I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted.

X

Went to office hours when I gave/received help.

2)a) $$\sum_{i=1}^{n} \frac{\omega_i^{T} \exp(-\beta_T y_i G_T(x_i))}{\cancel{Z_T}} = \cancel{1}\, Z_T$$

$$Z_T = \sum \omega_T \exp(\beta_T) + \sum \omega_T \exp(-\beta_T)$$

$$\boxed{Z_T = \sqrt{\frac{err_T}{1-err_T}}\,(1-err_T) + \sqrt{\frac{1-err_T}{err_T}}\,err_T}$$

b) $$\omega_i^{2} = \frac{\omega_i^{(1)} \exp(-\beta_1 y_i G_1(x_i))}{Z_1}$$

$$\omega_i^{(1)} = \frac{1}{n}$$

$\vdots$

$$\frac{1}{n} \prod_{j=1}^{J} \frac{\exp(\underbrace{-\beta_j y_i G_j(x_i)}_{-yM(x_i)})}{Z_j}$$

$$\frac{1}{n} \prod_{j=1}^{J} \frac{\exp(-yM(x_i))}{Z_j}$$

c) $\sum_{i=1}^{n} e^{-y_i (M(x_i))} = \sum e^{-y_i (M(x_i))} + \sum e^{-y_i M(x_i)}$

$$\frac{2}{2} \sum e^{-y_i (M(x_i))} \geq \frac{B}{2}$$

$$\boxed{\sum e^{-y_i (M(x_i))} \geq B}$$

d)

$Z_T \leq 2\sqrt{0.49 \cdot (1-0.49)}$ $\boxed{Z_T \leq 0.9997}$

$$\frac{1}{n \prod Z_T} e^{-y_i M(x_i)}$$

gets larger so whole thing goes to 0, so
contribution will be very small so $B=0$

e) AdaBoost essentially looks at a small subset of points, and only follows that in terms of optimization.

3a:

a) $\boxed{\sum_{a=1}^{b} U_{ia} D_{aa} V^T_{ja}}$

3b:

b) X_i should be ith row of U

Y_i should be jth row of V

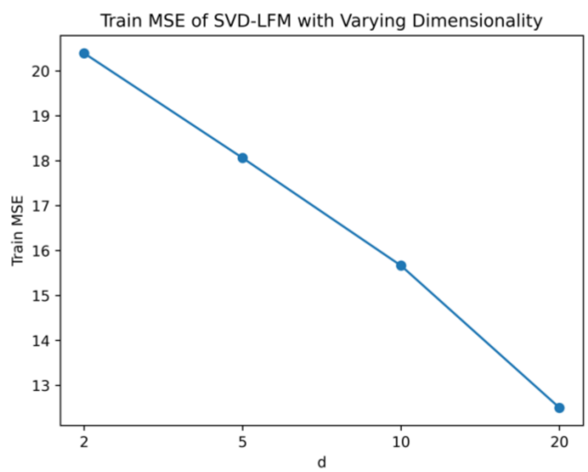
3c:

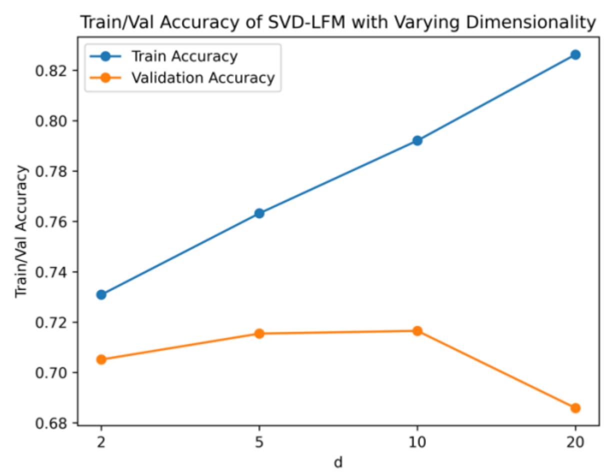
```
def svd_lfm(R):

    a = np.isnan(R)
    b = np.where(a, 0, R)

    c, d, e = svd(b, full_matrices=False)
    f = sqrt(d)

    g = np.diag(f[:c.shape[1]])
    user_vecs = c @ g

    h = (np.diag(f[:e.shape[0]]) @ e)
    movie_vecs = h.T

    return user_vecs, movie_vecs
```

3d:

```python
def get_train_mse(R, user_vecs, movie_vecs):
    a = 0
    b = 0
    for i in range(0, R.shape[0]):
        for j in range(0, R.shape[1]):
            if not np.isnan(R[i, j]):
                c = np.dot(user_vecs[i], movie_vecs[j])
                a += (R[i, j] - c) ** 2
                b += 1
    if b:
        return a / b
    else:
        return float('inf')
```

3e:





20 gives an optimal performance.

3f:

$$\frac{\partial L}{\partial x_i} = 2\sum_{j\in S_i} (x_i \cdot y_j - R_{ij}) y_j + 2\lambda x_i = 0$$

$$\sum_{j\in S_i} (x_i \cdot y_j) y_j - \sum_{j\in S_i} R_{ij} y_j + \lambda x_i = 0$$

$$x_i = \sum_{j\in S_i} y_j y_j^T + \lambda I = \boxed{\sum_{ij\in S} (R_{ij} - x_i \cdot y_j) y_j}$$

```
Start optim, train MSE: 30.49, train accuracy: 0.5950, val accuracy: 0.5799
Iteration 1, train MSE: 14.73, train accuracy: 0.7622, val accuracy: 0.6331
Iteration 2, train MSE: 12.82, train accuracy: 0.7863, val accuracy: 0.6740
Iteration 3, train MSE: 11.70, train accuracy: 0.7996, val accuracy: 0.6930
Iteration 4, train MSE: 11.18, train accuracy: 0.8061, val accuracy: 0.7108
Iteration 5, train MSE: 10.88, train accuracy: 0.8095, val accuracy: 0.7100
Iteration 6, train MSE: 10.70, train accuracy: 0.8115, val accuracy: 0.7073
Iteration 7, train MSE: 10.59, train accuracy: 0.8128, val accuracy: 0.7106
Iteration 8, train MSE: 10.52, train accuracy: 0.8137, val accuracy: 0.7081
Iteration 9, train MSE: 10.47, train accuracy: 0.8143, val accuracy: 0.7106
Iteration 10, train MSE: 10.43, train accuracy: 0.8149, val accuracy: 0.7117
Iteration 11, train MSE: 10.40, train accuracy: 0.8152, val accuracy: 0.7111
Iteration 12, train MSE: 10.38, train accuracy: 0.8153, val accuracy: 0.7111
Iteration 13, train MSE: 10.37, train accuracy: 0.8155, val accuracy: 0.7100
Iteration 14, train MSE: 10.35, train accuracy: 0.8155, val accuracy: 0.7100
Iteration 15, train MSE: 10.34, train accuracy: 0.8157, val accuracy: 0.7117
Iteration 16, train MSE: 10.33, train accuracy: 0.8158, val accuracy: 0.7133
Iteration 17, train MSE: 10.32, train accuracy: 0.8159, val accuracy: 0.7130
Iteration 18, train MSE: 10.31, train accuracy: 0.8160, val accuracy: 0.7133
Iteration 19, train MSE: 10.30, train accuracy: 0.8161, val accuracy: 0.7117
Iteration 20, train MSE: 10.30, train accuracy: 0.8161, val accuracy: 0.7138
```

```python
def update_user_vecs(user_vecs, movie_vecs, R, user_rated_idxs):

    a = 0.1
    for i in range(0, R.shape[0]):
        d = user_rated_idxs[i]
        if d.size > 0:
            temp1 = movie_vecs[d].T
            temp2 = movie_vecs[d]
            temp3 = np.eye(best_d)
            c = temp1 @ temp2 + a * temp3

            d = movie_vecs[d].T @ R[i, d]
            user_vecs[i] = np.linalg.solve(c, d)

    return user_vecs

# Part (f): Function to update user vectors
def update_movie_vecs(user_vecs, movie_vecs, R, movie_rated_idxs):

    a = 0.1
    for i in range(0, R.shape[1]):
        d = movie_rated_idxs[i]
        if d.size > 0:
            temp1 = user_vecs[d].T
            temp2 = user_vecs[d]
            temp3 = np.eye(best_d)
            c = temp1 @ temp2 + a * temp3

            d = user_vecs[d].T @ R[d, i]
            movie_vecs[i] = np.linalg.solve(c, d)
    return movie_vecs
```

4a:

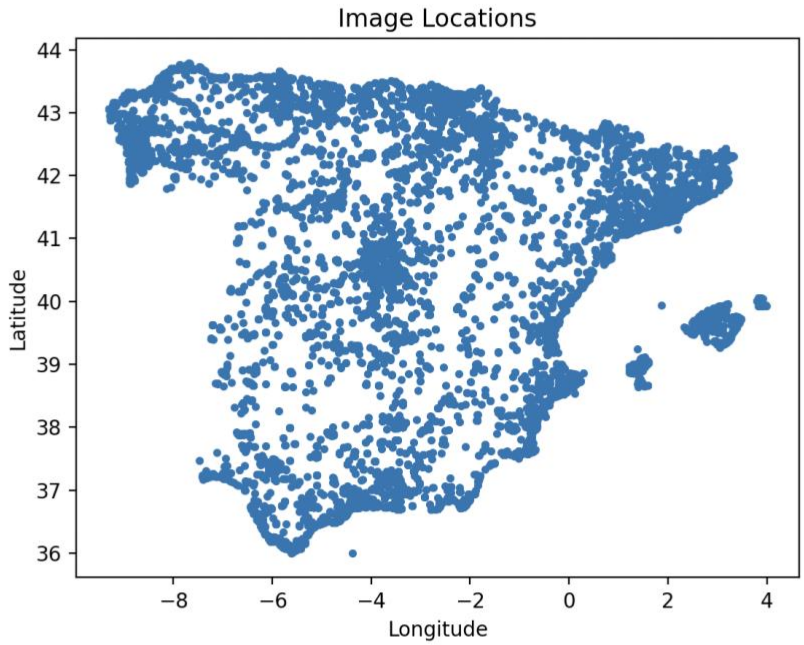


Image Locations
44
43
42
41
40
39
38
37
36
Latitude
−8
−6
−4
−2
0
2
4
Longitude

Image Features by Longitude after PCA
15
10
5
0
−5
−10
−10
−5
0
5
10
15
2
0
−2
−4
−6
−8
Longitude

4b:

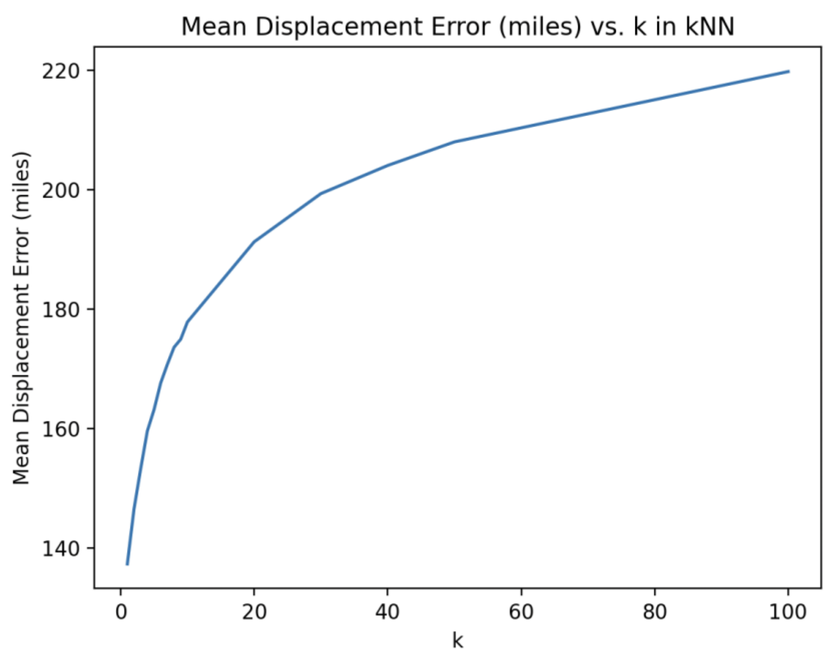
```
##### TODO(b): Your Code Here #####
a = np.where(test_files == '53633239060.jpg')[0][0]
b, c = knn.kneighbors([test_features[a]])

d = train_files[c[0]]
e = train_labels[c[0]]
print("Nearest Images:", d)
print("The Coordinates of The Nearest Images:", e)
```

They are all correct.

4c: MDE is 209.86266 miles.

```
a, b = np.mean(train_labels, axis=0)

temp12 = test_labels.shape
c = np.full(temp12, [a, b])
temp13 = ((test_labels[:, 0] - a) * 69)
temp14 = ((test_labels[:, 1] - b) * 52)
d = np.mean(np.sqrt(temp13 ** 2 + temp14 ** 2))
print("Baseline MDE: " , d)
```

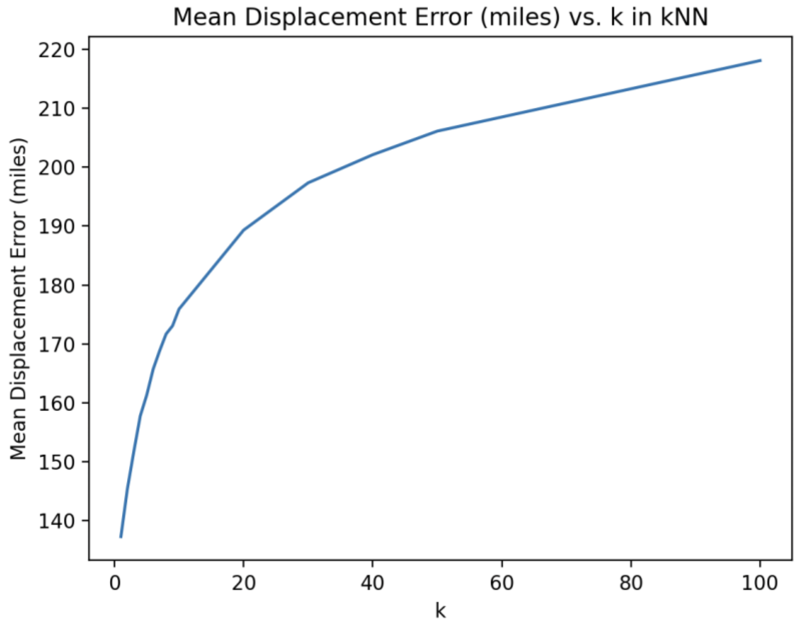
4d:



0 is lowest error with lowest error of 140 miles.

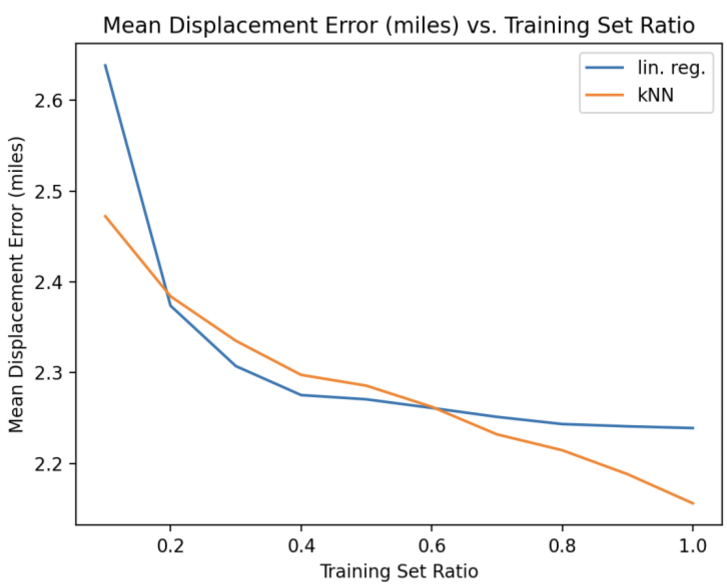
4e:
At k = 0 we have high variance but low bias. At k = 100 we have low variance and high bias. At intermediate k values, the bias and variance are intermediate as well.

4f:



What is the best value of k? 0.
What is the MDE in miles? 140 miles.
How does performance compare to part (e)? The same.

4g:



I would expect kNN to continue improving while the linear regression seems to be leveling out.

```python
a1 = train_features[:num_samples]
a2 = train_labels[:num_samples]

b1 = LinearRegression()
b1.fit(a1, a2)
b2 = b1.predict(test_features)

temp32 = np.square(test_labels - b2)
temp31 = np.mean(temp32)
e_lin = np.sqrt(temp31)


temp33 = NearestNeighbors(n_neighbors=5)
c1 = temp33.fit(a1, a2)

b, c = c1.kneighbors(test_features)

temp34 = [a2[idx].mean(axis=0) for idx in c]
d1 = np.array(temp34)

temp35 = np.mean(np.square(test_labels - d1))
e_nn = np.sqrt(temp35)
```