Principal Component Analysis

Name: Rajarshi Chattopadhyay | Net Id: RXC170010

Import necessary libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from numpy import mean
from numpy import cov
from numpy.linalg import eig
import warnings
warnings.filterwarnings('ignore')
```

Create the training, validation, and test feature(X) and label(y) matrices from the corresponding datasets

In [2]:
```
X_trn = np.loadtxt('data/usps.train', delimiter=',', usecols=(np.arange(1,257)))
y_trn = np.loadtxt('data/usps.train', delimiter=',', usecols=(0))

print("X_trn = " + str(X_trn.shape))
print("Y_trn = " + str(y_trn.shape))

X_val = np.loadtxt('data/usps.valid', delimiter=',', usecols=(np.arange(1,257)))
y_val = np.loadtxt('data/usps.valid', delimiter=',', usecols=(0))

print("X_val = " + str(X_val.shape))
print("Y_val = " + str(y_val.shape))

X_tst = np.loadtxt('data/usps.test', delimiter=',', usecols=(np.arange(1,257)))
y_tst = np.loadtxt('data/usps.test', delimiter=',', usecols=(0))

print("X_tst = " + str(X_tst.shape))
print("Y_tst = " + str(y_tst.shape))
```

```
X_trn = (1000, 256)
Y_trn = (1000,)
X_val = (300, 256)
Y_val = (300,)
X_tst = (300, 256)
Y_tst = (300,)
```

Center the features data

In [3]:
```
# Centre each feature
C_trn = X_trn - mean(X_trn, axis=0)
print("C_trn = " + str(C_trn.shape))

C_val = X_val - mean(X_val, axis=0)
print("C_val = " + str(C_val.shape))

C_tst = X_tst - mean(X_tst, axis=0)
print("C_tst = " + str(C_tst.shape))
```

```
C_trn = (1000, 256)
C_val = (300, 256)
C_tst = (300, 256)
```

Obtain the features covariance matrix of the centered features

In [4]:
```
# Find the feature covariance matrix on the training set
Cov = cov(C_trn.T)
print("Cov = " + str(Cov.shape))
```

```
Cov = (256, 256)
```

Find the Eigen vectors for the features from the covariance matrix

```
In [5]: # Perform eigen-decomposition of the covariance matrix
        eigen_vals, eigen_vecs = eig(Cov)

        print("Eigen value\tEigen Vectors")
        print(str(eigen_vals.shape) + "\t\t" + str(eigen_vecs.shape))
```

```
Eigen value     Eigen Vectors
(256,)          (256, 256)
```

Find the top 16 Eigen vectors from the training set with the highest Eigen values

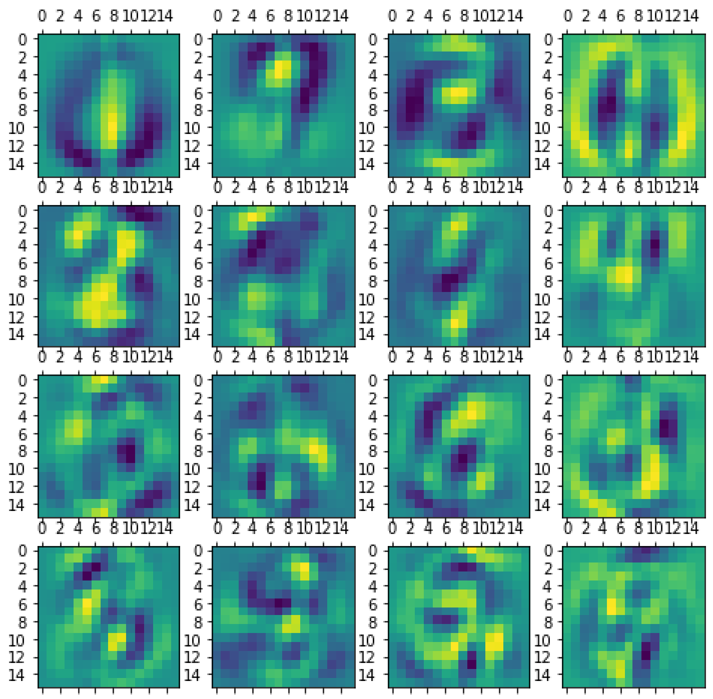
```
In [6]: # Make a list of (eigenvalue, eigenvector) tuples
        eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:,i]) for i in range(len(eigen_vals))]

        # Sort the (value, vector) tuples from high to low value
        eigen_pairs.sort(key=lambda x: x[0], reverse=True)

        # Find the top 16 eigenvectors
        vec_dict = {}
        i = 0
        for p in eigen_pairs[:16]:
            vec_dict[i] = p[1]
            i = i+1
        digits = vec_dict.values()
```

Compute the projections (Eigen digits) of the top 16 Eigen vectors, and vizualize

```
In [7]: # Visualize the top 16 eigendigits
        fig = plt.figure(figsize=(8,8))
        i=1
        for d in digits:
            ax = fig.add_subplot(4,4,i)
            ax.matshow(d.reshape((16,16)).astype(float))
            i=i+1
        plt.show()
```

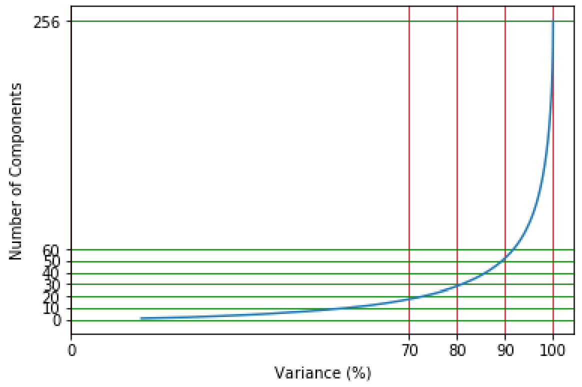


Plot the cumulative explained variance ratio vs. number of components from the training set eigens

```
In [8]: # Find cumulative explained variance in training set
total = np.sum(eigen_vals)
cum_exp_var_dict = {}
i = 1
sum = 0
for p in eigen_pairs:
    sum = sum + (p[0]*100/total)
    cum_exp_var_dict[i] = sum
    i = i+1

cv_dict = dict(zip(cum_exp_var_dict.values(),cum_exp_var_dict.keys()))

# Plot cumulative explained variance vs number of components plot of the traning set
plt.figure()
plt.grid(axis='x', color='r')
plt.grid(axis='y', color='g')
plt.plot(cv_dict.keys(), cv_dict.values())
plt.ylabel('Number of Components')
plt.xlabel('Variance (%)')
plt.yticks([0, 10, 20, 30, 40, 50, 60, 256])
plt.xticks([0, 70, 80, 90, 100])
plt.show()
```



To achieve 70%, 80% and 90% of the total variance in the training set, the respective approximate dimensionality required are as follows:

- k_{70} = ~18 components
- k_{80} = ~29 components
- k_{90} = ~54 components

For k_{100} all 256 components are required

To obtain X_{70} with 70% variance, we project 18 components
To obtain X_{80} with 80% variance, we project 29 components
To obtain X_{90} with 90% variance, we project 54 components
To obtain X_{100} with 100% variance, we project 256 components

Reduce the 256-dimensional feature space to a k-dimensional feature subspace, by choosing the "top k" eigenvectors with the highest eigenvalues to construct our 256 x k-dimensional eigenvector matrix W

```
In [9]: # Find projection matrix with top 18, 29, and 54 components
W_18 = np.hstack([eigen_vecs[i].reshape(256,1) for i in range(0,18)])
print("W_18 = " + str(W_18.shape))

W_29 = np.hstack([eigen_vecs[i].reshape(256,1) for i in range(0,29)])
print("W_29 = " + str(W_29.shape))

W_54 = np.hstack([eigen_vecs[i].reshape(256,1) for i in range(0,54)])
print("W_54 = " + str(W_54.shape))
```

```
W_18 = (256, 18)
W_29 = (256, 29)
W_54 = (256, 54)
```

Use the projection matrix to transform our datasets onto the new subspace

In [10]:
```python
# Find the projection of the training, validation, and test sets with the projection matrices
X_70_trn = C_trn.dot(W_18)
print("X_70_trn = " + str(X_70_trn.shape))

X_80_trn = C_trn.dot(W_29)
print("X_80_trn = " + str(X_80_trn.shape))

X_90_trn = C_trn.dot(W_54)
print("X_90_trn = " + str(X_90_trn.shape))

X_100_trn = C_trn
print("X_100_trn = " + str(X_100_trn.shape))

X_70_val = C_val.dot(W_18)
print("X_70_val = " + str(X_70_val.shape))

X_80_val = C_val.dot(W_29)
print("X_80_val = " + str(X_80_val.shape))

X_90_val = C_val.dot(W_54)
print("X_90_val = " + str(X_90_val.shape))

X_100_val = C_val
print("X_100_val = " + str(X_100_val.shape))

X_70_tst = C_tst.dot(W_18)
print("X_70_tst = " + str(X_70_tst.shape))

X_80_tst = C_tst.dot(W_29)
print("X_80_tst = " + str(X_80_tst.shape))

X_90_tst = C_tst.dot(W_54)
print("X_90_tst = " + str(X_90_tst.shape))

X_100_tst = C_tst
print("X_100_tst = " + str(X_100_tst.shape))
```

```
X_70_trn = (1000, 18)
X_80_trn = (1000, 29)
X_90_trn = (1000, 54)
X_100_trn = (1000, 256)
X_70_val = (300, 18)
X_80_val = (300, 29)
X_90_val = (300, 54)
X_100_val = (300, 256)
X_70_tst = (300, 18)
X_80_tst = (300, 29)
X_90_tst = (300, 54)
X_100_tst = (300, 256)
```

Learn different multi-class SVM classifiers for different regularization terms, and evaluate over validation set

In [11]:
```
alpha_range = np.arange(-4.0, 0, 1.0)
alpha_values = np.power(10.0, alpha_range)

from sklearn.linear_model import SGDClassifier

print("{:9s}\t{:9s}\t{:9s}\t{:9s}".format('k', 'alpha', 'Validation Error', 'Test Error'))

# Learn model with training set and evaluate on validation set for different alfa and projections

for a in alpha_values:
    clf = SGDClassifier(alpha=a, loss="hinge", penalty="l2")

    clf_70_dict = clf.fit(X_70_trn, y_trn)
    valErr_X_70 = 1 - clf_70_dict.score(X_70_val, y_val)
    tstErr_X_70 = 1 - clf_70_dict.score(X_70_tst, y_tst)

    clf_80_dict = clf.fit(X_80_trn, y_trn)
    valErr_X_80 = 1 - clf_80_dict.score(X_80_val, y_val)
    tstErr_X_80 = 1 - clf_80_dict.score(X_80_tst, y_tst)

    clf_90_dict = clf.fit(X_90_trn, y_trn)
    valErr_X_90 = 1 - clf_90_dict.score(X_90_val, y_val)
    tstErr_X_90 = 1 - clf_90_dict.score(X_90_tst, y_tst)

    clf_100_dict = clf.fit(X_100_trn, y_trn)
    valErr_X_100 = 1 - clf_100_dict.score(X_100_val, y_val)
    tstErr_X_100 = 1 - clf_100_dict.score(X_100_tst, y_tst)

    print("{:1d}\t\t{:0.4f}\t\t{:0.2f} %\t\t\t{:0.2f} %".format(70, a, valErr_X_70*100, tstErr_X_70*100)
    print("{:1d}\t\t{:0.4f}\t\t{:0.2f} %\t\t\t{:0.2f} %".format(80, a, valErr_X_80*100, tstErr_X_80*100)
    print("{:1d}\t\t{:0.4f}\t\t{:0.2f} %\t\t\t{:0.2f} %".format(90, a, valErr_X_90*100, tstErr_X_90*100)
    print("--------------------------------------------------------------------------------")
    print("{:1d}\t\t{:0.4f}\t\t{:0.2f} %\t\t\t{:0.2f} %".format(100, a, valErr_X_100*100, tstErr_X_100*1
    print("--------------------------------------------------------------------------------")
```

```
k                alpha            Validation Error        Test Error
70               0.0001           36.33 %                 38.33 %
80               0.0001           19.67 %                 17.00 %
90               0.0001           15.00 %                 13.67 %
--------------------------------------------------------------------------------
100              0.0001           10.67 %                 8.00 %
--------------------------------------------------------------------------------
70               0.0010           30.00 %                 35.00 %
80               0.0010           19.67 %                 19.67 %
90               0.0010           14.00 %                 10.00 %
--------------------------------------------------------------------------------
100              0.0010           9.00 %                  8.33 %
--------------------------------------------------------------------------------
70               0.0100           31.67 %                 34.33 %
80               0.0100           20.33 %                 18.00 %
90               0.0100           14.00 %                 12.33 %
--------------------------------------------------------------------------------
100              0.0100           7.33 %                  7.33 %
--------------------------------------------------------------------------------
70               0.1000           34.33 %                 35.67 %
80               0.1000           25.33 %                 22.33 %
90               0.1000           18.33 %                 16.67 %
--------------------------------------------------------------------------------
100              0.1000           10.00 %                 9.00 %
--------------------------------------------------------------------------------
```

It is identified that the best model based on validation set with feature selection is for (k, α) pair (90, 0.001), and that without feature selection is for (k, α) (100, 0.01)

The best models on test data produce the following amount of error

- **54 features (cumulative variance 90%), alpha 0.001 => 14.00% validation error, 10.00% test error**
- **all 256 features (cumulative variance 100%), alpha 0.01 => 7.33% validation error, 7.33% test error**