Assignment 3 (ML)

Akhil Babu Manam (927000968)

November 30, 2018

1 Midterm Correction (Midterm 1 Question 2)

1.1 Which of the following are possible hyper-parameters of the corresponding models?

1. The number of K neighbors in K-NN classification (**True**)
2. The number of trees, depth of each tree and weight of each weak learner in Gradient Descent Boosting. (**False**) because weight are not hyper parameters
3. The step size in gradient descent. (**True**)
4. The λ weight of regularization. (**True**)
5. The weights β in logistic regression. (**False**)

1.2 Assume a non-linear regression model that predicts the stock price of a company based on the average number of sales, ranging between 1 and 900, such that price $= g_w(sales, \sqrt{sales})$. Further assume that the regression equation is written as $g_w = w_0 + w_1x_1 + w_2x_2$. The weights w of the model are found through gradient descent. What would be good choices of x_1 and x_2 so that the gradient descent method converges in a reasonable time? Justify your answer for full credit.

A. $x_1 =$ **sales,** $x_2 = \frac{\sqrt{sales}}{30}$
B. $x_1 =$ **sales,** $x_2 = 1000 * \sqrt{sales}$
C. $x_1 = \frac{sales}{30}$, $x_2 = \sqrt{sales}$
D. $x_1 =$ **sales,** $x_2 = \frac{\sqrt{sales}}{30}$

Gradient descent performs better in terms of converging when both the dimensions are in similar ranges. Since sales is in range 0 and 900, $\sqrt{sales}$ is in range 0 and 30. Therefore, option **D**, 30*$\sqrt{sales}$ and sales will be in the same range 0 and 900. Hence, for faster convergence option **D** is best.

1.3 Recall that precision and recall are defined at a specific probability threshold as:

$$precision = \frac{Truepositives}{TruePositives + FalsePositives} \tag{1}$$

$$recall = \frac{Truepositives}{Truepositives + Falsenegatives} \tag{2}$$

Please calculate the precision, recall and F1 score with threshold p=0.45 and p=0.55. (You can approximate the fraction values to 2 decimal points).

With threshold p = 0.45:
True positives = 4 (50, 53, 75, 100)
True negatives = 1 (14)
False positives = 1 (57)
False negatives = 4 (0, 5, 10, 24)
precision = $\frac{4}{5} = 0.80$
recall = $\frac{4}{8} = 0.50$
F1 = $\frac{2(0.80)(0.50)}{0.80+0.50} = 0.615$

With threshold p = 0.55:
True positives = 2 (75, 100)
True negatives = 3 (14, 50, 53)
False positives = 1 (57)
False negatives = 4 (0, 5, 10, 24)
precision = $\frac{2}{3} = 0.67$
recall = $\frac{2}{6} = 0.33$
F1 = $\frac{2(0.67)(0.33)}{0.67+0.33} = 0.44$

1.4 Please show two iterations of gradient descent with the following problem. Assume we want to find the right regression coefficients for the following dataset - to estimate final grade based upon hours studied:

Start with $\beta_0^{(0)} = 0$, $\beta_1^{(0)} = 10$, $\alpha = 0.01$ and $\epsilon = 0.01$. α is the step size and ϵ is the stopping criterion. You can round your intermediate results to simplify the computation. Itś enough to get approximated results.

Let the model be $y = \beta_0 + \beta_1 x$. From the given, learning rate $\alpha = 0.01$ and the stopping criteria $\epsilon == 0.01$.

The general ridge regression minimization function would be,

$$Cost = J = RSS + \lambda(\Sigma_{j=0}^{M}\beta_j^2) = \Sigma_{i=1}^{M}(y_i - \beta_0 - \beta_1 x_i)^2 + \lambda\beta_0^2 + \lambda\beta_1^2 \tag{3}$$

For minimizing the Cost,

$$\frac{\partial J}{\partial \beta_0} = -2\Sigma_{i=1}^{M}(y_i - \beta_0 - \beta_1 x_i) + 2\lambda\beta_0 \tag{4}$$

$$\frac{\partial J}{\partial \beta_1} = -2\Sigma_{i=1}^{M}x_i(y_i - \beta_0 - \beta_1 x_i) + 2\lambda\beta_1 \tag{5}$$

The update equation based on gradient descent is as follows,

$$\beta_k^i = \beta_k^{i-1} - \alpha\frac{\partial J}{\partial \beta_k} \tag{6}$$

Assuming $\lambda = 0.1, computing \beta_0^1$ and β_1^1 using $\beta_0^0 = 0$ and $\beta_1^0 = 10$,

$$\frac{\partial J}{\partial \beta_0} = -2(98 - 90 + 73 - 80 + 24 - 10 + 57 - 60 + 18 - 20 + 83 - 100) = 14 \tag{7}$$

$$\frac{\partial J}{\partial \beta_1} = -2(9*8+8*(-7)+1*(14)+6*(-3)+2*(-2)+10*(-17))+10\lambda = 324+1 = 325 \tag{8}$$

Therefore, $\beta_0^1 = \beta_0^0 - \alpha\frac{\partial J}{\partial \beta_0} = 0 - 0.01(14) = -0.14$ and $\beta_1^1 = \beta_1^0 - \alpha\frac{\partial J}{\partial \beta_1} = 10 - 0.01(325) = 6.75$

After each iteration, we keep checking if the change in RSS from previous iteration is over the value ϵ or not and only continue if it is over it.

Therefore, for iteration 2,

$$\frac{\partial J}{\partial \beta_0} = -2(5.21 + 2.66 + 2.415 + 2.31 + 0.63 + 2.17) = -30.79 \tag{9}$$

$$\frac{\partial J}{\partial \beta_1} = -2(46.89+21.28+2.415+13.86+1.26+21.7)+10\lambda = -214.81+67.5 = -147.31 \tag{10}$$

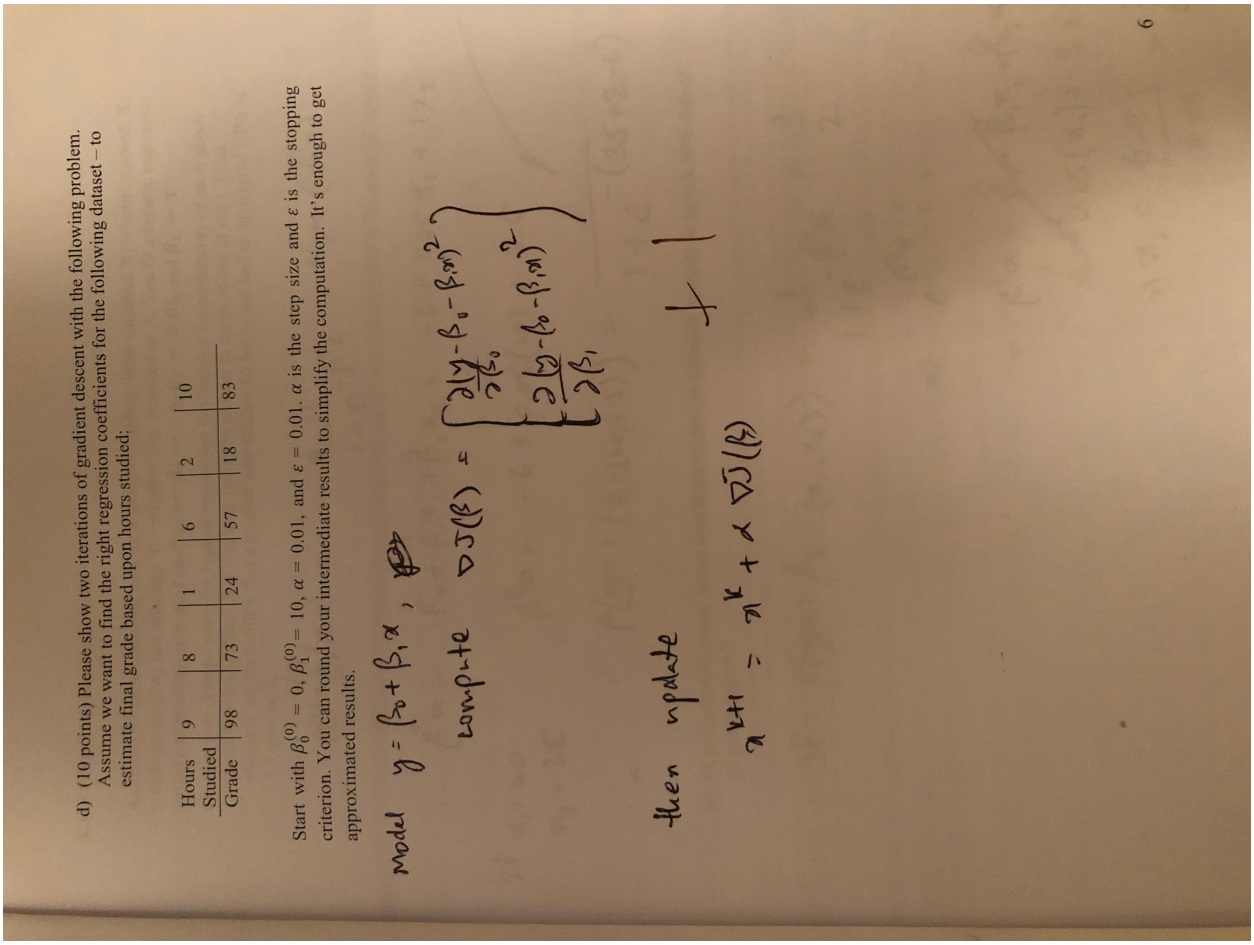
Therefore, $\beta_0^1 = \beta_0^0 - \alpha\frac{\partial J}{\partial \beta_0} = -0.14 - 0.01(-30.79) = 0.16$ and $\beta_1^1 = \beta_1^0 - \alpha\frac{\partial J}{\partial \beta_1} = 6.75 - 0.01(-147.31) = 8.22$

Explanation on answering this question from midterm 1 : I could not solve this question during midterm 1 due to time constraints, I could not have time to solve the d part of this problem. The image of unsolved part of this question can be found in fig.1.4.

2 Activation Function in Neural Networks

2.1 Provide a schematic representation of the network. Define the variables for the input, the output, and the weights

Input I = $(i_1, i_2, ..., i_D)$
Hidden Nodes H = $(h_1, h_2, ...h_M)$



d) (10 points) Please show two iterations of gradient descent with the following problem. Assume we want to find the right regression coefficients for the following dataset – to estimate final grade based upon hours studied:

Hours Studied	9	8	1	6	2	10
Grade	98	73	24	57	18	83

Start with $\beta_0^{(0)} = 0$, $\beta_1^{(0)} = 10$, $\alpha = 0.01$, and $\varepsilon = 0.01$. α is the step size and ε is the stopping criterion. You can round your intermediate results to simplify the computation. It's enough to get approximated results.

model $y = \beta_0 + \beta_1 x$,

compute $\nabla J(\beta) = \begin{Bmatrix} \frac{\partial J}{\partial \beta_0} \\ \frac{\partial J}{\partial \beta_1} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial (y - \beta_0 - \beta_1 x)^2}{\partial \beta_0} \\ \frac{\partial (y - \beta_0 - \beta_1 x)^2}{\partial \beta_1} \end{Bmatrix}$

then update

$\pi^{k+1} = \pi^k + \alpha \nabla J(\beta)$

Weights between layers H and I = $(W_{ij}^{(1)})$ where i denotes an input node and j denotes a hidden node.
Output O = (o)
Weights between layers O and H = $(W_{ij}^{(2)})$ where i denotes a hidden node and j denotes output node.
The diagram representing the neural network can be found in fig.1

2.2 Express the output as a function of the input

Let us assume a sigmoid function as the activation function at the hidden layer and output node. The sigmoid function can be represented as follows,

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{11}$$

Assuming input I = $(i_1, i_2, ..., i_D)$, hidden neurons $h_1, h_2, ..., h_M$ and weights as mentioned in fig.1, the kth hidden neuron can be expressed as follows

$$h_k = \sigma(w_{1k}^{(1)} i_1 + w_{2k}^{(1)} i_2 + ... + w_{Dk}^{(1)} i_D + b_{11}) = \sigma(\Sigma_{p=1}^{D} w_{pk}^{(1)} i_p + b_{11}) \tag{12}$$

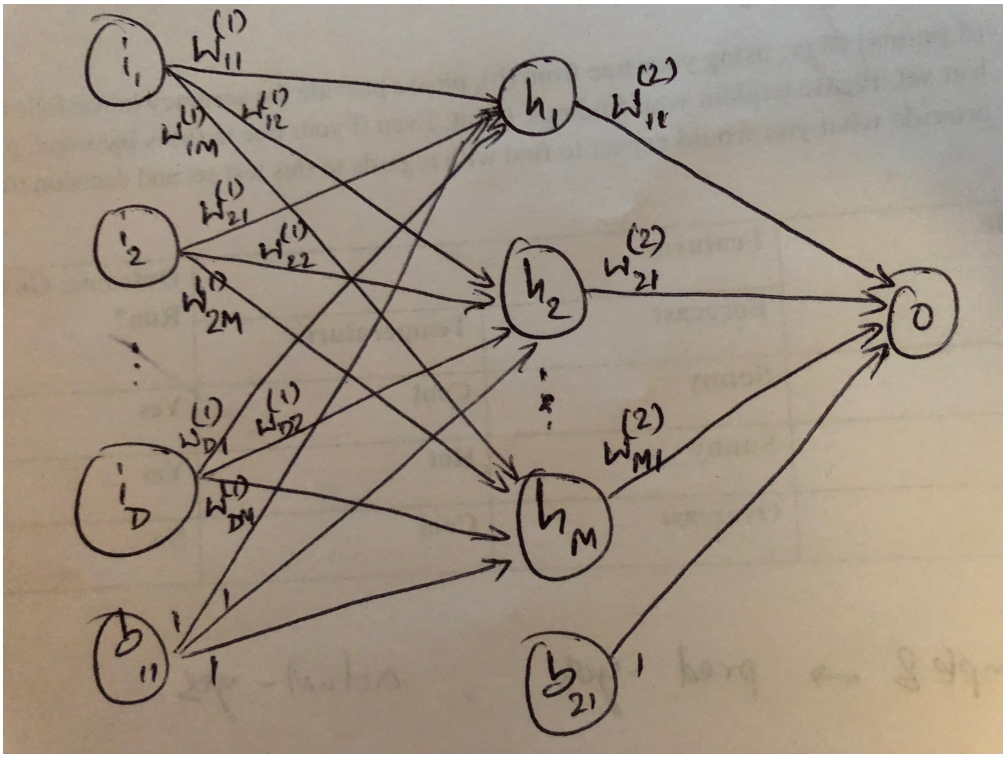


Figure 1: Eigenvalues vs. the component number

and the output can be expressed as

$$o = w_{11}^{(2)} h_1 + w_{21}^{(2)} h_2 + ... + w_{M1}^{(2)} h_M + b_{21} = \Sigma_{j=1}^{M} w_{j1}^{(2)} h_j + b_{21} \tag{13}$$

Combining eq.12 and eq.13,

$$o = \Sigma_{j=1}^{M} w_{ji}^{(2)} (\sigma(\Sigma_{p=1}^{D} w_{pk}^{(1)} i_p + b_{11})) + b_{21} \tag{14}$$

2.3 Calculate the number of parameters that need to be inferred

The number of parameters between input layer and hidden layer is $(D+1) * M$ and the number of parameters between hidden layer and the output layer is $(M+1) * 1$. Therefore, the total parameters of the network are $(D+2) * M + 1$.

2.4 Show that there exists an equivalent network with hidden unit activation functions given by the hyperbolic tangent, which computes exactly the same function, where the hyperbolic tangent is given by

The hyperbolic tangent function can be expressed as

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} = \frac{2}{1 + e^{-2x}} - \frac{1 + e^{-2x}}{1 + e^{-2x}} = \frac{2}{1 + e^{-2x}} - 1 \tag{15}$$

and the sigmoid function can be expressed as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{16}$$

By substituting eq.16 in eq.15, it can be written as follows,

$$tanh(x) = 2\sigma(2x) - 1 \tag{17}$$

Therefore,

$$\sigma(x) = \frac{tanh(\frac{x}{2}) + 1}{2} \tag{18}$$

Substituting eq.18 in eq.14 from section 2.2,

$$o = \Sigma_{j=1}^{M} w_{ji}^{(2)} (\frac{tanh(\frac{\Sigma_{p=1}^{D} w_{pk}^{(1)} i_p + b_{11})}{2}) + 1}{2}) + b_{21} = \Sigma_{j=1}^{M} (\frac{w_{ji}^{(2)} tanh(\frac{\Sigma_{p=1}^{D} w_{pk}^{(1)} i_p + b_{11})}{2})}{2} + \frac{w_{ji}^{(2)}}{2}) + b_{21} \tag{19}$$

By substituting $\frac{w_{ji}^{(2)}}{2} = w_{ji}^{(2)'}$ and $\Sigma_{j=1}^{M} \frac{w_{ji}^{(2)}}{2} + b_{21} = b_{21}^{'}$, eq.14 can be re-written as,

$$o = \Sigma_{j=1}^{M} w_{ji}^{(2)'} (tanh(\Sigma_{p=1}^{D} w_{pk}^{(1)} i_p + b_{11})) + b_{21}^{'} \tag{20}$$

Hence, a new neural network with replaced weights between hidden and output layers and bias of the hidden layer can be mathematically represented using tanh activation function which is similar to the sigmoid activation based neural network but with changes weights and bias.

3 Image processing for human faces

3.1 Download the dataset from the above URL. Implement Principal Component Analysis (PCA) on the input images. Assume that the input vector of PCA contains all rows of an image stacked one on top of the other. You can use available libraries that calculate eigenvalues and eigenvectors of a matrix.

The code for performing principal component analysis (PCA) on the input images is available in appendix.

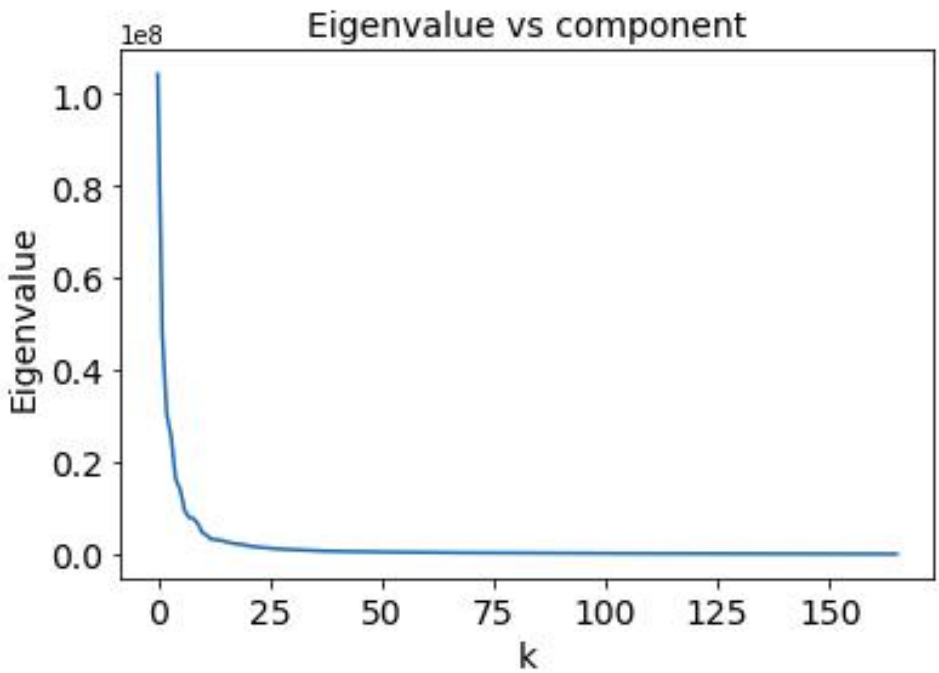


Figure 2: Eigenvalues vs. the component number

3.2 Plot a curve displaying the first k eigenvalues 1, . . . , K, i.e. the energy of the first K principal components. How many components do we need to capture 50% of the energy?

The ratio captured by top-k components can be computed as follows:

$$\mathbf{Energy(K)} = \frac{\mathbf{\Sigma_{k=1}^{K}\lambda_k}}{\mathbf{\Sigma_{allk}\lambda_k}} \tag{21}$$

The Energy(k=2) is 0.445 and Energy(k=3) is 0.534. Therefore, 3 components are needed to capture more than 50% variance.

3.3 Plot the top 10 eigenfaces, i.e. the eigenvectors uk, k = 1, . . . , 10 obtained by PCA.

The top 10 eigenfaces obtaine by PCA are as follows:

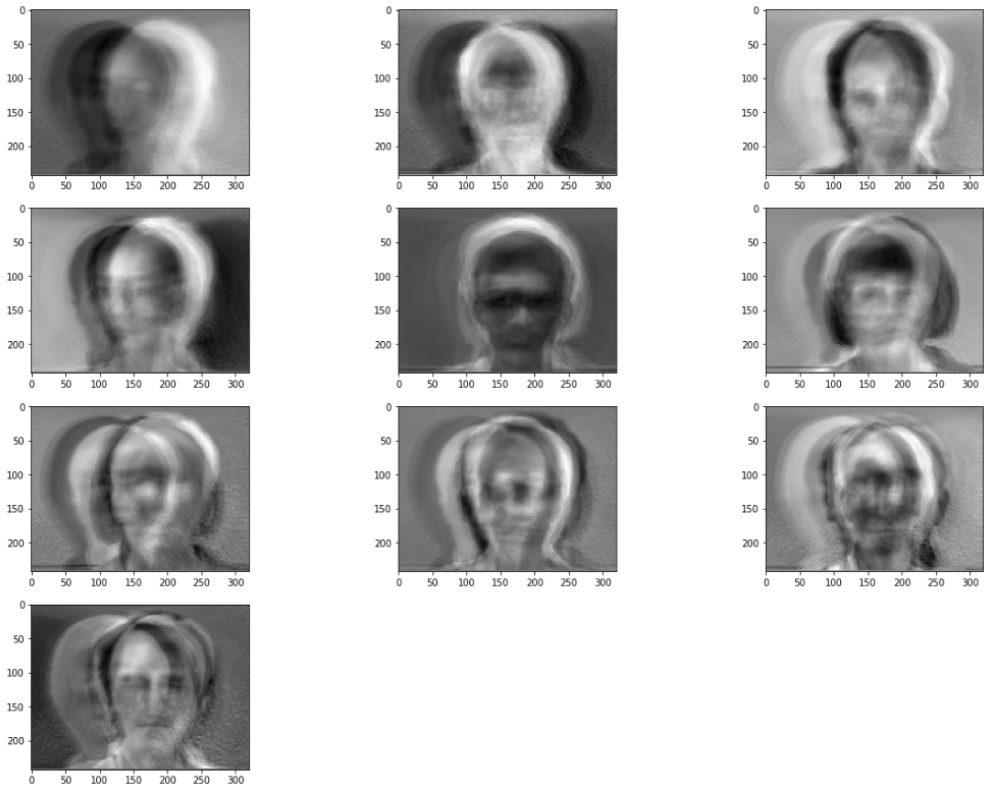


Figure 3: Eigenfaces corresponding to top 10 eigenvalues arranged from top left to bottom right

3.4 Select a couple of images from the data. Use the first k eigenfaces as a basis to reconstruct the images. Visualize the reconstructed images using 1, 10, 20, 30, 40, 50 components. How many components do we need to achieve a visually good result?

The original and reconstruction of 2 original face images can be seen below in fig.4 and fig.5 by using different components of PCA eigenfaces. It has been observed that reconstructed images are generally recognizable by using 40-50 components.

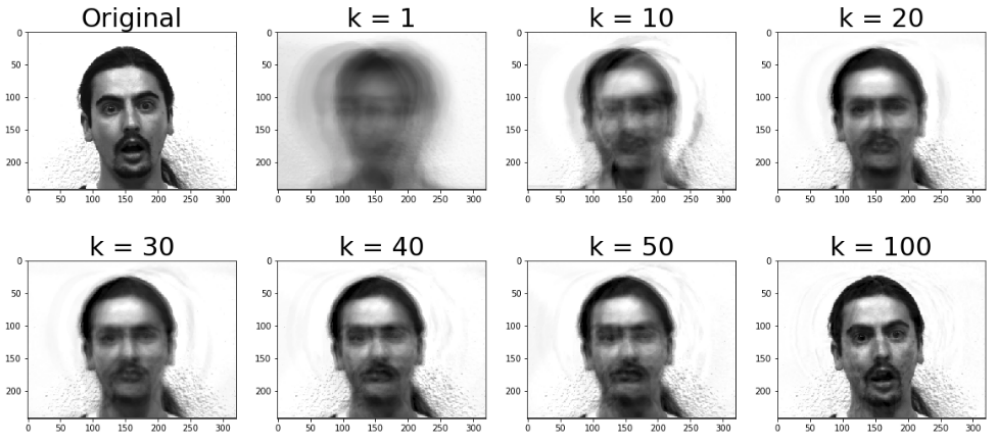


Figure 4: Face 1 original and reconstructions

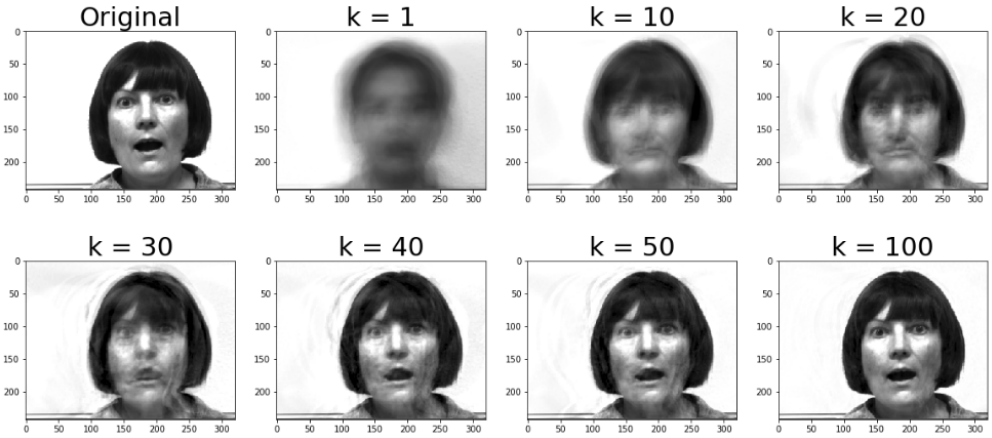


Figure 5: Face 2 original and reconstructions

3.5 Perform face recognition: Split the input data into training and testing making sure that every person is included in each set. Use as input features the transformed feature space that resulted from PCA. Experiment with different number of PCA components through a 5-fold cross-validation. Use an outer 5-fold cross-validation to build predictors using a boosting method (e.g. adaboost, xgboost), support vector machines (report kernel you used), and CNNs (experiment with filter size, stride size, activation function, drop out). Report the recognition accuracy on the test set.

Initially, training and testing sets are separated maintaing the same class distribution as

Classifier	No. of components (PCA - best)	Train Accuracy	Test Accuracy
Adaboost	24	77.97	52.94
Xgboost	22	76.55	73.52
SVM (poly)	50	76.58	82.35
SVM (rbf)	98	10.65	17.64

Figure 6: Results on the face recognition task

in the actual data with 80% for training and 20% for testing. Then k-fold cross validation was used to evaluate each classifier (adaboost, xgboost, SVM with poly and rbf kernel). While performing this k-fold cross validation, an inner loop was also run to find the best pca dimension to perform the recognition task. Finally pca with the best split was chosen and test results are calculated. Fig.6 shows the results of various classifiers and the number of components used finally. The number of components is chosen by computing the mean over all the folds corresponding to a classifier and then choosing number of components with highest accuracy.

3.6 Perform type recognition with Gaussian Mixture Models: Setting the number of components to be equal to the number of poses each subject can have, and using the data obtained by PCA, see if a Gaussian Mixture Model, trained with Expectation Maximization, naturally clusters the pictures by pose type.

For analyzing this part, we have always considered the number of gaussian components to be equal to the number of different types of images available which is 12 as mentioned in the question. Using 12 gaussian components and the expectation maximization technique, gaussian mixture models were trained using PCA components of images. The gassian mixture models are trained using 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110 and 120 PCA components. The accuracy of type recognition task is computed at each of these PCA components. For computing accuracy, each mixture is assigned a class which is the mode of the types in that mixture and finally the accuracy is computed as the number of types correctly assigned over total number of images. It has been observed that the accuracy was highest around 80 - 100 PCA components. It has been observed that leftlight and rightlight images are pure and got clustered into single mixtures respectively and all other clusters had mixture of types although they had clear majorities. To visualize the clusters, all the images are converted to 2-dimensions and then a EM algorithm is applied for tuning 12 gaussian clusters. Finally,

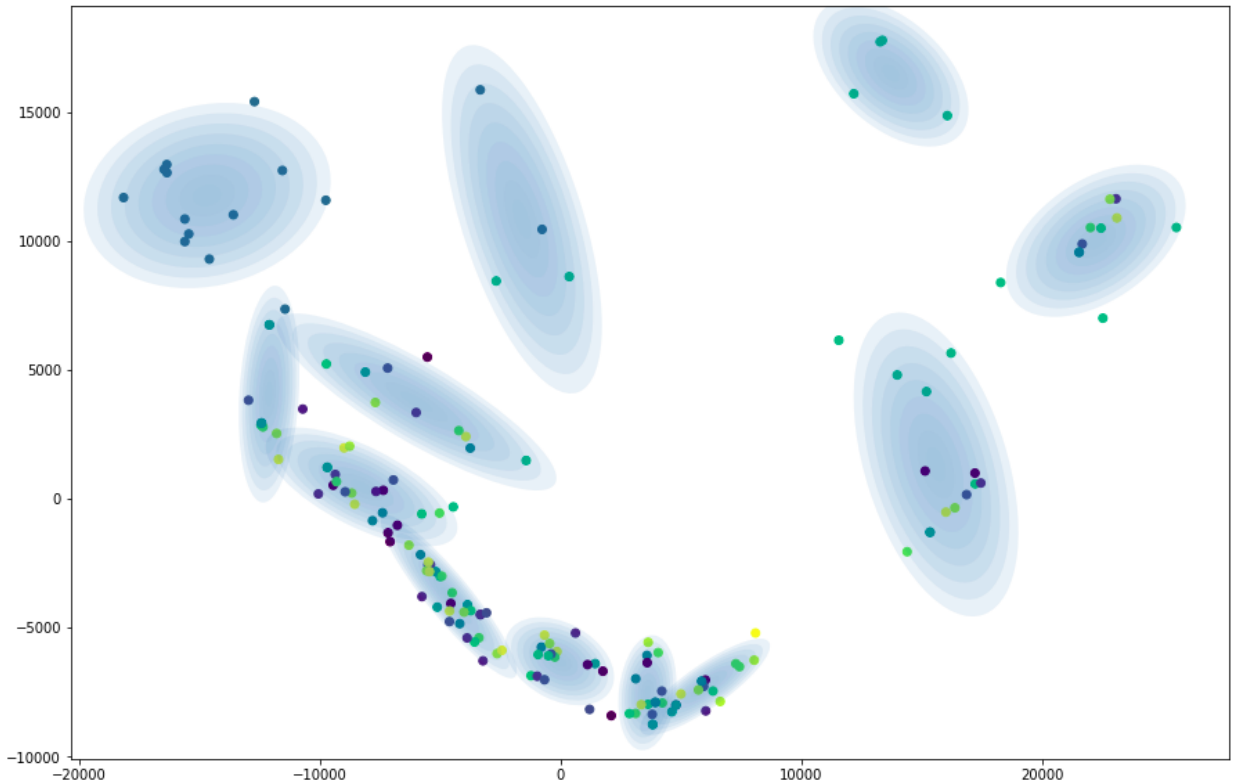


Figure 7: Full Gaussian mixtures plotted around their corresponding centers with representation using variances and each point coloured based on the type of the image.

the clusters and image points coloured according to type are plotted in fig.7.

4 Appendix

4.1 Code for 3.a

```
# load all images as a dictionary
from PIL import Image
import numpy as np
import os
data_tag = {}
for subdir, dirs, files in os.walk('../yalefaces/yalefaces'):
for f in files:
    fpath = subdir + os.sep + f
    if f != 'Icon' and f != 'Readme.txt':
        data_tag[f] = np.asarray(Image.open(fpath))

# prepare data - stack all rows of each image
```

```
data_vec = {}
for k,v in data_tag.items():
    data_vec[k] = np.ravel(v)

# compute eigen analysis on the vectors
data = data_vec.values()
data = np.array(data)
n_feat = data.shape[1]
print n_feat
#print np.mean(data, axis=0).reshape(1, n_feat).shape
data_m = np.mean(data, axis=0)
data = data - np.mean(data, axis=0)

from sklearn.decomposition import PCA
pca = PCA(n_components=data.shape[0])
pca.fit(data)
eigenvals = pca.explained_variance_
```

4.2 Code for 3.b

```
from sklearn.decomposition import PCA
pca = PCA(n_components=data.shape[0])
pca.fit(data)
eigenvals = pca.explained_variance_
ratio = np.array([np.sum(eigenvals[:i+1])/np.sum(eigenvals) for i in
    range(eigenvals.shape[0])])
print ratio

import matplotlib.pyplot as plt
plt.plot(eigenvals)
plt.xticks(size=14)
plt.yticks(size=14)
plt.xlabel('k', size=14)
plt.ylabel('Eigenvalue', size=14)
plt.title('Eigenvalue vs component', size=14)
plt.show()
```

4.3 Code for 3.c

```
# top 10 eigenfaces
eigvecs = pca.components_
dim = data_tag.values()[0].shape

for i in range(10):
```

```
    eigvecs[i,:] -= np.min(eigvecs[i,:])
    eigvecs[i,:] *= 255.0/np.max(eigvecs[i,:])

val1 = [np.reshape(eigvecs[i, :], (dim[0], dim[1])) for i in range(10)]
val = [Image.fromarray(np.reshape(eigvecs[i, :], (dim[0], dim[1]))) for i
    in range(10)]

k = 1
plt.figure(figsize=[20,15])
for i in val:
    plt.subplot(4,3,k)
    plt.imshow(i)
    k += 1
plt.show()
```

4.4 Code for 3.d

```
# reconstruct image using eigenvectors
data = data_vec.values()
data = np.array(data)
n_feat = data.shape[1]
print n_feat
#print np.mean(data, axis=0).reshape(1, n_feat).shape
data_m = np.mean(data, axis=0)
data = data - np.mean(data, axis=0)

from sklearn.decomposition import PCA
pca = PCA(n_components=data.shape[0])
pca.fit(data)

eigvecs = normalize(pca.components_, norm='l2', axis=1)
print eigvecs[0,:]

def reconstruct(eigvecs, n_comp, m_data, X):
    temp = np.zeros((m_data.shape))
    for i in range(n_comp):
        temp += np.dot(eigvecs[i,:],X) * eigvecs[i,:]
    temp += m_data
    return temp

X = data_vec.values()[0] - data_m
Y = reconstruct(eigvecs, 1, data_m, X)

X = np.reshape(data_vec.values()[0], data_tag.values()[0].shape)
Y = np.reshape(Y, data_tag.values()[0].shape)
```

```
X = Image.fromarray(X)
Y = Image.fromarray(Y)

j = 1
plt.figure(figsize=[20,9])
for k in range(1,2,1):
    X = data_vec.values()[k] - data_m
    X_im = np.reshape(data_vec.values()[k], data_tag.values()[k].shape)
    plt.subplot(2,4,j)
    plt.imshow(X_im, cmap='gray')
    plt.title('Original', size=30)
    j += 1
    for i in range(1,8,1):
        if i > 1 and i != 7:
            i = i*10-10
        if i == 7:
            i = 100
        Y = reconstruct(eigvecs, i, data_m, X)
        Y_im = Image.fromarray(np.reshape(Y, data_tag.values()[k].shape))
        plt.subplot(2,4,j)
        plt.imshow(Y_im, cmap='gray')
        plt.title('k = ' + str(i), size=30)
        j += 1
plt.show()
```

4.5 Code for 3.e

```
# split data into train and test
from sklearn.model_selection import train_test_split
ids = []
labels = []
for k in data_tag.keys():
    ids.append(k)
    labels.append(k.split('.')[0])
ids = np.array(ids)
labels = np.array(labels)
train_ids, test_ids, train_labels, test_labels = train_test_split(ids,
    labels, test_size=0.2, stratify=labels)

X = np.array([data_vec[i] for i in list(train_ids)])
y = train_labels
pca = PCA(n_components=100)
X = pca.fit_transform(X)

# AdaBoostClassifier
from sklearn.model_selection import KFold
```

```
kf = KFold(n_splits=5)
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
acc = {}
for train_index, test_index in kf.split(X):
    for k in range(10, 100, 2):
        if k % 20 == 0:
            print k
        X = np.array([data_vec[i] for i in list(train_ids)])
        y = train_labels
        pca = PCA(n_components = k)
        X = pca.fit_transform(X)
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=5))
        clf.fit(X_train, y_train)
        pred = clf.predict(X_test)
        if acc.has_key(k):
            acc[k].append(accuracy_score(y_test, pred))
        else:
            acc[k] = [accuracy_score(y_test, pred)]
mean_acc = {}
for k,v in acc.items():
    mean_acc[k] = np.mean(np.array(v))

chosen_k = sorted((v,k) for k,v in mean_acc.items())[-1][1]
print mean_acc[chosen_k]

X = np.array([data_vec[i] for i in list(train_ids)])
y = train_labels
pca = PCA(n_components=chosen_k)
X = pca.fit_transform(X)
clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=5))
clf.fit(X, y)
test_data = np.array([data_vec[i] for i in list(test_ids)])
test_data = pca.transform(test_data)
pred = clf.predict(test_data)
print accuracy_score(test_labels, pred)
print chosen_k
```

4.6 Code for 3.f

```
# Type recognition
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
```

```
ids = []
types = []
for k in data_tag.keys():
    ids.append(k)
    types.append(k.split('.')[1])
ids = np.array(ids)
types = np.array(types)
#types = enc.fit_transform(np.array(types))

# clustering in n dimensions
acc = {}
gmms = {}
for n in range(10, 120, 10):
    X = np.array([data_vec[i] for i in list(ids)])
    y = types
    pca = PCA(n_components=n)
    X = pca.fit_transform(X)
    gmm = GaussianMixture(n_components=len(set(types)),
        covariance_type='diag')
    labels = gmm.fit(X).predict(X)
    from scipy import stats
    tot = 0
    mixtures = []
    for i in range(12):
        mix = types[labels==i]
        tot += np.sum(mix == stats.mode(mix, axis=None))
        mixtures.append(list(mix))
    acc[n] = tot/float(167)
    gmms[n] = mixtures

chosen_k = sorted((v,k) for k,v in acc.items())[-1][1]
for i in gmms[chosen_k]:
    print i
    print ''
print chosen_k
print acc[chosen_k]
print sorted((v,k) for k,v in acc.items())

# plotting with clustering in 2 dimensions
X = np.array([data_vec[i] for i in list(ids)])
y = types
pca = PCA(n_components=2)
X = pca.fit_transform(X)
types = enc.fit_transform(np.array(types))

from sklearn.mixture import GaussianMixture
```

```
from matplotlib.patches import Ellipse
gmm = GaussianMixture(n_components=len(set(types)),
    covariance_type='full')
labels = gmm.fit(X).predict(X)
plt.figure(figsize=[15,10])
ax = plt.gca()
ax.scatter(X[:, 0], X[:, 1], c=types, cmap='viridis')
ax.legend()
for p,c in zip(gmm.means_, gmm.covariances_):
    U, s, Ut = np.linalg.svd(c)
    angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
    c = np.sqrt(s)
    for i in range(1,10,1):
        ax.add_patch(Ellipse(p, i*c[0]/2.0, i*c[1]/2.0, angle,
            alpha=i*0.01))
plt.show()
```