of the codebook for each block called indices
- A codebook as a (K, B^2) numpy.ndarray called codebook
- B

You will return a (N, M) numpy.ndarray representing the image. **(10 points)**

In [61]:
```
def decompressImg(indices, codebook,B):
    #Put your code here
    N ,M = indices.shape
    image = np.zeros((N*B,M*B))
    for i in range(N):
        for j in range(M):
            image[i*B:(i+1)*B,j*B:(j+1)*B]=codebook[indices[i][j]].reshape(B,E
    return image
```

Run your vector quantizer with 5×5 blocks on the provided image with codebook sizes $K = 2, 5, 10, 20, 50, 100, 200$ (i.e. generate codebooks from this image of those sizes, compress the image using those codebooks and reconstruct the images). Display and comment on the reconstructed images (you may be quantitative (e.g. PSNR) or qualitative). Which code book would you pick? Why? Make sure to take into account the bits per pixel used by the compressor.

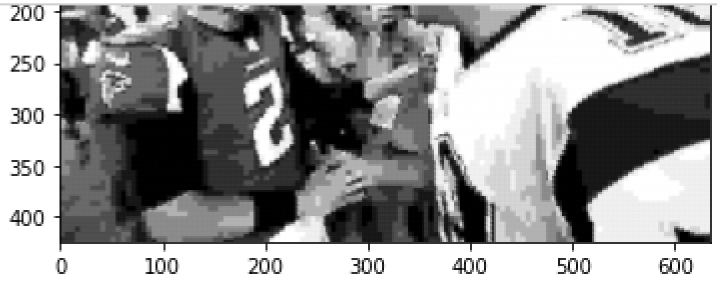
Note the number of bits per pixel can be approximated as $\frac{\log_2 K}{25}$ and the codebook takes approximately $200K$ bits (assuming each pixel is stored as 8 bits). Some good ideas on quantitative arguments for codebook size can be found in Gonzalez & Woods, Digital Image Processing 3e or Gersho & Gray, Signal Compression & Vector Quantization. It is not necessary to look at these references for quantitative arguments, though. **(10 points)**

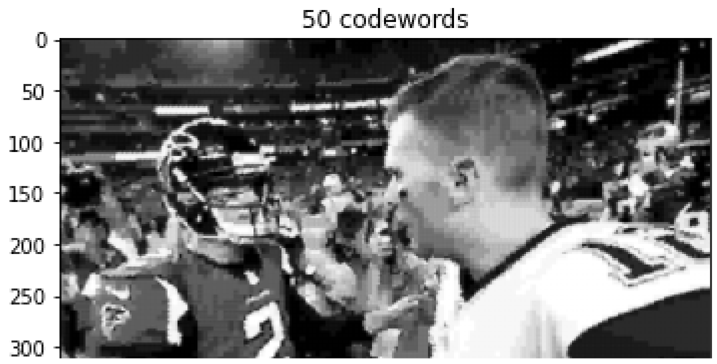
The image used is under fair use from Bleacher Report (http://bleacherreport.com/articles/2688697-tom-brady-comments-on-friendship-with-matt-ryan-ahead-of-super-bowl-51).

In [62]:
```
# The provided image is stored in image
image = np.asarray(Image.open("mrtb.jpg").convert("L"))
imshow(image, cmap = cm.Greys_r)
```

Out[62]:
```
<matplotlib.image.AxesImage at 0x2e047a40790>
```

In [63]:
```python
import warnings
warnings.filterwarnings("ignore")

#Put your code here
B = 5
for K in [2,5,10,20,50,100,200]:
    codebook,_ = trainVQ(image,B,K)
    cmpimg = compressImg(image,codebook,B)
    dcpimg = decompressImg(cmpimg,codebook,B)
    plt.figure()
    plt.title("{} codewords".format(K))
    imshow(dcpimg, cmap = cm.Greys_r)
```





I would pick the 200 codebook because out of all the images, the 200 codebook output looks the best. It also contains the most number of bits per pixel at roughly 0.30575424759 bits per pixel.

Problem 3: Linear Regression (35 points)

In this problem, you will do model selection for linear regression using Ordinary Least Squares, Ridge Regression and the LASSO.

The dataset you will use has 8 features:

```
lcavol - log cancer volume
lcaweight - log prostate weight
age
lbph - log of amount of benign prostatic hyperplasia
svi - seminal vesicle invasion
lcp - log capsular penetration
gleason - Gleason score
pgg45 - percent of Gleason scores 4 or 5
```

and you will predict the level of a prostate-specific antigen. The data set was collected from a set of men about to receive a radical prostatectomy. More details about this dataset are given in Section 3.2.1 in Elements of Statistical Learning 2e by Hastie et al.

In [13]:
```
# Load the data
trainp= genfromtxt('trainp.csv', delimiter=',')

# Training data:
trainfeat=trainp[:,:-1] #Training features (rows are feature vectors)
trainresp=trainp[:,-1] #Training responses

valp= genfromtxt('valp.csv',delimiter=',')
# Validation data:
valfeat=valp[:,:-1] #Validation Features (rows are feature vectors)
valresp=valp[:,-1] #Validation Response

# Standardize and center the features
ftsclr=StandardScaler()
trainfeat = ftsclr.fit_transform(trainfeat)
valfeat= ftsclr.transform(valfeat)
# and the responses
rsclr=StandardScaler()
trainresp = (rsclr.fit_transform(trainresp.reshape(-1,1))).reshape(-1)
valresp= (rsclr.transform(valresp.reshape(-1,1))).reshape(-1)

# The training features are in trainfeat
# The training responses are in trainresp
# The validation features are in valfeat
# The validation responses are in valresp
```

Since we centered the responses, we can begin with a benchmark model: Always predict the response as zero (mean response on the training data). Calculate the validation RSS for this model. **(5 points)**

If another model does worse than this, it is a sign that something is amiss.

Note: The RSS on a data set with V samples is given by $\frac{1}{V}||\mathbf{y}-\hat{\mathbf{y}}||^2$ where $\mathbf{y}$ is a vector of the responses, and $\hat{\mathbf{y}}$ is the predicted responses on the data.

In [38]:
```
# Put your code here
rss = np.linalg.norm(0-valresp)**2.0/len(valresp)
print("RSS for benchmark model:", rss)
```

```
RSS for benchmark model: 0.7338520919126769
```

0.7338520919126769

First, you will try (Ordinary) Least Squares. Use `sklearn.linear_model.LinearRegression` with the default options. Calculate the validation RSS. **(5 points)**

Note: The .score() method returns an R^2 value (https://en.wikipedia.org/wiki/Coefficient_of_determination), not the RSS, so you shouldn't use it

```
In [64]: # Put your code here
         model = linear_model.LinearRegression()
         model.fit(trainfeat, trainresp)
         pred = model.predict(valfeat)
         rss = np.linalg.norm(pred-valresp)**2.0/len(valresp)
         print("RSS for Linear Regression model:", rss)
```

```
RSS for Linear Regression model: 0.3623070990381976
```

0.3623070990381976

Now, you will apply ridge regression with `sklearn.linear_model.Ridge` .

Sweep the regularization/tuning parameter $\alpha = 0, \ldots, 100$ with 1000 equally spaced values.

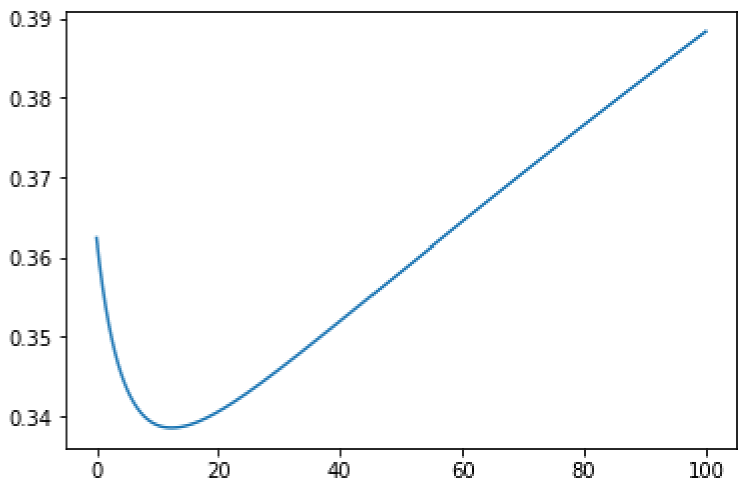
Make a plot of the RSS on the validation set versus α. What is the minimizing α, corresponding coefficients and validation error?

Larger values of α shrink the weights in the model more. $\alpha = 0$ corresponds to the LS solution. **(10 points)**

In [67]:
```
# Put your code here
a = np.linspace(0,100,1000)
rss_array = np.zeros(a.shape)
for i in range(len(a)):
    model = linear_model.Ridge(alpha=a[i])
    model.fit(trainfeat, trainresp)
    pred = model.predict(valfeat)
    rss_array[i] = np.linalg.norm(pred-valresp)**2/len(valresp)

cls = np.argmin(rss_array)
alpha = a[cls]
rss = rss_array[cls]
best = linear_model.Ridge(alpha=best_a)
best.fit(trainfeat, trainresp)
coef = best_model.coef_

print('Minimizing alpha: %s' % alpha)
print("Corresponding coefficients:", coef)
print("RSS: %s" % rss)
a = np.linspace(0,100,1000)
plot(a, rss_array)
```

```
Minimizing alpha: 12.312312312312313
Corresponding coefficients: [ 0.47490839  0.18706547 -0.          0.07857742
0.13591787  0.
  0.          0.04868055]
RSS: 0.3384853327816165
```

Out[67]:
```
[<matplotlib.lines.Line2D at 0x2e04ee76ec0>]
```



Minimizing α: 12.312312312312313
Corresponding coefficients: 0.42874536, 0.22631082, -0.06455834, 0.15539095, 0.21579184, -0.05436134, 0.02588654, 0.13447399
Validation Error: 0.3384853327816165

Now, you will apply the LASSO with `sklearn.linear_model.Lasso` .

Sweep the tuning/regularization parameter $\alpha = 0, \ldots, 1$ with 1000 equally spaced values.

Make a plot of the RSS on the validation set versus α. What is the minimizing α, corresponding coefficients and validation error?

Larger values of α lead to sparser solutions (i.e. less features used in the model), with a sufficiently large value of α leading to a constant prediction. Small values of α are closer to the LS solution, with $\alpha = 0$ being the LS solution. **(10 points)**

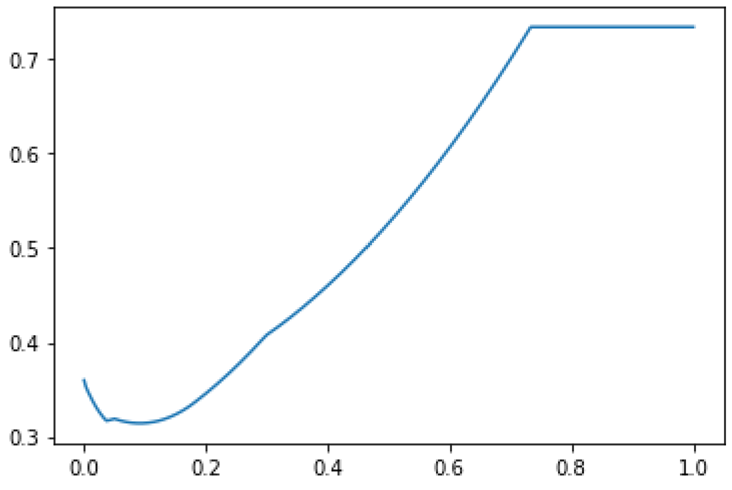
In [68]:

```python
# Put your code here
a = np.linspace(1,0,1000,endpoint=False)[::-1]
rss_array = np.zeros(a.shape)
for i in range(len(a)):
    model = linear_model.Lasso(alpha=a[i])
    model.fit(trainfeat, trainresp)
    pred = model.predict(valfeat)
    rss_array[i] = np.linalg.norm(pred-valresp)**2/len(valresp)

cls = np.argmin(rss_array)
alpha = a[cls]
rss = rss_array[cls]
best = linear_model.Lasso(alpha=best_a)
best.fit(trainfeat, trainresp)
coef = best_model.coef_

print('Minimizing alpha: %s' % alpha)
print("Corresponding coefficients:", coef)
print("RSS: %s" % rss)
a = np.linspace(1,0,1000,endpoint=False)[::-1]
plot(a, rss_array)
```

```
Minimizing alpha: 0.09299999999999997
Corresponding coefficients: [ 0.47490839  0.18706547 -0.          0.07857742
0.13591787  0.
  0.          0.04868055]
RSS: 0.31426491652065836
```

Out[68]:

```
[<matplotlib.lines.Line2D at 0x2e04eeddea0>]
```



Minimizing α: 0.09299999999999997
Corresponding coefficients: 0.47490839, 0.18706547, 0, 0.07857742, 0.13591787, 0, 0, 0.04868055

Which features were selected by Ridge Regression when minimizing the RSS on the validation set? Which features were selected by LASSO when minimizing the RSS on the validation set? Which model would you choose (and why)? **(5 points)**

RSS had age, lcp, and gleason selected (minimized the most) and LASSO also had the same features selected (minimized to 0). If I had to chose a model, I would choose LASSO due to it having a smaller RSS value.