Problem 1: PCA and Feature Selection

SVMs and PCA

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import pandas as pd
import cvxopt
from tqdm.notebook import trange
from sklearn.svm import SVC
```

In [2]:
```
sonar_train = pd.read_csv('sonar_train.data', header=None)
sonar_test = pd.read_csv('sonar_test.data', header=None)
sonar_valid = pd.read_csv('sonar_valid.data', header=None)

sonar_train.loc[sonar_train[60] == 2, 60] = -1
sonar_test.loc[sonar_test[60] == 2, 60] = -1
sonar_valid.loc[sonar_valid[60] == 2, 60] = -1

def normalize(data, mean, std):
    return (data - mean) / std

def split_data(data):
    return data.iloc[:, :60].to_numpy(), data.iloc[:, 60:].to_numpy()

X, y_train = split_data(sonar_train)
train_mean = X.mean(axis=0)
train_std = X.std(axis=0)

X_train = normalize(X, train_mean, train_std)
X, y_validation = split_data(sonar_valid)
X_validation = normalize(X, train_mean, train_std)
X, y_test = split_data(sonar_test)
X_test = normalize(X, train_mean, train_std)
```

Perform PCA on the training data to reduce the dimensionality of the data set (ignoring the class labels for the moment). What are the top six eigenvalues of the data covariance matrix?

In [3]:
```
def compute_covariance(norm_data):
    # Covariance matrix has dimensions (p x p)
    # Usually computed with variables as rows and observations as columns (np.
cov)
    return norm_data.T.dot(norm_data)

covariance_mat = compute_covariance(X_train) #np.cov(X_train.T)
```

In [4]:
```
e_val, e_vec = np.linalg.eig(covariance_mat)
i_rev = e_val.argsort()[::-1]
eig_vals = e_val[i_rev]
eig_vecs = e_vec[:, i_rev]
print("Top 6 Eigen values", eig_vals[:6])
```

```
Top 6 Eigen values [1344.02076581 1196.11970978  541.52327862  351.33131561
313.13028781
  267.04094634]
```

For each k ∈ {1, 2, 3, 4, 5, 6}, project the training data into the best k dimensional subspace (with respect to the Frobenius norm) and use the SVM with slack formulation to learn a classifier for each c ∈ {1, 10, 100, 1000}. Report the error of the learned classifier on the validation set for each k and c pair.

In [5]:
```
C = [1, 10, 100, 1000]
```

In [6]:
```
data = {
    'k': [],
    'c': [],
    'Training Data Error': [],
    'Validation Data Error': []
}
for k in range(1,7):
    U = eig_vecs[:,:k]
    X_proj = X_train.dot(U)
    X_valid_proj = X_validation.dot(U)
    for c in C:
        data['k'].append(k)
        data['c'].append(c)
        clf = SVC(C=c, kernel='linear')
        clf.fit(X_proj, y_train.ravel())
        y_pred = clf.predict(X_proj)
        data['Training Data Error'].append(1 - np.mean(y_pred == y_train.ravel
()))
        valid_pred = clf.predict(X_valid_proj)
        data['Validation Data Error'].append(1 - np.mean(valid_pred == y_valid
ation.ravel()))
```

```
In [7]: pd.DataFrame(data)
```

```
Out[7]:
```

	k	c	Training Data Error	Validation Data Error
0	1	1	0.509615	0.461538
1	1	10	0.509615	0.461538
2	1	100	0.509615	0.461538
3	1	1000	0.509615	0.461538
4	2	1	0.432692	0.307692
5	2	10	0.432692	0.307692
6	2	100	0.432692	0.307692
7	2	1000	0.432692	0.307692
8	3	1	0.269231	0.211538
9	3	10	0.269231	0.211538
10	3	100	0.269231	0.211538
11	3	1000	0.278846	0.211538
12	4	1	0.288462	0.211538
13	4	10	0.288462	0.211538
14	4	100	0.288462	0.211538
15	4	1000	0.288462	0.211538
16	5	1	0.269231	0.250000
17	5	10	0.269231	0.250000
18	5	100	0.269231	0.250000
19	5	1000	0.269231	0.250000
20	6	1	0.240385	0.269231
21	6	10	0.240385	0.269231
22	6	100	0.240385	0.269231
23	6	1000	0.240385	0.269231

How does it compare to the best classifier (with the same possible c choices) without feature selection?

In [8]:
```
data = {
    'c': [],
    'Training Data Error': [],
    'Validation Data Error': []
}
for c in C:
    data['c'].append(c)
    clf = SVC(C=c, kernel='linear', random_state=0)
    clf.fit(X_train, y_train.ravel())
    y_pred = clf.predict(X_train)
    data['Training Data Error'].append(1 - np.mean(y_pred == y_train.ravel()))
    valid_pred = clf.predict(X_validation)
    data['Validation Data Error'].append(1 - np.mean(valid_pred == y_validatio
n.ravel()))
pd.DataFrame(data)
```

Out[8]:

	c	Training Data Error	Validation Data Error
0	1	0.009615	0.211538
1	10	0.000000	0.230769
2	100	0.000000	0.230769
3	1000	0.000000	0.230769

What is the error of the best k/c pair on the test data? How does it compare to the best classifier (with the same possible c choices) without feature selection? Explain your observations.

Best k = 3 and c = 1, 10, 100, 1000

In [9]:
```
U = eig_vecs[:,:3]
X_proj = X_train.dot(U)
X_test_proj = X_test.dot(U)
res = {
    'c': [],
    'Test data error': []
}
for c in [1, 10, 100, 1000]:
    res['c'].append(c)
    clf = SVC(C=c, kernel='linear', random_state=0)
    clf.fit(X_proj, y_train.ravel())
    res['Test data error'].append(1 - np.mean(clf.predict(X_test_proj) == y_te
st.ravel()))
pd.DataFrame.from_dict(res)
```

Out[9]:

	c	Test data error
0	1	0.192308
1	10	0.192308
2	100	0.192308
3	1000	0.192308

In [10]:
```python
# SVM without feature selection
res = {
    'c': [],
    'Test data error': []
}
for c in [1]:
    res['c'].append(c)
    clf = SVC(C=c, kernel='linear', random_state=0)
    clf.fit(X_train, y_train.ravel())
    res['Test data error'].append(1 - np.mean(clf.predict(X_test) == y_test.ra
vel()))
pd.DataFrame.from_dict(res)
```

Out[10]:

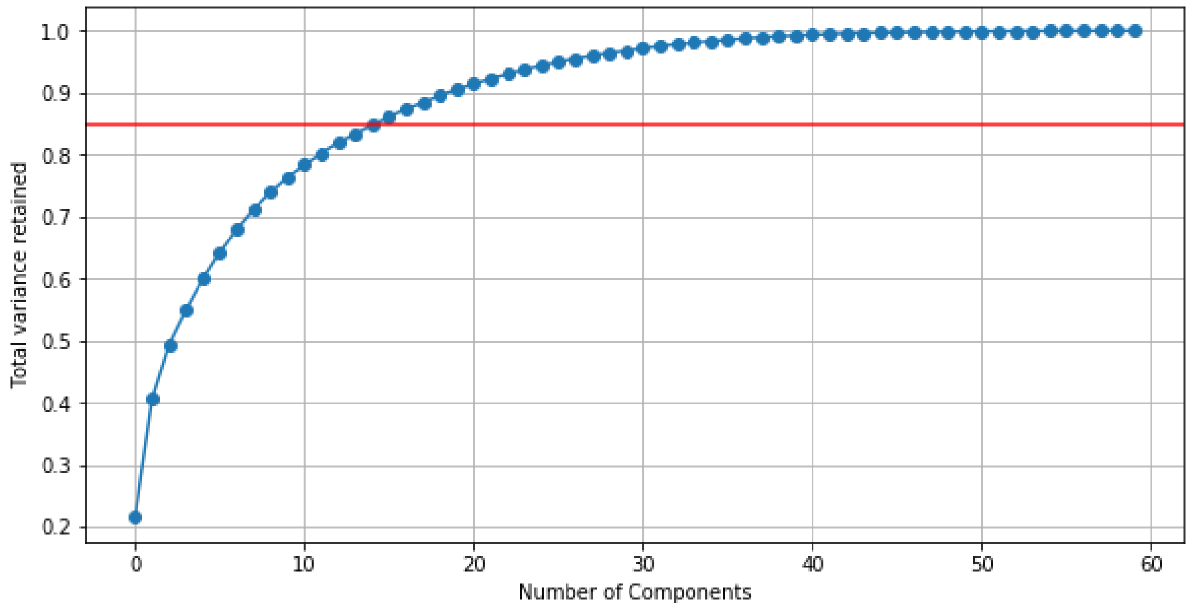
	c	Test data error
0	1	0.211538

SVM on reduced dimension data performs better on test data compared to SVM without any feature selection

If you had to pick a value of k before evaluating the performance on the validation set (e.g., if this was not a supervised learning problem), how might you pick it?

Value of k or the number of components could be picked by heuristics. We can pick the number of components needed to explain at least 85% of the data variance. For this dataset, we can pick k = 14.

In [11]:
```python
fig = plt.figure(figsize=(10, 5))
plt.plot(np.cumsum(eig_vals) / np.sum(eig_vals), marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Total variance retained')
plt.axhline(y=0.85, color='r', linestyle='-')
plt.grid()
```



PCA for Feature Selection

1. Compute the top k eigenvalues and eigenvectors of the covariance matrix corresponding to the data matrix omitting the labels (recall that the rows of the data matrix are the input data points).

2. Define π

3. Sample s columns independently from the probability distribution defined by π.

In [12]:
```
def select_k_features(features, s, pi):
    sel_features = np.random.choice(features, s, p=pi, replace=False)
    return sel_features
```

Why does π define a probability distribution?

π has values ranging between 0 and 1. Also sum of all values in π = 1.

Again, using the UCI Sonar data set, for each k ∈ {1, . . . , 10} and each s ∈ {1, . . . , 20}, report the average test error of the SVM with slack classifier over 100 experiments. For each experiment use only the s selected features (note that there may be some duplicates, so only include each feature once).

In [13]:
```
data = {
    'k': [],
    's': [],
    'Average test error': []
}
datapoints, features = X_train.shape
for k in trange(1,11):
    V = eig_vecs[:k]
    pi = np.sum(V**2, axis=0) / k
    for s in trange(1,21):
        error = []
        for _ in range(100):
            sel_features = select_k_features(features, s, pi)
            clf = SVC(C=1, kernel='linear', random_state=0)
            clf.fit(X_train[:,sel_features], y_train.ravel())
            y_pred = clf.predict(X_test[:,sel_features])
            error.append(1 - np.mean(y_pred==y_test.ravel()))
        mean_error = np.mean(error)
        data['k'].append(k)
        data['s'].append(s)
        data['Average test error'].append(mean_error)
res = pd.DataFrame.from_dict(data)
```

In [16]:
```
pd.set_option("display.max_rows", None, "display.max_columns", None)
res
```

Out[16]:

	k	s	Average test error
0	1	1	0.460769
1	1	2	0.401154
2	1	3	0.378269
3	1	4	0.367692
4	1	5	0.350192
5	1	6	0.320769
6	1	7	0.318846
7	1	8	0.318077
8	1	9	0.305192
9	1	10	0.302115
10	1	11	0.291731
11	1	12	0.285962
12	1	13	0.292308
13	1	14	0.286154
14	1	15	0.285385
15	1	16	0.283654
16	1	17	0.281731
17	1	18	0.267115
18	1	19	0.268269
19	1	20	0.268269
20	2	1	0.435962
21	2	2	0.389038
22	2	3	0.342885
23	2	4	0.343269
24	2	5	0.311154
25	2	6	0.299808
26	2	7	0.296731
27	2	8	0.297308
28	2	9	0.280962
29	2	10	0.265385
30	2	11	0.267885
31	2	12	0.266731
32	2	13	0.258269
33	2	14	0.252500
34	2	15	0.259423

	k	s	Average test error
35	2	16	0.252885
36	2	17	0.240192
37	2	18	0.247500
38	2	19	0.247692
39	2	20	0.239423
40	3	1	0.428077
41	3	2	0.364231
42	3	3	0.347885
43	3	4	0.320769
44	3	5	0.293654
45	3	6	0.294423
46	3	7	0.282885
47	3	8	0.280769
48	3	9	0.265577
49	3	10	0.271154
50	3	11	0.264231
51	3	12	0.252692
52	3	13	0.258269
53	3	14	0.256731
54	3	15	0.248077
55	3	16	0.247692
56	3	17	0.247692
57	3	18	0.259423
58	3	19	0.242500
59	3	20	0.253077
60	4	1	0.430385
61	4	2	0.376346
62	4	3	0.331923
63	4	4	0.323077
64	4	5	0.305192
65	4	6	0.281346
66	4	7	0.281731
67	4	8	0.280385
68	4	9	0.273654
69	4	10	0.257308
70	4	11	0.256346

	k	s	Average test error
71	4	12	0.258654
72	4	13	0.263077
73	4	14	0.255385
74	4	15	0.258654
75	4	16	0.247692
76	4	17	0.250385
77	4	18	0.245769
78	4	19	0.250962
79	4	20	0.247115
80	5	1	0.434615
81	5	2	0.365577
82	5	3	0.345577
83	5	4	0.302115
84	5	5	0.295192
85	5	6	0.286731
86	5	7	0.283077
87	5	8	0.271538
88	5	9	0.266923
89	5	10	0.273269
90	5	11	0.265000
91	5	12	0.258269
92	5	13	0.255385
93	5	14	0.261538
94	5	15	0.250769
95	5	16	0.254808
96	5	17	0.240962
97	5	18	0.246731
98	5	19	0.242308
99	5	20	0.242500
100	6	1	0.427692
101	6	2	0.372308
102	6	3	0.341923
103	6	4	0.317500
104	6	5	0.294231
105	6	6	0.282308
106	6	7	0.282115

	k	s	Average test error
107	6	8	0.267500
108	6	9	0.270385
109	6	10	0.265385
110	6	11	0.260000
111	6	12	0.255577
112	6	13	0.248462
113	6	14	0.248654
114	6	15	0.244231
115	6	16	0.252308
116	6	17	0.252115
117	6	18	0.239231
118	6	19	0.239038
119	6	20	0.242885
120	7	1	0.430577
121	7	2	0.352885
122	7	3	0.344808
123	7	4	0.313654
124	7	5	0.289423
125	7	6	0.282885
126	7	7	0.274038
127	7	8	0.265577
128	7	9	0.260385
129	7	10	0.254808
130	7	11	0.258654
131	7	12	0.259038
132	7	13	0.259038
133	7	14	0.242885
134	7	15	0.250192
135	7	16	0.242500
136	7	17	0.243846
137	7	18	0.241538
138	7	19	0.244615
139	7	20	0.245192
140	8	1	0.428269
141	8	2	0.384038
142	8	3	0.336731

	k	s	Average test error
143	8	4	0.330000
144	8	5	0.301346
145	8	6	0.289038
146	8	7	0.269615
147	8	8	0.280385
148	8	9	0.274231
149	8	10	0.256731
150	8	11	0.258269
151	8	12	0.253846
152	8	13	0.262500
153	8	14	0.251346
154	8	15	0.245769
155	8	16	0.246154
156	8	17	0.248462
157	8	18	0.245192
158	8	19	0.236538
159	8	20	0.241154
160	9	1	0.415577
161	9	2	0.373846
162	9	3	0.336154
163	9	4	0.318462
164	9	5	0.297500
165	9	6	0.289615
166	9	7	0.287692
167	9	8	0.268846
168	9	9	0.275192
169	9	10	0.276731
170	9	11	0.251731
171	9	12	0.247115
172	9	13	0.261731
173	9	14	0.251346
174	9	15	0.248846
175	9	16	0.250192
176	9	17	0.234038
177	9	18	0.245769
178	9	19	0.247308

	k	s	Average test error
179	9	20	0.252308
180	10	1	0.431731
181	10	2	0.366154
182	10	3	0.333269
183	10	4	0.310000
184	10	5	0.300192
185	10	6	0.286154
186	10	7	0.285769
187	10	8	0.266923
188	10	9	0.262115
189	10	10	0.256731
190	10	11	0.256731
191	10	12	0.264423
192	10	13	0.257885
193	10	14	0.242308
194	10	15	0.247692
195	10	16	0.242115
196	10	17	0.244038
197	10	18	0.235385
198	10	19	0.231154
199	10	20	0.231731

```
In [15]: res.loc[res['Average test error'] == res['Average test error'].min()]
```

Out[15]:

	k	s	Average test error
198	10	19	0.231154

Does this provide a reasonable alternative to SVM with slack formulation without feature selection on this data set? What are the pros and cons of this approach?

SVM with feature selection (k = 7, s = 20, c = 1) gives lowest average test error of 23% whereas SVM without feature selection (c = 1) gives lowest test error of 21%. There is not much improvement with feature selection.

Pros:

- Training individual models is computationally faster as number of features is less

Cons:

- Choosing k and s is difficult. Grid search like above takes long time

Problem 2: Spectral Clustering.

1. Compute the "Laplacian matrix", L = D−A, where D is a diagonal matrix with Dii = $\sum$ j Aij for all i. Argue that this matrix is positive semidefinite

```
In [1]: import numpy as np
        import pandas as pd
        from sklearn.cluster import KMeans
        import matplotlib.pyplot as plt
```

```
In [2]: def compute_laplacian(A):
            m, n = A.shape
            D = np.zeros((m, m))
            for i in range(m):
                D[i, i] = A[i].sum()
            return D - A
```

```
In [3]: def compute_similarity_matrix(X, sigma):
            shape = X.shape
            if len(shape) == 2:
                m, n = shape
            else:
                m = shape[0]
            K = np.zeros((m, m))
            X_sq = -2 * np.dot(X, X.T)
            X_sq = X_sq.astype(np.float64)
            X_sq += (X ** 2).sum(axis=1).reshape(-1, 1)
            X_sq += (X ** 2).sum(axis=1)
            K = X_sq / (-2 * (sigma ** 2))
            np.exp(K, K)
            return K
```

2. Compute the eigenvectors of the Laplacian using eig() in MATLAB (numpy in Python).

3. Construct a matrix V ∈ R n×k whose columns are the eigenvectors that correspond to the k smallest eigenvalues of L.

4. Let y1, . . . , yn denote the rows of V . Use the kmeans() algorithm in MATLAB (scikit-learn in Python) to cluster the rows of V into clusters S1, . . . , Sk.

5. The final clusters C1, . . . , Ck should be given by assigning vertex i of the input set to cluster Cj if yi ∈ Sj .

In [4]:
```python
def form_clusters(labels, X, K):
    clusters = {}
    for i in range(len(X)):
        clusters[labels[i]] = X[i]
    return clusters

def spectral_clustering(A, K=2):
    L  =  compute_laplacian(A)
    eigen_val, eigen_vectors = np.linalg.eigh(L)
    idx = eigen_val.argsort()[0:K]
    k_eigen_val = eigen_val[idx]
    V = eigen_vectors[:,idx]
    V = np.nan_to_num(V)
    spectral = KMeans(n_clusters=K).fit(V)
    labels = spectral.labels_
    clusters = form_clusters(labels, V, K)
    return clusters, labels
```

A Simple Comparison

1. Use the spectral clustering algorithm above to compute the clustering for the matrix of twodimensional points returned by the function circs() (attached to this problem set) above with k = 2 and different values of σ.

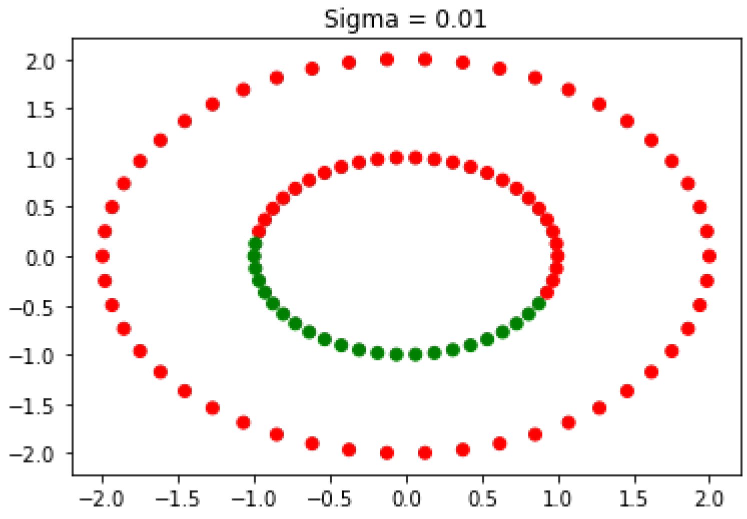
In [5]:
```
def circs():

    #X = zeros(2,100);
    #y = 0;
    #for i = 0:pi/25:2*pi
        #y = y + 1;
        #X(1, y) = cos(i);
        #X(2, y) = sin(i);
    #end
    #for i = 0:pi/25:2*pi
        #y = y + 1;
        #X(1, y) = 2*cos(i);
        #X(2, y) = 2*sin(i);
    #end


    X = np.zeros((2, 100))
    y = 0

    for i in np.arange(0, 2*np.pi, np.pi/25.0):
        X[0, y] = np.cos(i)
        X[1, y] = np.sin(i)
        y += 1

    for i in np.arange(0, 2*np.pi, np.pi/25.0):
        X[0, y] = 2*np.cos(i)
        X[1, y] = 2*np.sin(i)
        y += 1
    return X
```

In [6]:
```
X = circs().T
K = 2
```

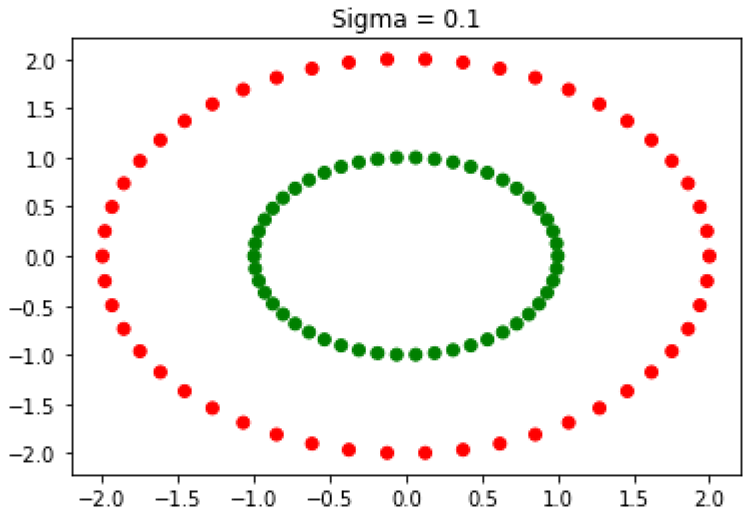
In [7]:
```
sigma_list = [0.01, 0.1, 1, 10, 100]
```

```
In [8]: for sigma in sigma_list:
            A = compute_similarity_matrix(X, sigma)
            clusters, labels = spectral_clustering(A, K)
            label_colors = ['r' if l else 'g' for l in labels]
            plt.title(f'Sigma = {sigma}')
            plt.scatter(X[:, 0], X[:, 1], c=label_colors)
            plt.show()
```

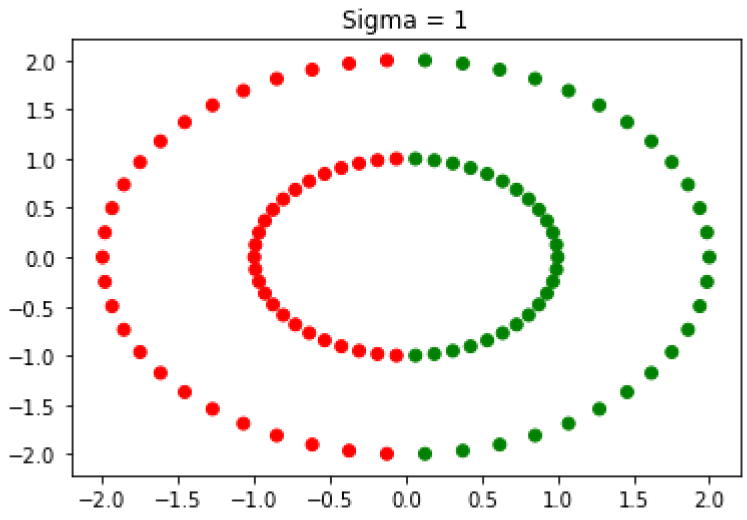
```
Laplacian Computed
Eigen values computed
```



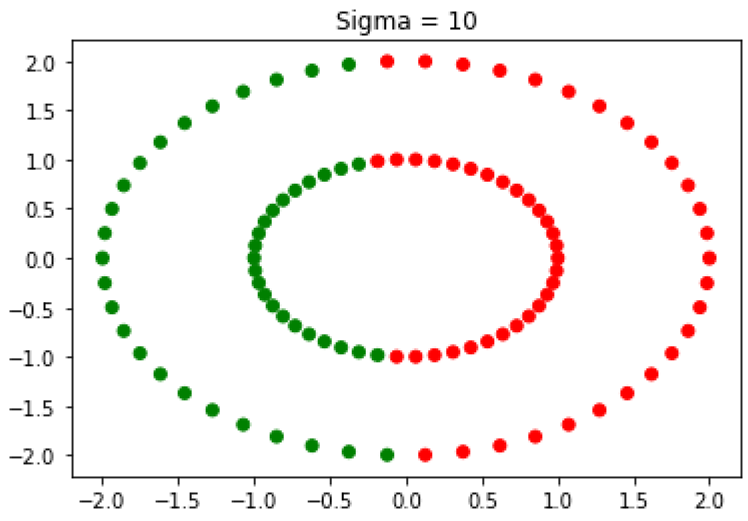
```
Laplacian Computed
Eigen values computed
```



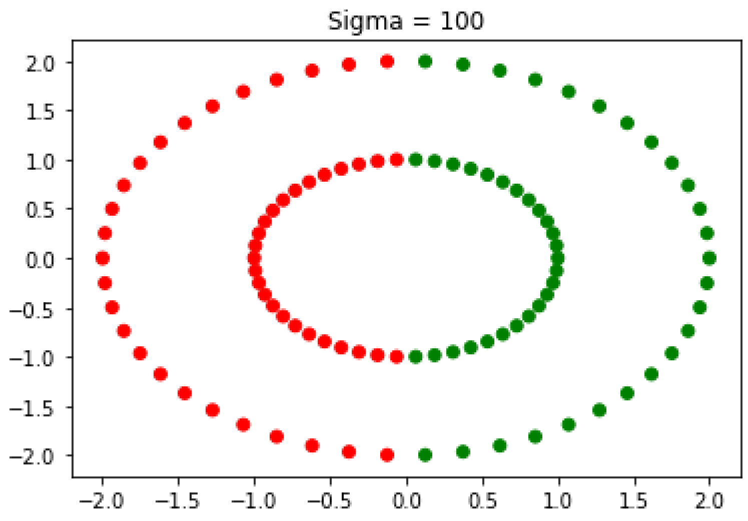
```
Laplacian Computed
Eigen values computed
```



```
Laplacian Computed
Eigen values computed
```

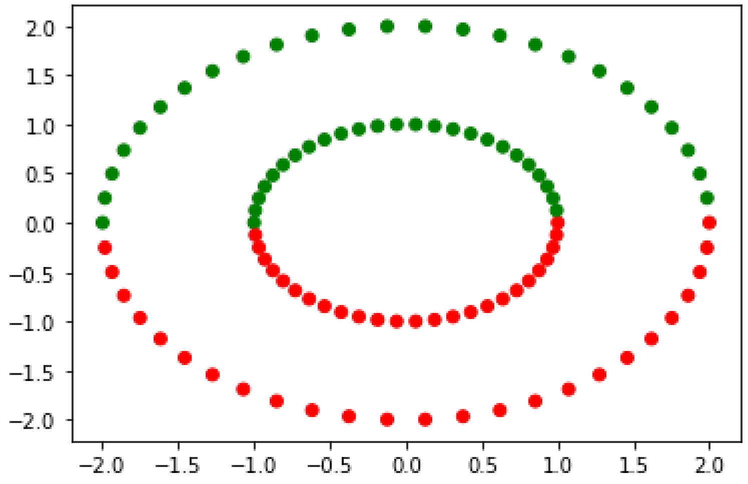


```
Laplacian Computed
Eigen values computed
```



Use the k-means algorithm in MATLAB/Python to compute an alternative clustering.

In [9]:
```
kmeans = KMeans(n_clusters=K).fit(X)
label_color = ['r' if l else 'g' for l in kmeans.labels_]
plt.scatter(X[:, 0], X[:, 1], c=label_color)
plt.show()
```



3. Find a choice of σ such that the spectral method outperforms k-means. How do you know that there is no k-means solution (i.e., a choice of centers and clusters) that performs this well? Include the output of your code in your submission

$\sigma = 0.1$ produces a good clustering compared to k-means

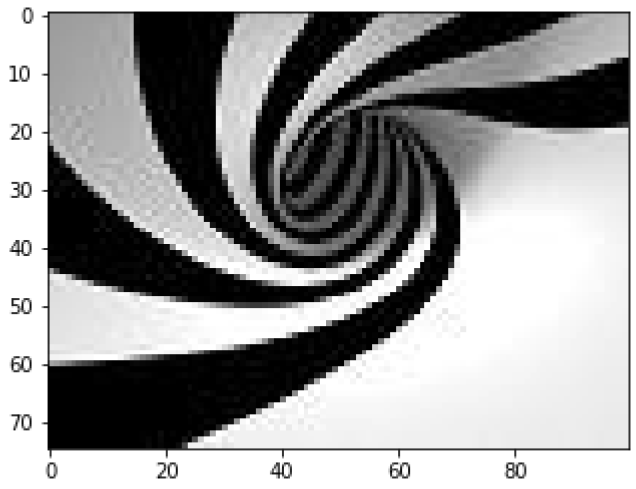
k-means is well suited for linearly separable data. This dataset with k = 2 is not linearly separable and k-means will perform poorly. However, spectral clustering is similar to applying feature map to data and then performing k-means clustering on higher dimensions where the data would be linearly separable.

Partitioning Images

1. We can use the same spectral technique to partition images. Here, we consider each pixel of a grayscale image as a single intensity and construct a similarity matrix for pairs of pixels just as before

2. Perform the same comparison of spectral clustering and k-means as before using the image bw.jpg that was attached as part of the homework. Again, set k = 2. You can use imread() to read an image from a file in MATLAB.

In [10]:
```python
img = plt.imread('bw.jpg')
plt.imshow(img, cmap='gray')
h, w = img.shape
img = img.ravel()
img = img.reshape(-1,1)
```

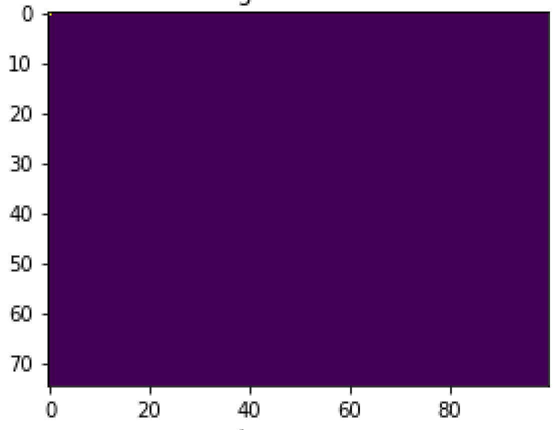


In [16]:
```python
for sigma in sigma_list:
    print(f"computing for Sigma = {sigma}")
    A = compute_similarity_matrix(img, sigma)
    clusters, labels = spectral_clustering(A, K)
    image_labels = np.array(labels).astype(np.float)
    image_labels = np.reshape(image_labels, (h, w))
    plt.imsave(f'bw{sigma}.png',image_labels)
```

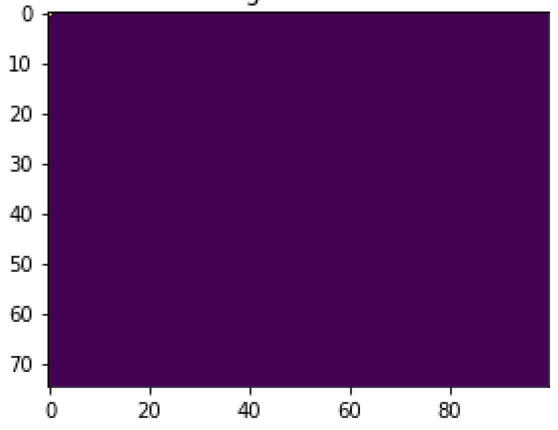
```
computing for Sigma = 0.01

/home/xnkr/ml/lib/python3.6/site-packages/ipykernel_launcher.py:13: RuntimeWa
rning: overflow encountered in exp
  del sys.path[0]

computing for Sigma = 0.1

/home/xnkr/ml/lib/python3.6/site-packages/ipykernel_launcher.py:13: RuntimeWa
rning: overflow encountered in exp
  del sys.path[0]

computing for Sigma = 1
computing for Sigma = 10
computing for Sigma = 100
```

In [41]:
```
fig,a =  plt.subplots(5,1,figsize=(20,20))
it = 0
for i in range(5):
    s = sigma_list[i]
    k = plt.imread(f'bw{s}.png')
    a[i].set_title(f'Sigma = {s}')
    a[i].imshow(k)
plt.show()
```

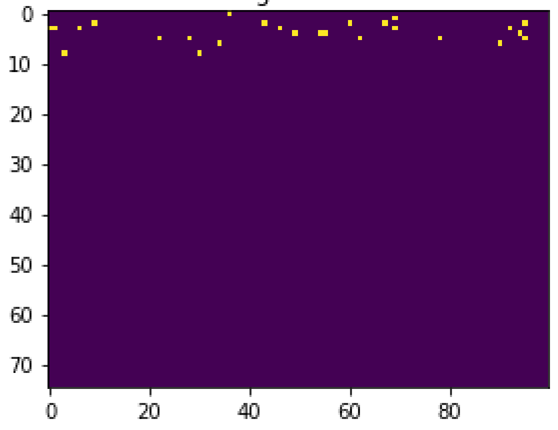
Sigma = 0.01



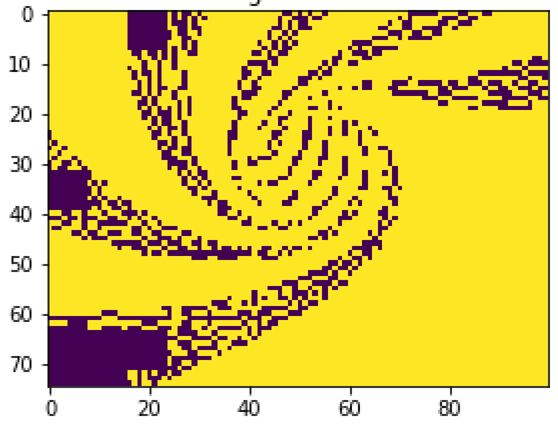
Sigma = 0.1



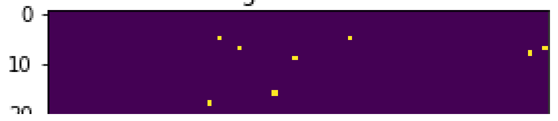
Sigma = 1

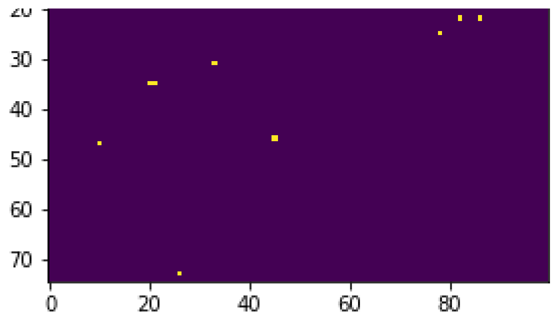


Sigma = 10



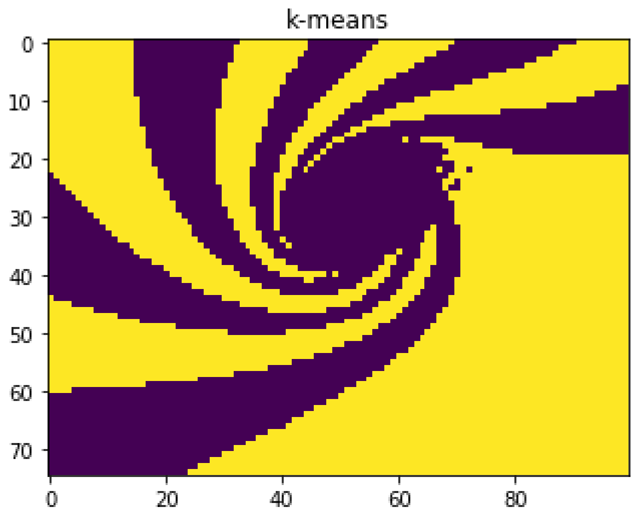
Sigma = 100





In [25]:
```
kmeans = KMeans(n_clusters=2).fit(img)
clusters_kmeans = form_clusters(kmeans.labels_, img, K)
image_labels = np.reshape(kmeans.labels_, (75, 100))
plt.imsave('bw-kmeans.jpg',image_labels)
plt.title('k-means')
plt.imshow(image_labels)
```

Out[25]:
```
<matplotlib.image.AxesImage at 0x7fccae0d69b0>
```



2.1 Prove that L is positive semidefinite

$$x^T L x \geq 0$$

$L = D - A$

$D_i = \sum_j A_{ij}$ $\qquad A = e^{-\frac{1}{2\sigma^2}\|x_i - x_j\|^2}$

$$x^T L x = x^T (D - A) x$$

$$= \sum_i d_i x_i^2 - \sum_{ij} A_{ij} x_i x_j$$

$$= 2 \sum A_{ij} (x_i)^2 - 2 \sum A_{ij} x_i x_j$$

$$= \sum A_{ij} x_i^2 - 2 \sum A_{ij} x_i x_j + \sum A_{ij} x_j^2$$

$$= \sum A_{ij} (x_i - x_j)^2$$

$A_{ij} \geq 0$ $\qquad (x_i - x_j)^2 \geq 0$

$$\Rightarrow x^T L x \geq 0$$