geyser

November 15, 2025

1 Implement EM Algorithm

1.1 Plotting geyser dataset

[1]:

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

[2]:

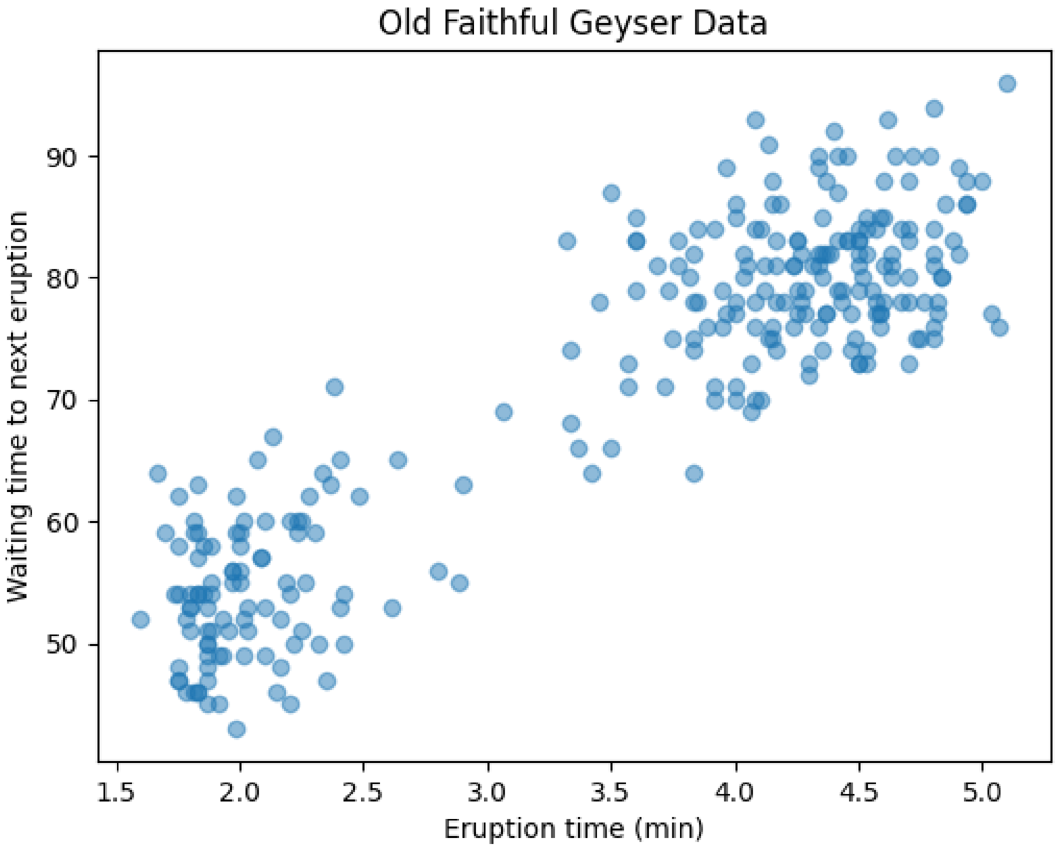
```
df = pd.read_csv("./data.txt", sep=r"\s+", engine="python",header=0)
df.head()
```

[2]:

```
   id  eruptions  waiting
0   1      3.600       79
1   2      1.800       54
2   3      3.333       74
3   4      2.283       62
4   5      4.533       85
```

[3]:

```
import matplotlib.pyplot as plt
plt.scatter(df['eruptions'], df['waiting'], alpha=0.5)
plt.xlabel('Eruption time (min)')
plt.ylabel('Waiting time to next eruption')
plt.title('Old Faithful Geyser Data')
plt.show()
```



1.2 Implement and run EM algorithm

1.2.1 E and M steps

I implemented the E and M steps from scratch for k=2.

[19]:

```
import numpy as np
from scipy.stats import multivariate_normal

def E_step(D, mean_1, cov_1, phi_1, mean_2, cov_2, phi_2):
    eps = 1e-6
    cov_1_reg = cov_1 + eps * np.eye(D.shape[1])
    cov_2_reg = cov_2 + eps * np.eye(D.shape[1])

    probs_1 = multivariate_normal.pdf(D, mean=mean_1, cov=cov_1_reg,␣
 ↪allow_singular=True) * phi_1
    probs_2 = multivariate_normal.pdf(D, mean=mean_2, cov=cov_2_reg,␣
 ↪allow_singular=True) * phi_2

    total = probs_1 + probs_2 + 1e-12
```

```python
    posterior_1 = probs_1 / total
    posterior_2 = probs_2 / total

    return posterior_1, posterior_2
```

[65]:

```python
def M_step(D, posterior_1, posterior_2):
    eps = 1e-6
    n_1 = max(np.sum(posterior_1), 1e-6)
    n_2 = max(np.sum(posterior_2), 1e-6)

    # mu
    mean_1 = posterior_1 @ D / n_1
    mean_2 = posterior_2 @ D / n_2

    # sigma
    diff_1 = D - mean_1
    diff_2 = D - mean_2

    R1 = np.diag(posterior_1)
    R2 = np.diag(posterior_2)

    sigma_1 = diff_1.T @ R1 @ diff_1 / n_1
    sigma_2 = diff_2.T @ R2 @ diff_2 / n_2

    sigma_1 += eps * np.eye(D.shape[1])
    sigma_2 += eps * np.eye(D.shape[1])

    # phi
    phi_1 = n_1 / (n_1 + n_2)
    phi_2 = n_2 / (n_1 + n_2)

    return mean_1, sigma_1, phi_1, mean_2, sigma_2, phi_2
```

1.2.2 Termination conditions

I will split the training set into a 80-20 split into train and validation. Then, I will continue the algorithm based on the validation log likelihood. If the validation log likelihood (that the data points are realizations of the gaussians) only changes by a very small amount (less than 1e-5), then I will terminate. Effectively, I am continuing until validation evaluation metric does not change.

[66]:

```python
# split data
from sklearn.model_selection import train_test_split

df_train, df_test = train_test_split(
    df, test_size=0.2, random_state=5
)
```

```
print(df_train.shape, df_test.shape)
```

```
(217, 3) (55, 3)
```

[67]:

```
# remove id column
D_train = df_train[['eruptions', 'waiting']].to_numpy()
D_test = df_test[['eruptions', 'waiting']].to_numpy()
```

[77]:

```
# Initialize gaussian distributions
np.random.seed(5)

mean_1 = D_train[np.random.choice(D_train.shape[0])]
cov_1 = np.random.rand(2, 2)
cov_1 = cov_1 @ cov_1.T  # make it symmetric positive definite
phi_1 = np.random.rand()

mean_2 = D_train[np.random.choice(D_train.shape[0])]
cov_2 = np.random.rand(2, 2)
cov_2 = cov_2 @ cov_2.T  # make it symmetric positive definite
phi_2 = 1 - phi_1
print(mean_1, cov_1, phi_1)
print(mean_2, cov_2, phi_2)
```

```
[ 1.75 47.  ] [[0.69415081 0.83431094]
 [0.83431094 1.09161704]] 0.08982103773760686
[ 4.933 86.   ] [[0.85537205 0.3246399 ]
 [0.3246399  0.1233298 ]] 0.9101789622623931
```

[78]:

```
print("D_train shape:", D_train.shape)
print("D_test shape:", D_test.shape)
print("mean_1 shape:", mean_1.shape)
print("mean_2 shape:", mean_2.shape)
print("cov_1 shape:", cov_1.shape)
print("cov_2 shape:", cov_2.shape)
```

```
D_train shape: (217, 2)
D_test shape: (55, 2)
mean_1 shape: (2,)
mean_2 shape: (2,)
cov_1 shape: (2, 2)
cov_2 shape: (2, 2)
```

[79]:

```
# Run EM algorithm
def run_EM(D_train, D_test, mean_1, cov_1, phi_1, mean_2, cov_2, phi_2, max_iters=100):
    log_likelihoods = []
    means_1_list = [mean_1.copy()]
    means_2_list = [mean_2.copy()]
    eps = 1e-6
    # Code adapted from ChatGPT (OpenAI 2025)
    for i in range(max_iters):
        # E-step
        posterior_1, posterior_2 = E_step(D_train, mean_1, cov_1, phi_1,␣
 ↪mean_2, cov_2, phi_2)

        # M-step
        mean_1, cov_1, phi_1, mean_2, cov_2, phi_2 = M_step(D_train,␣
 ↪posterior_1, posterior_2)

        # Compute log-likelihood
        ll_1 = multivariate_normal.pdf(D_test, mean=mean_1, cov=cov_1 + eps *␣
 ↪np.eye(D_test.shape[1])) * phi_1
        ll_2 = multivariate_normal.pdf(D_test, mean=mean_2, cov=cov_2 + eps *␣
 ↪np.eye(D_test.shape[1])) * phi_2
        log_likelihood = np.sum(np.log(ll_1 + ll_2 + 1e-12))
        log_likelihoods.append(log_likelihood)

        # Save means for analysis
        means_1_list.append(mean_1.copy())
        means_2_list.append(mean_2.copy())

        print(f"Iteration {i+1}, Log Likelihood: {log_likelihood:.6f}")
        if i > 0 and abs(log_likelihood - log_likelihoods[-2]) < 1e-5:
            break

    return mean_1, cov_1, phi_1, mean_2, cov_2, phi_2, log_likelihoods,␣
 ↪means_1_list, means_2_list
```

[80]:
```
mean_1, cov_1, phi_1, mean_2, cov_2, phi_2, log_likelihoods, means_1_list,␣
 ↪means_2_list = run_EM(
    D_train, D_test,
    mean_1, cov_1, phi_1,
    mean_2, cov_2, phi_2,
    max_iters=100
)
```

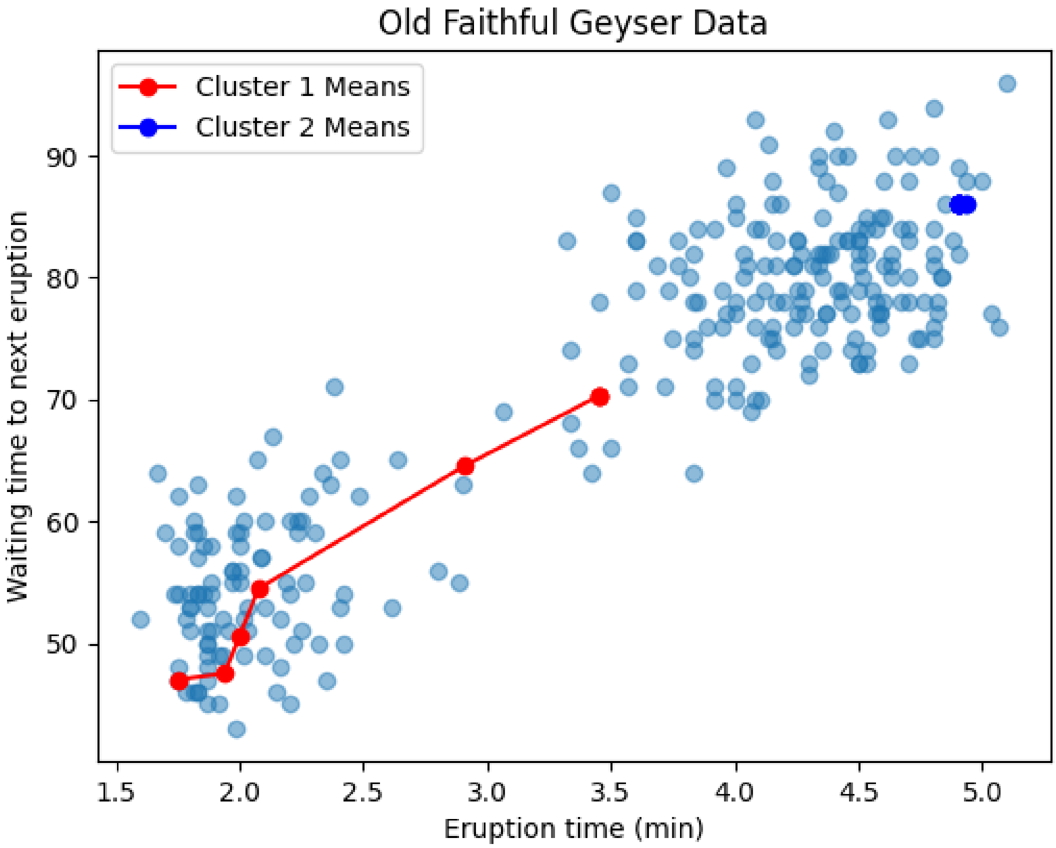
```
Iteration 1, Log Likelihood: -1373.799228
Iteration 2, Log Likelihood: -1099.324766
Iteration 3, Log Likelihood: -961.680773
Iteration 4, Log Likelihood: -270.565657
Iteration 5, Log Likelihood: -257.186770
Iteration 6, Log Likelihood: -257.186438
Iteration 7, Log Likelihood: -257.186438
```

[81]:
```
print(means_1_list)
```

```
[array([ 1.75, 47.  ]), array([ 1.93937086, 47.49746924]), array([ 1.99906204,
50.57189338]), array([ 2.07589857, 54.47415801]), array([ 2.91082821,
64.57408706]), array([ 3.44998344, 70.27105137]), array([ 3.44999032,
70.27112662]), array([ 3.44999033, 70.27112667])]
```

[82]:

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

# Code adapted from ChatGPT (OpenAI 2025)
# Scatter plot of the data
plt.scatter(df['eruptions'], df['waiting'], alpha=0.5)
plt.xlabel('Eruption time (min)')
plt.ylabel('Waiting time to next eruption')
plt.title('Old Faithful Geyser Data')

# Convert your lists of means to arrays
means_1_array = np.vstack(means_1_list)  # shape (n_iters, 2)
means_2_array = np.vstack(means_2_list)  # shape (n_iters, 2)

# Plot the trajectory of cluster 1
plt.plot(means_1_array[:,0], means_1_array[:,1],
         marker='o', color='red', linestyle='-', label='Cluster 1 Means')

# Plot the trajectory of cluster 2
plt.plot(means_2_array[:,0], means_2_array[:,1],
         marker='o', color='blue', linestyle='-', label='Cluster 2 Means')

plt.legend()
plt.show()
```



1.2.3 GMM can easily get stuck in local minima

As seen, the final clusters look very odd. For example, the red cluster 1 mean is in between the two clusters visually while the cluster 2 mean is in the edge of the upper right cluster. I think this is a local minima that was found in the GMM algorithm, so I reran the algorithm with a different initialization random seed.

[74]:
```
# Initialize gaussian distributions
np.random.seed(6)

mean_1 = D_train[np.random.choice(D_train.shape[0])]
cov_1 = np.random.rand(2, 2)
cov_1 = cov_1 @ cov_1.T  # make it symmetric positive definite
phi_1 = np.random.rand()

mean_2 = D_train[np.random.choice(D_train.shape[0])]
cov_2 = np.random.rand(2, 2)
cov_2 = cov_2 @ cov_2.T  # make it symmetric positive definite
phi_2 = 1 - phi_1
print(mean_1, cov_1, phi_1)
```

```python
print(mean_2, cov_2, phi_2)
```

```
[ 4.35 85.  ] [[0.94156194 0.13838297]
 [0.13838297 0.14110962]] 0.9850288152326052
[ 2.2 45. ] [[0.4561061  0.43842066]
 [0.43842066 0.50002887]] 0.014971184767394785
```

[75]:

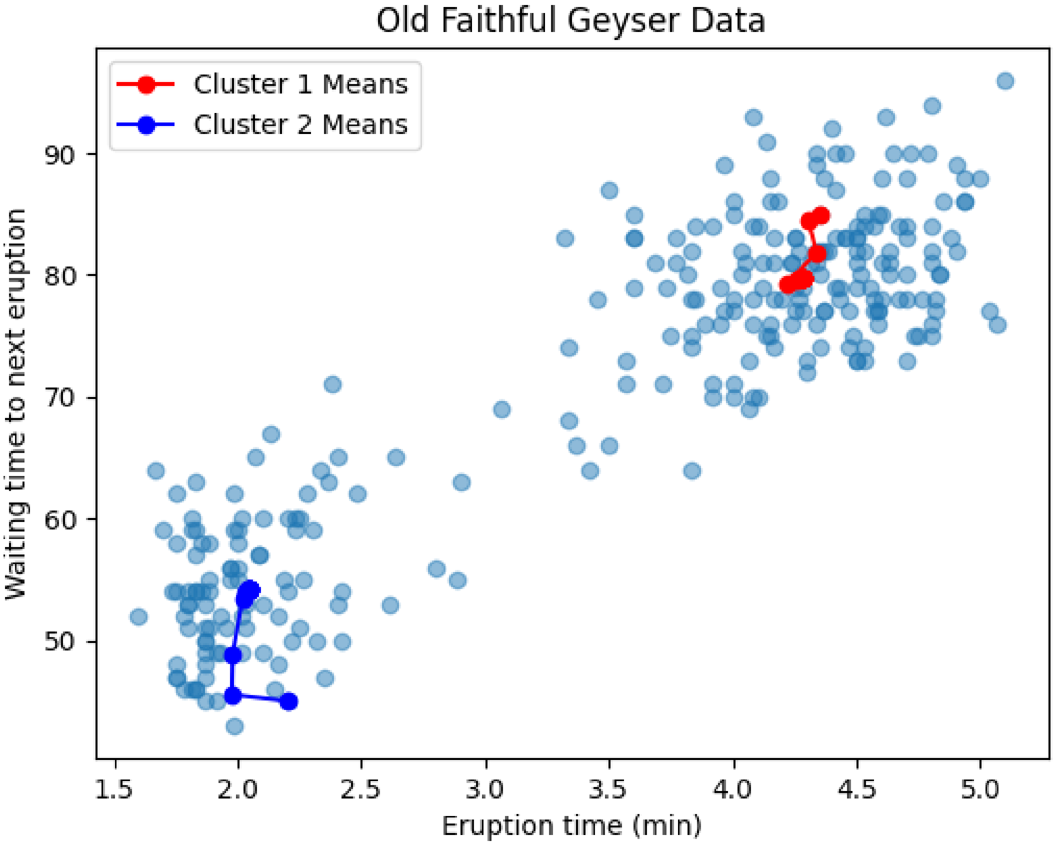
```python
mean_1, cov_1, phi_1, mean_2, cov_2, phi_2, log_likelihoods, means_1_list,␣
 ↪means_2_list = run_EM(
    D_train, D_test,
    mean_1, cov_1, phi_1,
    mean_2, cov_2, phi_2,
    max_iters=100
)
```

```
Iteration 1, Log Likelihood: -846.147703
Iteration 2, Log Likelihood: -300.933973
Iteration 3, Log Likelihood: -221.318760
Iteration 4, Log Likelihood: -218.988023
Iteration 5, Log Likelihood: -218.723442
Iteration 6, Log Likelihood: -218.768995
Iteration 7, Log Likelihood: -218.795848
Iteration 8, Log Likelihood: -218.804330
Iteration 9, Log Likelihood: -218.806560
Iteration 10, Log Likelihood: -218.807111
Iteration 11, Log Likelihood: -218.807245
Iteration 12, Log Likelihood: -218.807278
Iteration 13, Log Likelihood: -218.807285
```

[76]:

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

# Code adapted from ChatGPT (OpenAI 2025)
# Scatter plot of the data
plt.scatter(df['eruptions'], df['waiting'], alpha=0.5)
plt.xlabel('Eruption time (min)')
plt.ylabel('Waiting time to next eruption')
plt.title('Old Faithful Geyser Data')

# Convert your lists of means to arrays
means_1_array = np.vstack(means_1_list)  # shape (n_iters, 2)
means_2_array = np.vstack(means_2_list)  # shape (n_iters, 2)

# Plot the trajectory of cluster 1
plt.plot(means_1_array[:,0], means_1_array[:,1],
         marker='o', color='red', linestyle='-', label='Cluster 1 Means')

# Plot the trajectory of cluster 2
```

```
plt.plot(means_2_array[:,0], means_2_array[:,1],
         marker='o', color='blue', linestyle='-', label='Cluster 2 Means')

plt.legend()
plt.show()
```



1.3 K means clustering

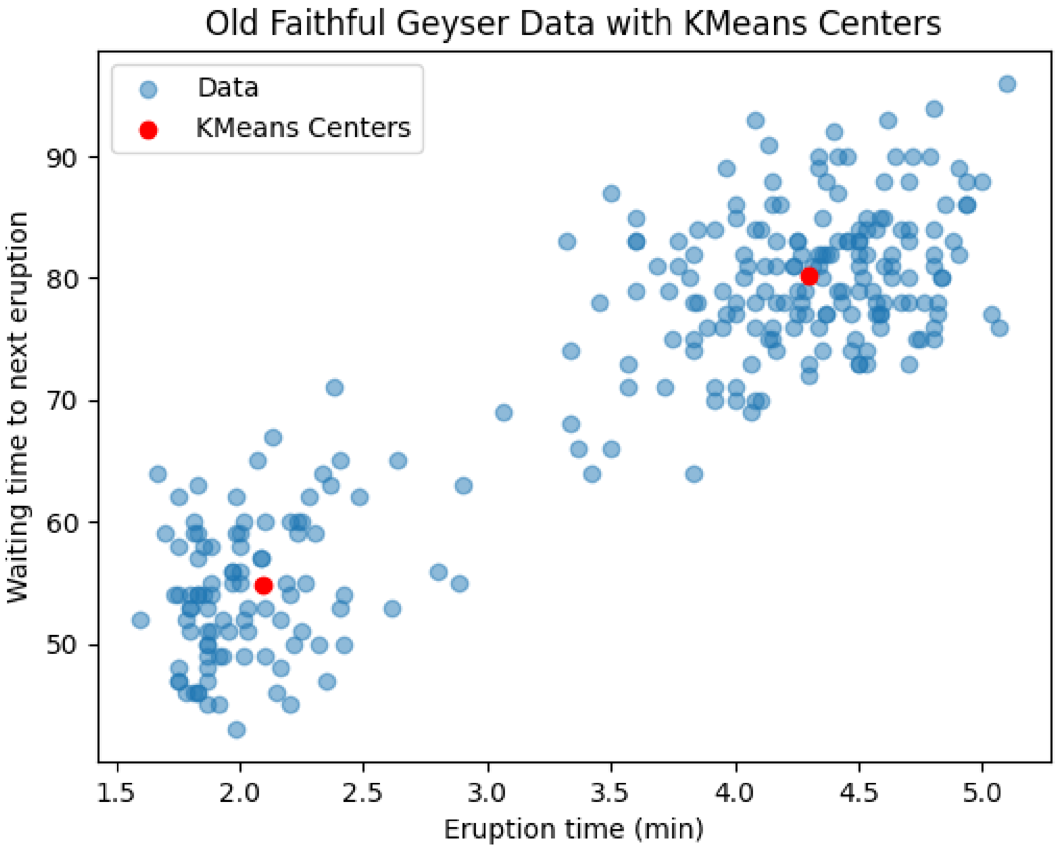
I predict that the K-means clusters would be very similar for K=2. This is because the two clusters are visually round and do not overlap very much. This means the clusters don't have too much covariance between the two feature dimensions, so K-means can still cluster well.

[51]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=5)
kmeans.fit(df[['eruptions', 'waiting']].values)
```

[51]:
```
KMeans(n_clusters=2, random_state=5)
```

[55]:

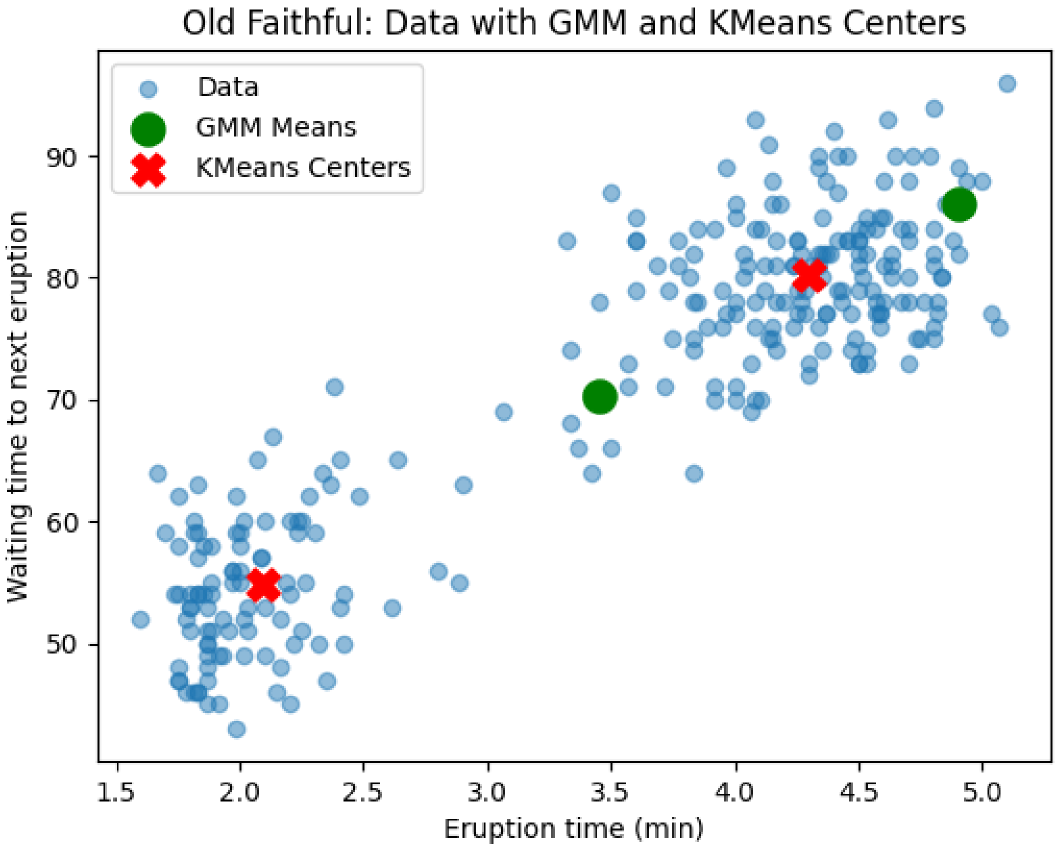
```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans

# Code adapted from ChatGPT (OpenAI 2025)
# Your data
X = df[['eruptions', 'waiting']].values

labels = kmeans.labels_
centers = kmeans.cluster_centers_

# Scatter plot of the original data
plt.scatter(df['eruptions'], df['waiting'], alpha=0.5, label='Data')

# Overlay KMeans cluster centers
plt.scatter(centers[:,0], centers[:,1], color='red', label='KMeans Centers')

plt.xlabel('Eruption time (min)')
plt.ylabel('Waiting time to next eruption')
plt.title('Old Faithful Geyser Data with KMeans Centers')
plt.legend()
plt.show()
```



As seen, there is some similarity between the K-means clustering such that both clustering algorithms put one center at each of the two visual clusters. However, the exact center of the GMM clusters are more

[83]:

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

# Scatter plot of the data
plt.scatter(df['eruptions'], df['waiting'], alpha=0.5, label='Data')

# Plot final GMM means
gmm_means = np.vstack([means_1_list, means_2_list])
plt.scatter(gmm_means[:,0], gmm_means[:,1], color='green', marker='o', s=150,␣
  ↪label='GMM Means')

# Plot KMeans centers
kmeans_centers = kmeans.cluster_centers_
plt.scatter(kmeans_centers[:,0], kmeans_centers[:,1], color='red', marker='X',␣
  ↪s=150, label='KMeans Centers')

plt.xlabel('Eruption time (min)')
plt.ylabel('Waiting time to next eruption')
plt.title('Old Faithful: Data with GMM and KMeans Centers')
plt.legend()
plt.show()
```


Old Faithful: Data with GMM and KMeans Centers

[ ]:

eigenface

November 15, 2025

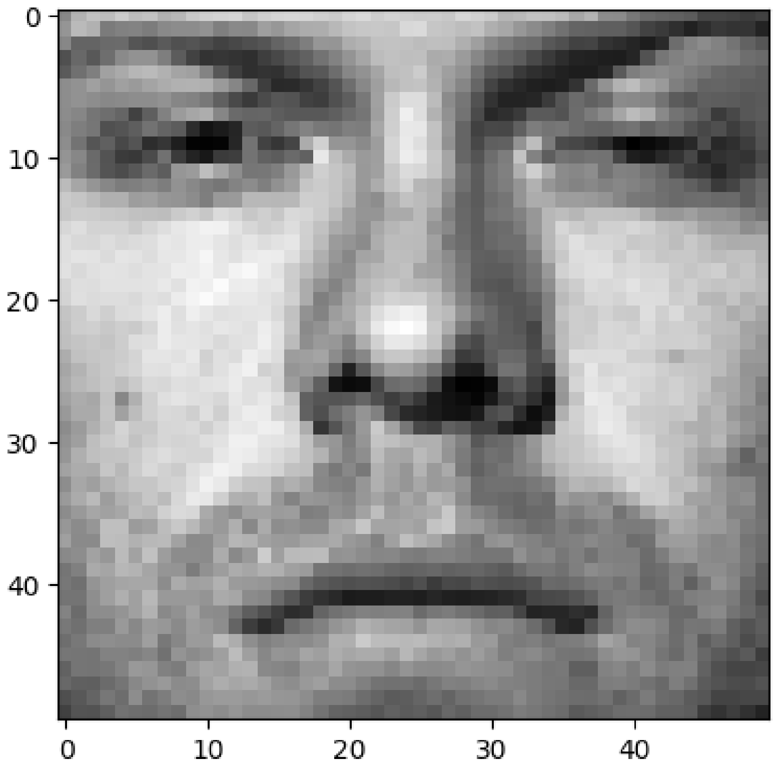
1 Eigenface for face recognition

1.1 Loading training dataset

[1]:

```
import numpy as np
import matplotlib.image as misc
from matplotlib import pylab as plt
import matplotlib.cm as cm
%matplotlib inline

train_labels, train_data = [], []
for line in open('./faces/train.txt'):
    im = misc.imread(line.strip().split()[0])
    train_data.append(im.reshape(2500,))
    train_labels.append(line.strip().split()[1])
train_data, train_labels = np.array(train_data, dtype=float), np.
  ↪array(train_labels, dtype=int)
print(train_data.shape, train_labels.shape)
plt.imshow(train_data[10, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```

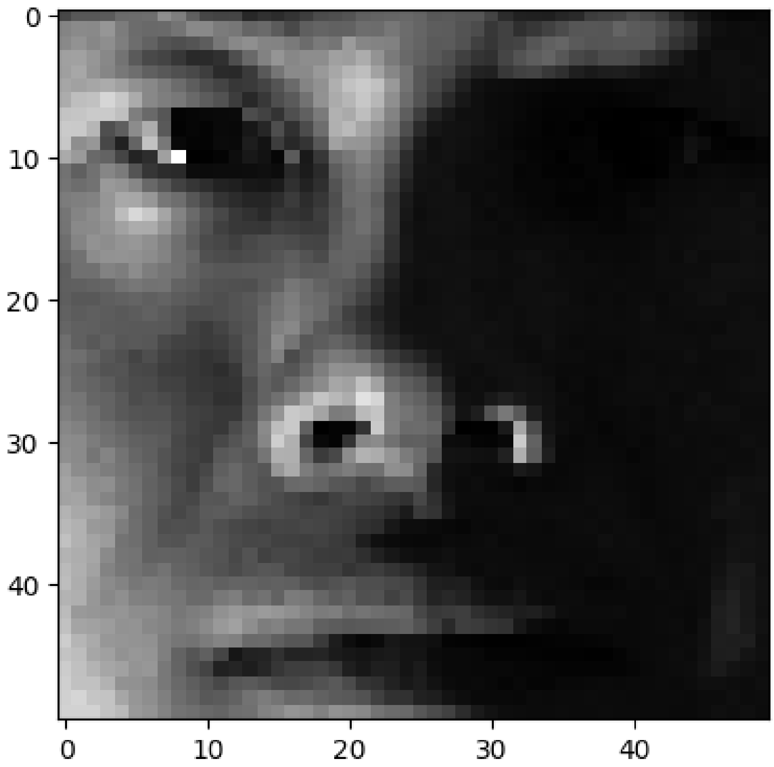
```
(540, 2500) (540,)
```



1.2 Loading testing dataset

[2]:
```
test_labels, test_data = [], []
for line in open('./faces/test.txt'):
    im = misc.imread(line.strip().split()[0])
    test_data.append(im.reshape(2500,))
    test_labels.append(line.strip().split()[1])
test_data, test_labels = np.array(test_data, dtype=float), np.
 ↪array(test_labels, dtype=int)
print(test_data.shape, test_labels.shape)
plt.imshow(test_data[10, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```

```
(100, 2500) (100,)
```



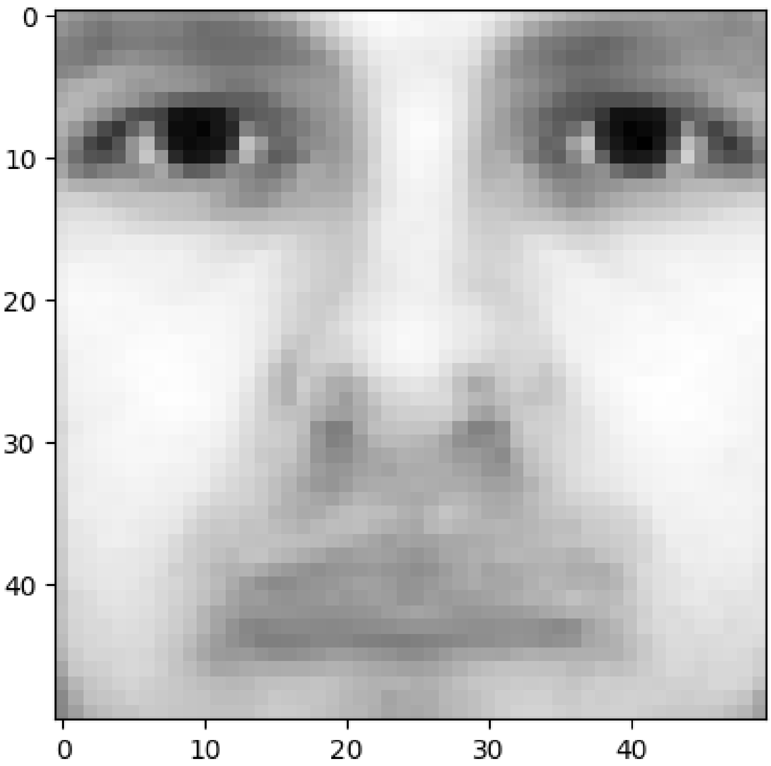
1.3 Average Face

I computed the average of the training set by summing up the values of all rows in X and dividing by the number of faces.

[3]:
```
average = np.sum(train_data, axis=0)
average /= train_data.shape[0]
print(average.shape)
```

```
(2500,)
```

[4]:
```
plt.imshow(average.reshape(50,50), cmap = cm.Greys_r)
plt.show()
```

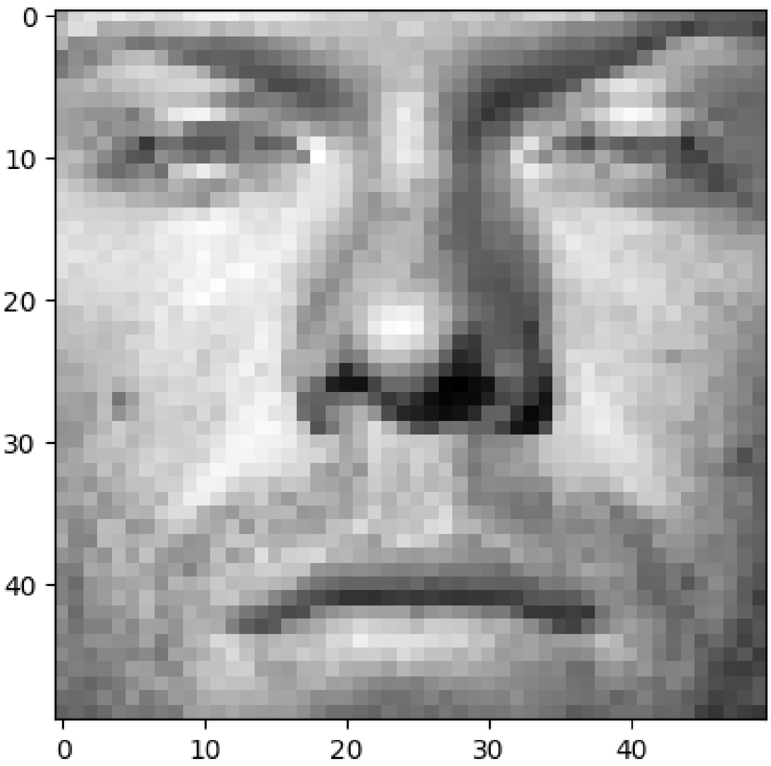


1.4 Mean Subtraction

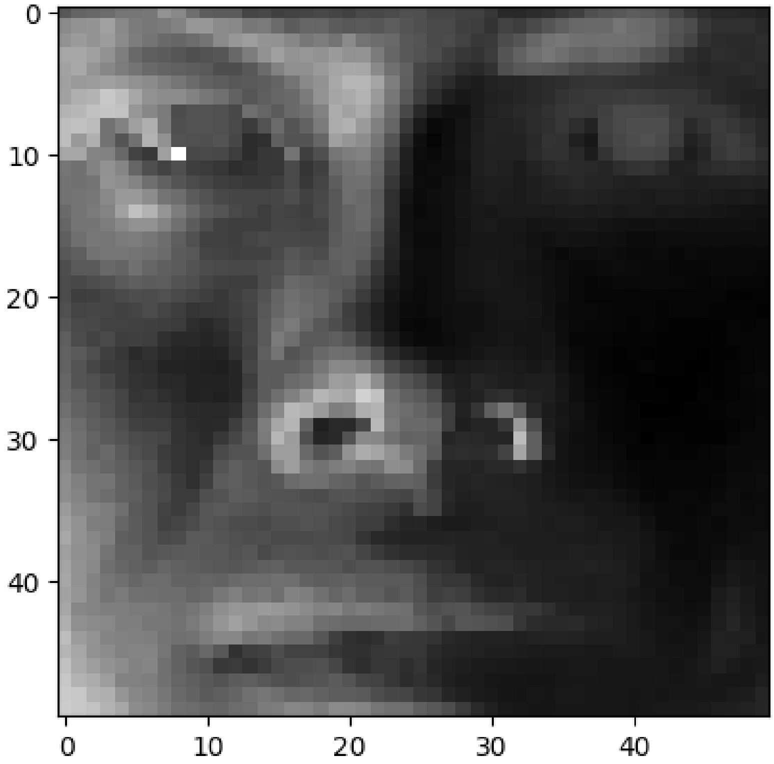
Taking this average face, I took each sample and subtracted the mean. For both the training and test set. Note that the average face was computed from only the training set (and we use the same average face for both training and test sets).

[5]:
```
train_data_subtracted = train_data - average
test_data_subtracted = test_data - average
```

[6]:
```
# train sample 10 with mean subtraction
plt.imshow(train_data_subtracted[10].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```



[7]:
```
# test sample 10 with mean subtraction
plt.imshow(test_data_subtracted[10].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```



1.5 Eigenface

I performed the eigendecomposition on each XTX to get the eigenvectors of the sample. I used the original test/train dataset, not the one from the mean subtraction.

[8]:
```
def decompose_eigen(single_image):
    X = single_image.reshape(50,50)
    A = X.T @ X
    eigvals, eigvecs = np.linalg.eig(A)
    return eigvecs
```

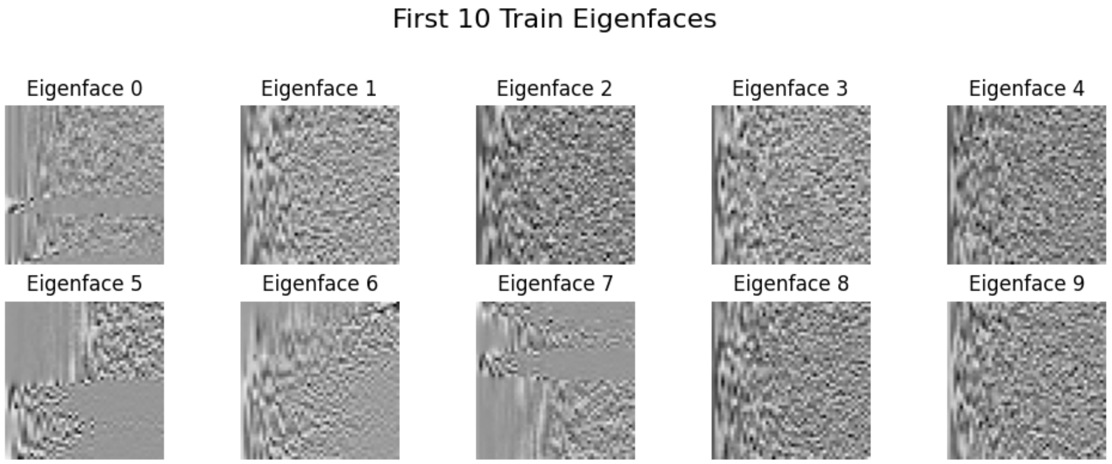
[9]:
```
train_eigenvecs = np.apply_along_axis(decompose_eigen, axis=1, arr=train_data)
test_eigenvecs = np.apply_along_axis(decompose_eigen, axis=1, arr=test_data)
```

[10]:
```
train_eigenvecs.shape
```

[10]:
```
(540, 50, 50)
```

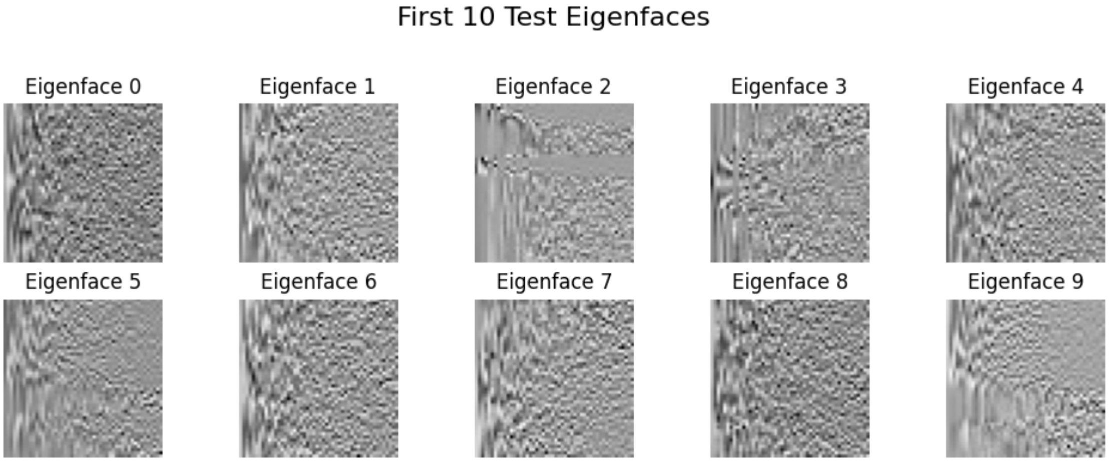
[11]:
```
# Visualize some train eigenfaces
# Code adapted from ChatGPT (OpenAI 2025)
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

```python
fig, axes = plt.subplots(2, 5, figsize=(10, 4))   # 2 rows × 5 columns

for i, ax in enumerate(axes.flat):
    img = train_eigenvecs[i].reshape(50, 50)
    ax.imshow(img, cmap=cm.Greys_r)
    ax.set_title(f"Eigenface {i}")
    ax.axis('off')   # remove axes ticks

fig.suptitle("First 10 Train Eigenfaces", fontsize=16)  # <-- ADD TITLE HERE

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```



[13]:

```python
# Visualize some test eigenfaces
# Code adapted from ChatGPT (OpenAI 2025)
import matplotlib.pyplot as plt
import matplotlib.cm as cm

fig, axes = plt.subplots(2, 5, figsize=(10, 4))   # 2 rows × 5 columns

for i, ax in enumerate(axes.flat):
    img = test_eigenvecs[i].reshape(50, 50)
    ax.imshow(img, cmap=cm.Greys_r)
    ax.set_title(f"Eigenface {i}")
    ax.axis('off')   # remove axes ticks

fig.suptitle("First 10 Test Eigenfaces", fontsize=16)  # <-- ADD TITLE HERE

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```



1.6 Eigenface Feature

This function takes the top r eigenfaces of the previously calculculated eigenfaces and multiplies them to z, the original face images.

[30]:
```
print(train_eigenvecs.shape)
print(train_data.shape)
```

```
(540, 50, 50)
(540, 2500)
```

[22]:
```
train_eigenvecs_flat = train_eigenvecs.reshape(train_eigenvecs.shape[0], -1)
test_eigenvecs_flat = test_eigenvecs.reshape(test_eigenvecs.shape[0], -1)
```

[39]:
```
print(train_eigenvecs_flat.shape) # my train_eigenvecs is "VT"
print(train_eigenvecs_flat[:23,:].shape)
```

```
(540, 2500)
(23, 2500)
```

[52]:
```
def get_f_train(r):
    f = train_eigenvecs_flat[:r,:] @ train_data.T
    return f.T

def get_f_test(r):
    f = train_eigenvecs_flat[:r,:] @ test_data.T
    return f.T
```

[53]:
```
get_f_train(10).shape
```

[53]:
```
(540, 10)
```

1.7 Face Recognition

I used logistic regression from scikit learn with a “one-vs-rest” logistic regression. This required using OveVsRestClassifier. I tested many r values from 1,2,5,10,20,50,100,150,200.

[54]:
```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
```

[55]:
```
model = OneVsRestClassifier(estimator=LogisticRegression())
```

[63]:
```
from sklearn.metrics import accuracy_score
rs = [1,2,5,10,20,50,100,150,200]
accs = {}
for r in rs:
    X_train = get_f_train(r)
    X_test = get_f_test(r)
    model.fit(X_train, train_labels)
    print(X_train.shape, X_test.shape)


    Y_preds = model.predict(X_test)
    acc =  accuracy_score(test_labels, Y_preds)
    accs[r] = acc
```

```
(540, 1) (100, 1)
(540, 2) (100, 2)
(540, 5) (100, 5)
(540, 10) (100, 10)
(540, 20) (100, 20)
```

```
/home/jiwonjjeong/cornell-tech/CT-applied-machine-
learning/venv/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed
to converge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=100).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
(540, 50) (100, 50)
(540, 100) (100, 100)
(540, 150) (100, 150)
```

```
(540, 200) (100, 200)
```

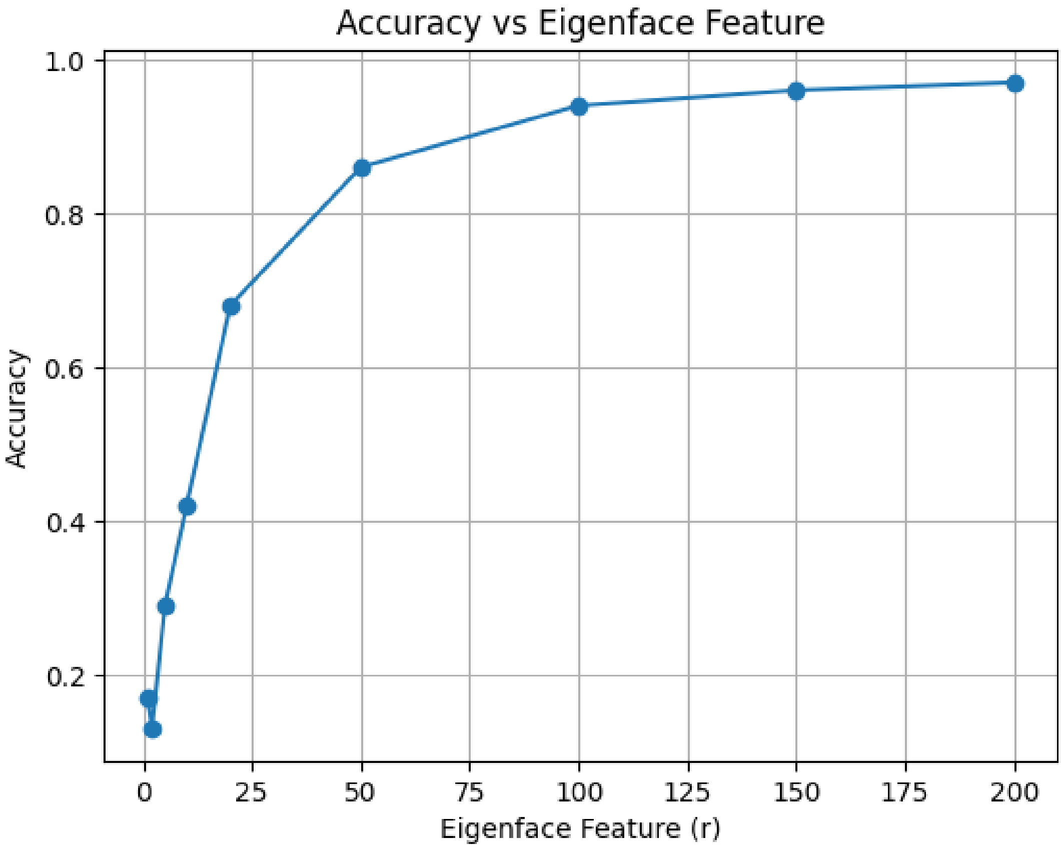
[64]:

```
print(accs)
```

```
{1: 0.17, 2: 0.13, 5: 0.29, 10: 0.42, 20: 0.68, 50: 0.86, 100: 0.94, 150: 0.96,
200: 0.97}
```

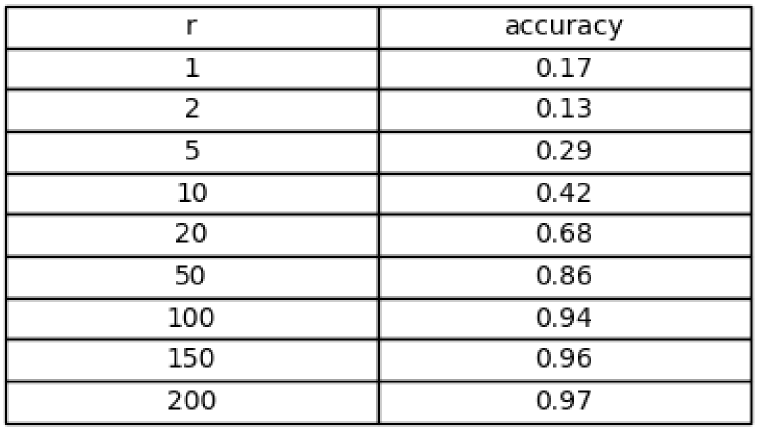
[66]:

```
import matplotlib.pyplot as plt

# Code adapted from ChatGPT (OpenAI 2025)
# assume accs is your dictionary: {1: acc, 2: acc, ...}

keys = sorted(accs.keys())
values = [accs[k] for k in keys]

plt.plot(keys, values, marker='o')
plt.xlabel("Eigenface Feature (r)")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Eigenface Feature")
plt.grid(True)
plt.show()
```



[68]:

```
import matplotlib.pyplot as plt

# Code adapted from ChatGPT (OpenAI 2025)
# Sort keys so rows appear in order
keys = sorted(accs.keys())
values = [accs[k] for k in keys]

fig, ax = plt.subplots(figsize=(5, len(keys) * 0.4))
ax.axis('off')

# Build table data
table_data = [["r", "accuracy"]]
for k, v in zip(keys, values):
    table_data.append([k, v])

# Create table
table = ax.table(
    cellText=table_data,
    loc='center',
    cellLoc='center'
)

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1, 1.3)

plt.show()
```



r	accuracy
1	0.17
2	0.13
5	0.29
10	0.42
20	0.68
50	0.86
100	0.94
150	0.96
200	0.97

b)

From Bayes Rule

$$P_\theta(z=k|x) = \frac{p_\theta(z=k,x)}{P_\theta(x)}$$

Represent $P_\theta(x)$ by sum over all cluster assignments

$$P_\theta(z=k|x) = \frac{P_\theta(z=k,x)}{\sum_{l=1}^{K} P_\theta(x|z=l)\, P_\theta(z=l)}$$

$$= \frac{P_{\mu,\Sigma}(x|z=k)\cdot P_\phi(z=k)}{\sum P_{\mu,\Sigma}(x|z=l)\, P_\phi(z=l)}$$

c)

$$\overset{\text{for cluster k}}{\mu_k} = \frac{\overset{\text{sum over whole D}}{\sum_{i=1}^{n}} P(z=k|x^{(i)})\, x^{(i)}}{n_k}$$

$$\Sigma_k = \frac{\sum_{i=1}^{n} P(z=k|x^{(i)})(x^{(i)}-\mu_k)(x^{(i)}-\mu_k)^T}{n_k}$$

$$n_k = \sum_{i=1}^{n} P(z=k|x^{(i)})$$

$$\phi_k = \frac{n_k}{n}$$

① $X = UDV^T$

$X^T = (UDV^T)^T \quad (AB)^T = B^T A^T$

$X^T = V^{TT} D^T U^T$

$X^T = V D^T U^T$

Thus

$X^T X = V D^T U^T U D V^T \quad U^T U = I$

$= V D^T I D V^T \quad D^T I = D^T$

$= V D^T D V^T$

Since D is a diagonal matrix, and nonneg
D^T is diagonal. and nonneg
So $D^T D$ is diagonal and nonneg.

$X^T X = V M V^T$

where $M = D^T D$ and M is an $n \times n$ diagonal
and V is an orthonomal $n \times n$ matrix
and V^T is an orthonormal $n \times n$ matrix
Thus $X^T X = VMV^T$ is an eigendecomposition

②

a) $M^TM =$

$$\begin{bmatrix} 39 & 57 & 60 \\ 57 & 118 & 53 \\ 60 & 53 & 127 \end{bmatrix}$$

$$MM^T = \begin{bmatrix} 10 & 9 & 26 & 3 & 26 \\ 9 & 62 & 8 & -5 & 85 \\ 26 & 8 & 72 & 10 & 50 \\ 3 & -5 & 10 & 2 & 1 \\ 26 & 85 & 50 & 1 & 138 \end{bmatrix}$$

b) M^TM:

$$\begin{bmatrix} 0.426 \\ 0.615 \\ 0.663 \end{bmatrix} \begin{bmatrix} 0.904 \\ -0.302 \\ -0.302 \end{bmatrix} \begin{bmatrix} -0.014 \\ -0.729 \\ 0.685 \end{bmatrix}$$

MM^T

$$\begin{bmatrix} -0.165 \\ -0.471 \\ -0.336 \\ -0.003 \\ -0.798 \end{bmatrix} \begin{bmatrix} -0.955 \\ -0.035 \\ 0.271 \\ 0.044 \\ -0.104 \end{bmatrix} \begin{bmatrix} 0.245 \\ -0.453 \\ 0.829 \\ 0.169 \\ -0.133 \end{bmatrix} \begin{bmatrix} -0.064 \\ 0.750 \\ 0.328 \\ 0.045 \\ -0.568 \end{bmatrix} \begin{bmatrix} -0.106 \\ -0.047 \\ -0.174 \\ 0.970 \\ 0.119 \end{bmatrix}$$

d) $M = UDV^T$

but also

$$M^TM = VD^TDV^T$$

$$MM^T = UDD^TU^T$$

from eigenvales of M^TM,

$$V^T = \begin{bmatrix} 0.426 & \cancel{0.904} & -0.014 \\ 0.615 & \cancel{-0.302} & -0.729 \\ 0.663 & \cancel{-0.302} & 0.685 \end{bmatrix}^T$$

zero eigenvalues

from eigenvalues of MM^T

$$U = \begin{bmatrix} -0.165 & \cancel{-0.955} & 0.245 & \cancel{-0.064} & \cancel{-0.106} \\ -0.471 & \cancel{-0.035} & -0.453 & \cancel{0.750} & \cancel{-0.047} \\ -0.336 & \cancel{0.271} & 0.829 & \cancel{0.328} & \cancel{-0.174} \\ -0.003 & \cancel{0.044} & 0.169 & \cancel{0.045} & \cancel{0.970} \\ -0.798 & \cancel{0.104} & -0.133 & \cancel{-0.568} & \cancel{0.119} \end{bmatrix}$$

$$M = \begin{bmatrix} -0.165 & 0.245 \\ -0.471 & -0.453 \\ -0.336 & 0.829 \\ -0.003 & 0.169 \\ -0.798 & -0.133 \end{bmatrix} \begin{bmatrix} \sqrt{214.67} & 0 \\ 0 & \sqrt{69.330} \end{bmatrix} \begin{bmatrix} 0.426 & 0.615 & 0.663 \\ -0.014 & -0.729 & 0.685 \end{bmatrix}$$

$$N = \begin{bmatrix} -0.165 & \cancel{0.245} \\ -0.471 & \cancel{-0.473} \\ -0.336 & \cancel{0.829} \\ -0.003 & \cancel{0.169} \\ -0.798 & \cancel{-0.133} \end{bmatrix} \begin{bmatrix} \sqrt{214.67} & 0 \\ 0 & \cancel{\sqrt{69.330}} \end{bmatrix} \begin{bmatrix} 0.426 & 0.615 & 0.663 \\ \cancel{-0.014} & \cancel{-0.729} & \cancel{0.684} \end{bmatrix}$$

$$N = \begin{bmatrix} 1.029 & 1.486 & 1.603 \\ 2.944 & 4.249 & 4.584 \\ 2.100 & 3.031 & 3.270 \\ 0.020 & 0.0297 & 0.0321 \\ 4.983 & 7.192 & 7.758 \end{bmatrix}$$

svd

November 15, 2025

1 SVD of Rank Deficient Matrix

```
[1]: import numpy as np
```

```
[32]: M = np.array([[1,0,3],
          [3,7,2],
          [2,-2,8],
          [0,-1,1],
          [5,8,7],
          ])
      M
```

```
[32]: array([[ 1,  0,  3],
             [ 3,  7,  2],
             [ 2, -2,  8],
             [ 0, -1,  1],
             [ 5,  8,  7]])
```

```
[33]: A = M.T @ M
      A
```

```
[33]: array([[ 39,  57,  60],
             [ 57, 118,  53],
             [ 60,  53, 127]])
```

```
[34]: B = M @ M.T
      B
```

```
[34]: array([[ 10,   9,  26,   3,  26],
             [  9,  62,   8,  -5,  85],
             [ 26,   8,  72,  10,  50],
             [  3,  -5,  10,   2,  -1],
             [ 26,  85,  50,  -1, 138]])
```

1.1 Eigenvalues

[35]:

```
eigvals, eigvecs = np.linalg.eig(A)
eigvals, eigvecs
```

[35]:

```
(array([ 2.14670489e+02, -1.50990331e-14,  6.93295108e+01]),
 array([[ 0.42615127,  0.90453403, -0.01460404],
        [ 0.61500884, -0.30151134, -0.72859799],
        [ 0.66344497, -0.30151134,  0.68478587]]))
```

[36]:

```
eigvals, eigvecs = np.linalg.eig(B)
eigvals, eigvecs
```

[36]:

```
(array([ 2.14670489e+02, -5.99520433e-15,  6.93295108e+01,  1.23720729e-14,
        -7.24621456e-16]),
 array([[-0.16492942, -0.95539856,  0.24497323, -0.06464508, -0.10671808],
        [-0.47164732, -0.03481209, -0.45330644,  0.75010839, -0.04684757],
        [-0.33647055,  0.27076072,  0.82943965,  0.3284142 , -0.17364364],
        [-0.00330585,  0.04409532,  0.16974659,  0.04591676,  0.97063121],
        [-0.79820031,  0.10366268, -0.13310656, -0.5685017 ,  0.11890961]]))
```

1.2 SVD

[59]:

```
eigvals, eigvecs = np.linalg.eig(A)
V = eigvecs[:, (0,2)]
eigvals, eigvecs = np.linalg.eig(B)
U = eigvecs[:, (0,2)]
M = U @ np.array([[-np.sqrt(2.14670489e+02), 0],[0,-np.sqrt(6.93295108e+01)]]) @ V.T
M
```

[59]:

```
array([[ 1.05957729,  2.97232071,  0.20641116],
       [ 2.88975624,  1.4999211 ,  7.16934763],
       [ 2.20171904,  8.063796  , -1.45863887],
       [ 0.04128223,  1.05957729, -0.93573059],
       [ 4.96762859,  6.38498522,  8.51790055]])
```

1.3 1-D Approximation

[58]:

```
V = V[:,0:1]
U = U[:,0:1]
N = U @ np.array([[-np.sqrt(2.14670489e+02)]]) @ V.T
N
```

[58]:

```
array([[1.02978864, 1.48616035, 1.60320558],
       [2.94487812, 4.24996055, 4.58467382],
       [2.10085952, 3.031898  , 3.27068057],
       [0.02064112, 0.02978864, 0.0321347 ],
       [4.98381429, 7.19249261, 7.75895027]])
```

[61]:

```
eigvals, eigvecs = np.linalg.eig(A)
V = eigvecs[:, (0,2)]
eigvals, eigvecs = np.linalg.eig(B)
U = eigvecs[:, (0,2)]
M = U @ np.array([[-np.sqrt(2.14670489e+02), 0],[0,0]]) @ V.T
M
```

[61]:

```
array([[1.02978864, 1.48616035, 1.60320558],
       [2.94487812, 4.24996055, 4.58467382],
       [2.10085952, 3.031898  , 3.27068057],
       [0.02064112, 0.02978864, 0.0321347 ],
       [4.98381429, 7.19249261, 7.75895027]])
```

[ ]: