CS 289A – Spring 2023 – Homework 7

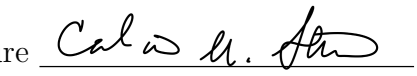
Colin Skinner, [redacted]

1 Honor Code

I did not collaborate with any students. I did refer to ChatGPT frequently.

"I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."

Signed Colin Skinner

Signature  Date 5/5/2023

2

2.1 a)

Note that

$$\sum_{i=1}^{n} w_i^{(T)} = \sum_{y_i = G_T(X_i)} w_i^{(T)} + \sum_{y_i \neq G_T(X_i)} w_i^{(T)}$$

and if we have the condition that $\sum_{i=1}^{n} w_i^{(T)} = 1$, then

$$\sum_{y_i = G_T(X_i)} w_i^{(T)} = 1 - \sum_{y_i \neq G_T(X_i)} w_i^{(T)}$$

and also

$$\text{err}_T = \frac{\sum_{y_i \neq G_T(X_i)} w_i^{(T)}}{\sum_{i=1}^{n} w_i^{(T)}}$$

$$= \sum_{y_i \neq G_T(X_i)} w_i^{(T)}$$

Additionally, if $\sum_{i=1}^{n} w_i^{(T)} = 1$, then

$$\sum_{y_i = G_T(X_i)} w_i^{(T)} = 1 - \text{err}_T$$

If we also have the condition that $\sum_{i=1}^{n} w_i^{(T+1)} = 1$, and we define

$$w_i^{(T+1)} = \frac{w_i^{(T)} \exp\left(-\beta_T y_i G_T(X_i)\right)}{Z_T}$$

then

$$\sum_{i=1}^{n} w_i^{(T+1)} = \sum_{i=1}^{n} \frac{w_i^{(T)} \exp\left(-\beta_T y_i G_T(X_i)\right)}{Z_T}$$

$$1 = \frac{1}{Z_T} \sum_{i=1}^{n} w_i^{(T)} \exp\left(-\beta_T y_i G_T(X_i)\right)$$

$$Z_T = \sum_{y_i = G_T(X_i)} w_i^{(T)} \exp\left(-\beta_T y_i G_T(X_i)\right) + \sum_{y_i \neq G_T(X_i)} w_i^{(T)} \exp\left(-\beta_T y_i G_T(X_i)\right)$$

$$= \sum_{y_i = G_T(X_i)} w_i^{(T)} \exp\left(-\beta_T\right) + \sum_{y_i \neq G_T(X_i)} w_i^{(T)} \exp\left(\beta_T\right)$$

$$= \exp\left(\ln \sqrt{\frac{\text{err}_T}{1 - \text{err}_T}}\right) \sum_{y_i = G_T(X_i)} w_i^{(T)} + \exp\left(\ln \sqrt{\frac{1 - \text{err}_T}{\text{err}_T}}\right) \sum_{y_i \neq G_T(X_i)} w_i^{(T)}$$

$$= \sqrt{\frac{\text{err}_T}{1 - \text{err}_T}} \sum_{y_i = G_T(X_i)} w_i^{(T)} + \sqrt{\frac{1 - \text{err}_T}{\text{err}_T}} \sum_{y_i \neq G_T(X_i)} w_i^{(T)}$$

$$= \sqrt{\frac{\text{err}_T}{1 - \text{err}_T}} \left(1 - \text{err}_T\right) + \sqrt{\frac{1 - \text{err}_T}{\text{err}_T}} \left(\text{err}_T\right)$$

$$= \sqrt{\text{err}_T \left(1 - \text{err}_T\right)} + \sqrt{\text{err}_T \left(1 - \text{err}_T\right)}$$

$$= 2\sqrt{\text{err}_T \left(1 - \text{err}_T\right)}$$

Q.E.D.

2.2 b)

Note that

$$M(X_i) = \sum_{t=1}^{T} \beta_t G_t(X_i)$$

$$M(X_i) = \beta_T G_T(X_i) + \sum_{t=1}^{(T-1)} \beta_t G_t(X_i)$$

therefore

$$\beta_T G_T(X_i) = M(X_i) - \sum_{t=1}^{(T-1)} \beta_t G_t(X_i)$$

Then

$$w_i^{(T+1)} = \frac{w_i^{(T)} \exp\left(-\beta_T y_i G_T(X_i)\right)}{Z_T}$$

$$= \frac{w_i^{(T)} \exp\left(-y_i \left(M(X_i) - \sum_{t=1}^{(T-1)} \beta_t G_t(X_i)\right)\right)}{Z_T}$$

$$= \frac{w_i^{(T)} \exp\left(-y_i M(X_i) + \sum_{t=1}^{(T-1)} y_i \beta_t G_t(X_i)\right)}{Z_T}$$

$$= \frac{w_i^{(T)} \exp\left(-y_i M(X_i)\right) \exp\left(\sum_{t=1}^{(T-1)} y_i \beta_t G_t(X_i)\right)}{Z_T}$$

$$= \frac{w_i^{(T)} \exp\left(-y_i M(X_i)\right) \prod_{t=1}^{(T-1)} \exp\left(y_i \beta_t G_t(X_i)\right)}{Z_T}$$

Note that

$$w_i^{(t+1)} = \frac{w_i^{(t)} \exp\left(-y_i \beta_t G_t(X_i)\right)}{Z_t}$$

$$\exp\left(y_i \beta_t G_t(X_i)\right) = \frac{w_i^{(t)}}{w_i^{(t+1)} Z_t}$$

and therefore

$$\prod_{t=1}^{(T-1)} \exp\left(y_i \beta_t G_t(X_i)\right) = \prod_{t=1}^{(T-1)} \frac{w_i^{(t)}}{w_i^{(t+1)} Z_t}$$

$$= \frac{w_i^{(1)} w_i^{(2)} ... w_i^{(T-1)}}{w_i^{(2)} ... w_i^{(T-1)} w_i^{(T)} \prod_{t=1}^{(T-1)} Z_t}$$

$$= \frac{w_i^{(1)}}{w_i^{(T)} \prod_{t=1}^{(T-1)} Z_t}$$

Therefore, we have

$$\frac{w_i^{(T)} \exp\left(-y_i M(X_i)\right) \prod_{t=1}^{(T-1)} \exp\left(y_i \beta_t G_t(X_i)\right)}{Z_T} = \frac{w_i^{(T)} \exp\left(-y_i M(X_i)\right) \frac{w_i^{(1)}}{w_i^{(T)} \prod_{t=1}^{(T-1)} Z_t}}{Z_T}$$

$$= \frac{w_i^{(1)} \exp\left(-y_i M(X_i)\right)}{Z_T \prod_{t=1}^{(T-1)} Z_t}$$

$$= \frac{1}{n \prod_{t=1}^{T} Z_t} \exp\left(-y_i M(X_i)\right)$$

Q.E.D.

2.3 c)

Let $n = B + G$, where "B" is the number of misclassified sample points and "G" is the number of correctly classified sample points. Also, let C be the set of indices for the correctly classified points and M be the set of indices for the incorrectly classified sample points. Note that for a correctly classified sample point i

$$\infty < -y_i M(X_i) \leq 0$$

and therefore

$$0 < e^{-y_i M(X_i)} \leq 1$$

and note that

$$0 < \sum_{i \in C} e^{-y_i M(X_i)} < \sum_{i \in C} 1$$

so

$$0 < \sum_{i \in C} e^{-y_i M(X_i)} \leq G$$

Next, note that for an incorrectly classified sample point j

$$-y_j M(X_j) \geq 0$$

and therefore

$$e^{-y_j M(X_j)} \geq 1$$

Then, note that

$$\sum_{j \in M} e^{-y_j M(X_j)} \geq \sum_{j \in M} 1$$

so

$$\sum_{j \in M} e^{-y_j M(X_j)} \geq B$$

Putting these together we see

$$\sum_{i \in C} e^{-y_i M(X_i)} + \sum_{j \in M} e^{-y_j M(X_j)} \geq B$$

and therefore

$$\sum_{i=1}^{n} e^{-y_i M(X_i)} \geq B$$

Q.E.D.

2.4 d)

Note that

$$w_i^{(T+1)} = \frac{1}{n\prod_{t=1}^{T} Z_t} e^{-y_i M(X_i)}$$

so

$$e^{-y_i M(X_i)} = n w_i^{(T+1)} \prod_{t=1}^{T} Z_t$$

and therefore

$$B \leq \sum_{i}^{n} e^{-y_i M(X_i)}$$

$$= \sum_{i}^{n} n w_i^{(T+1)} \prod_{t=1}^{T} Z_t$$

and then

$$\lim_{T\to\infty} B \leq \lim_{T\to\infty} \sum_{i}^{n} n w_i^{(T+1)} \prod_{t=1}^{T} Z_t$$

$$= \sum_{i}^{n} n w_i^{(T+1)} \lim_{T\to\infty} \prod_{t=1}^{T} Z_t$$

Note that if $\text{err}_t \leq 0.49$ then $Z_t < 0.9998$, and also note that

$$\prod_{t=1}^{T} Z_t < 0.9998^T$$

and

$$\lim_{T\to\infty} \prod_{t=1}^{T} Z_t < \lim_{T\to\infty} 0.9998^T$$

$$= 0$$

Therefore

$$\lim_{T\to\infty} B \leq \sum_{i}^{n} n w_i^{(T+1)} * 0$$

$$= 0$$

And since B is the number of misclassified points $B \geq 0$. Therefore

$$\lim_{T\to\infty} B = 0$$

Q.E.D.

2.5 e)

At each iteration of training a weak learner decision tree, previously misclassified sample points are given higher weights, such that the decision tree will focus on selecting the best features for correctly classifying those sample points. If the tree depth is limited, then only the most informative features will be used in each weak learner. The combined set of features used in the metalearner will be some subset of all possible features, and if the number of weak learners times the average tree depth is much smaller than the number of total features, then the subset of features used by the metalearner will also be smaller, and will include the only most informative features for classification, i.e. you will have feature selection.

3

3.1 a)

We have

$$R = UDV^T$$

Where R is $n \times m$, U is $n \times m$, and D and V are $m \times m$. Note that for our problem $n > m$. Let U_i and V_j represent columns of U and V respectively.

$$R = \begin{bmatrix} | & | & & | \\ U_1 & U_2 & \dots & U_m \\ | & | & & | \end{bmatrix} \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_m \end{bmatrix} \begin{bmatrix} - & V_1^T & - \\ - & V_2^T & - \\ & \vdots & \\ - & V_m^T & - \end{bmatrix}$$

$$= \begin{bmatrix} | & | & & | \\ d_1U_1 & d_2U_2 & \dots & d_mU_m \\ | & | & & | \end{bmatrix} \begin{bmatrix} - & V_1^T & - \\ - & V_2^T & - \\ & \vdots & \\ - & V_m^T & - \end{bmatrix}$$

$$= d_1U_1V_1^T + d_2U_2V_2^T + ... + d_mU_mV_m^T$$

Where d_j is a scalar singular value and $U_jV_j^T$ is an outer product of the jth left and right singular vectors. Here, we see that for any entry

$$R_{ij} = \sum_{k=1}^{m} u_{ik}d_kv_{jk}$$

$$= u_{i1}d_1v_{j1} + u_{i2}d_2v_{j2} + ... + u_{im}d_mv_{jm}$$

$$= \begin{bmatrix} u_{i1}d_1 & u_{i2}d_2 & . & . & . & u_{im}d_m \end{bmatrix} \begin{bmatrix} v_{j1} \\ v_{j2} \\ . \\ . \\ . \\ v_{jm} \end{bmatrix}$$

$$= \begin{bmatrix} u_{i1} & u_{i2} & . & . & . & u_{im} \end{bmatrix} \begin{bmatrix} d_1 & & & & & \\ & d_2 & & & & \\ & & . & & & \\ & & & . & & \\ & & & & . & \\ & & & & & d_m \end{bmatrix} \begin{bmatrix} v_{j1} \\ v_{j2} \\ . \\ . \\ . \\ v_{jm} \end{bmatrix}$$

$$= U_{i,*} D V_{j,*}^T$$

$$\boxed{= (U_{i,*} D) \cdot V_{j,*}}$$

where $U_{i,*}$ and $V_{j,*}$ denote the *ith* and *jth rows* of U and V respectively.

3.2 b)

According to result from a), you would want

$$\boxed{x_i = U_{i,*}D}$$

and

$$\boxed{y_j = V_{j,*}}$$

3.3 c)

```
# Part (c): SVD to learn low-dimensional vector representations
def svd_lfm(R):

    # Fill in the missing values in R
    ##### TODO(c): Your Code Here #####
    R = np.nan_to_num(R)

    # Compute the SVD of R
    ##### TODO(c): Your Code Here #####
    U, d, Vh = spl.svd(R, full_matrices=False)

    # Construct user and movie representations
    ##### TODO(c): Your Code Here #####
    user_vecs = user_vecs = U*d
    movie_vecs = Vh.T

    return user_vecs, movie_vecs
```

3.4 d)

```
# Part (d): Compute the training MSE loss of a given vectorization
def get_train_mse(R, user_vecs, movie_vecs):

    # Compute the training MSE loss
    ##### TODO(d): Your Code Here #####

    mse_loss = 0
    for i in range(R.shape[0]):
        for j in range(R.shape[1]):

            if not np.isnan(R[i][j]):
                mse_loss += (np.dot(user_vecs[i],movie_vecs[j])-R[i][j])**2

    return mse_loss
```

3.5 e)

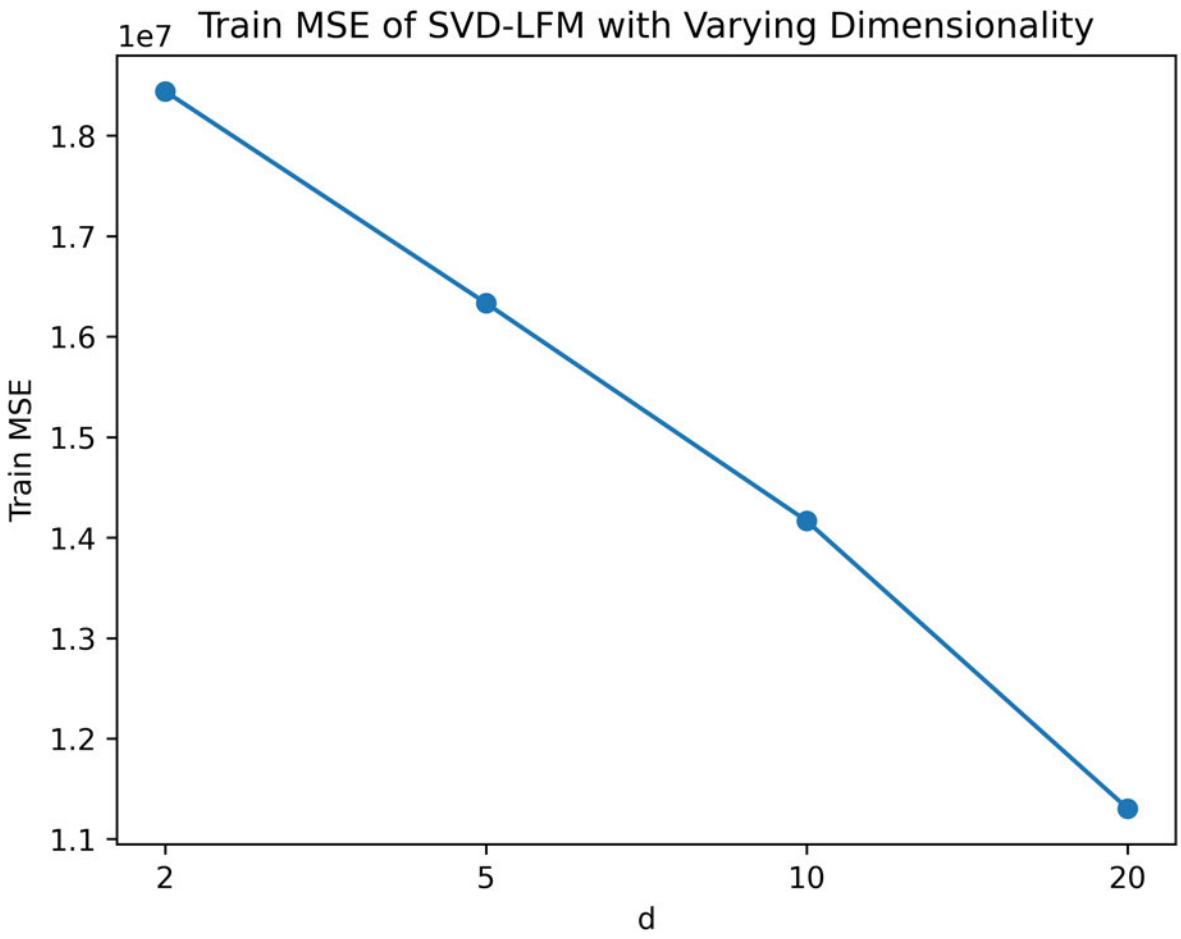


Figure 1: Mean squared error with varying dimensionality of user/movie combinations

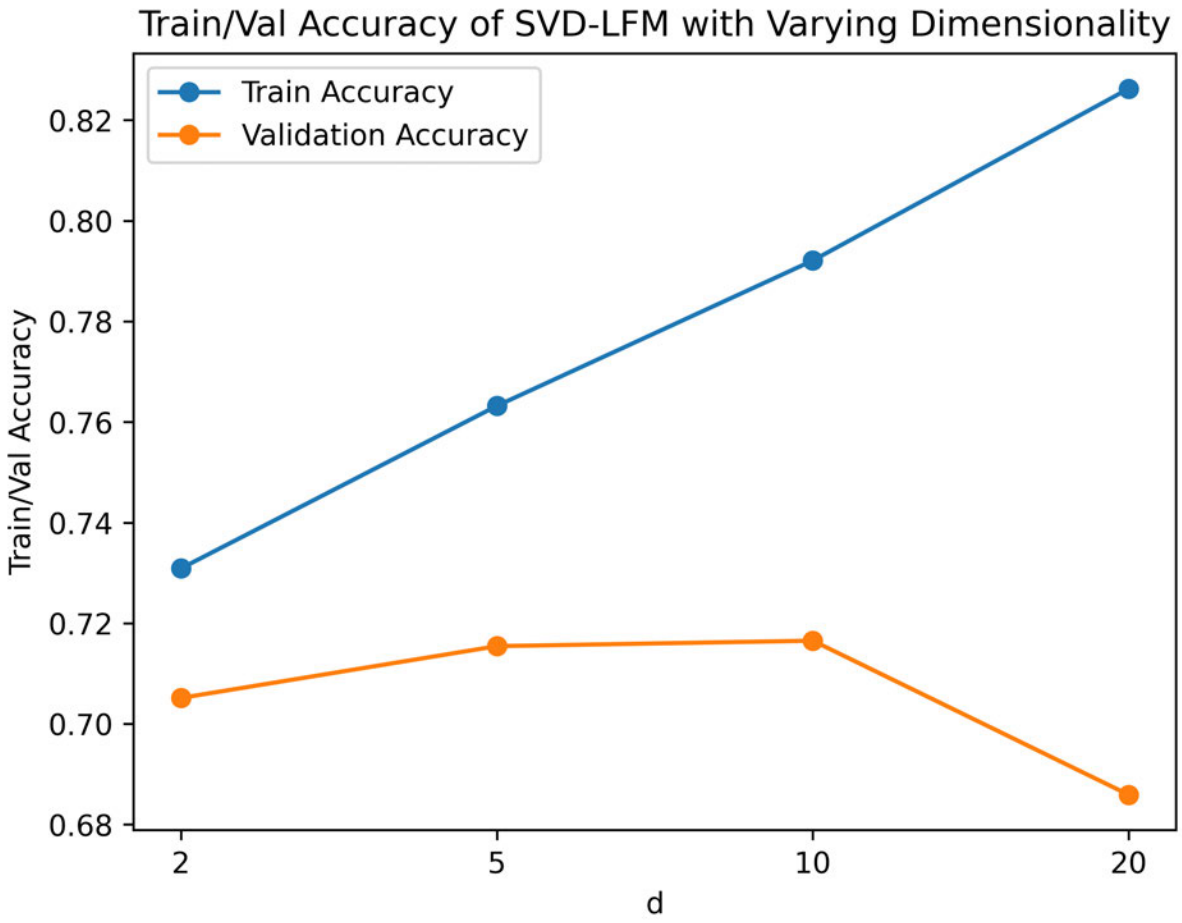


Figure 2: Training and validation accuracies with varying dimensionality of user/movie combinations

According to Figure 2, the optimal choice for the dimension of d is 10, as it gives the best accuracy for the validation set.

3.6 f)

Note: x_i and y_j are considered row vectors in this derivation, as they are row vectors in the user_vecs and movie_vecs arrays.

$$\frac{\partial}{\partial x_i} L(\{i\}, \{j\}) = 0$$

$$\frac{\partial}{\partial x_i} \sum_{(i,j) \in S} (x_i \cdot y_j - R_{ij})^2 + \frac{\partial}{\partial x_i} \sum_{i=1}^{n} \|x_i\|_2^2 + \frac{\partial}{\partial x_i} \sum_{j=1}^{m} \|y_j\|_2^2 = 0$$

$$\frac{\partial}{\partial x_i} \left[\sum_{j=1}^{m} (x_1 \cdot y_j - R_{1j})^2 + ... + \sum_{j=1}^{m} (x_i \cdot y_j - R_{ij})^2 + ... + \sum_{j=1}^{m} (x_n \cdot y_j - R_{nj})^2 \right] + \frac{\partial}{\partial x_i} \sum_{i=1}^{n} \|x_i\|_2^2 = 0$$

$$\frac{\partial}{\partial x_i} \sum_{j=1}^{m} (x_i \cdot y_j - R_{ij})^2 + \frac{\partial}{\partial x_i} \left[\|x_1\|_2^2 + ... + \|x_i\|_2^2 + ... + \|x_n\|_2^2 \right] = 0$$

$$\sum_{j=1}^{m} 2 \left(y_j x_i^T - R_{ij} \right) y_j^T + \frac{\partial}{\partial x_i} \|x_i\|_2^2 = 0$$

$$2 \sum_{j=1}^{m} \left(y_j x_i^T - R_{ij} \right) y_j^T + 2 x_i^T = 0$$

$$\sum_{j=1}^{m} (y_j x_i^T) y_j^T - \sum_{j=1}^{m} R_{ij} y_j^T + x_i^T = 0$$

$$\sum_{j=1}^{m} y_j^T (y_j x_i^T) + x_i^T = \sum_{j=1}^{m} y_j^T R_{ij}$$

$$y_1^T (y_1 x_i^T) + y_2^T (y_2 x_i^T) + ... + y_m^T (y_m x_i^T) + x_i^T = y_1^T R_{i1} + y_2^T R_{i2} + ... + y_m^T R_{im}$$

$$\begin{bmatrix} | & | & & | \\ y_1^T & y_2^T & \cdots & y_m^T \\ | & | & & | \end{bmatrix} \begin{bmatrix} y_1 x_i^T \\ y_2 x_i^T \\ \cdot \\ \cdot \\ \cdot \\ y_m x_i^T \end{bmatrix} + x_i^T = \begin{bmatrix} | & | & & | \\ y_1^T & y_2^T & \cdots & y_m^T \\ | & | & & | \end{bmatrix} \begin{bmatrix} R_{i1} \\ R_{i2} \\ \cdot \\ \cdot \\ \cdot \\ R_{im} \end{bmatrix}$$

$$Y^T Y x_i^T + x_i^T = Y^T R_i^T$$

Where R_i is the ith row vector of R and where Y is the orthogonal matrix with the each movie as a row and the columns are the latent movie features.

$$(Y^T Y + I) x_i^T = Y^T R_i^T$$

$$\boxed{x_i^T = (Y^T Y + I)^{-1} Y^T R_i^T}$$

and $(Y^T Y + I)$ is guaranteed to be invertible since Y is orthogonal. The update for y_i is derived in a similar fashion and is

$$\boxed{y_i^T = (X^T X + I)^{-1} X^T R_j}$$

where R_j is the jth column of R.

```
# Part (f): Learn better user/movie vector representations by minimizing loss
# begin solution
best_d = 10 # TODO(f): Use best from part (e)
# end solution
np.random.seed(20)
user_vecs = np.random.random((R.shape[0], best_d))
movie_vecs = np.random.random((R.shape[1], best_d))
user_rated_idxs, movie_rated_idxs = get_rated_idxs(np.copy(R))

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

    # Update user_vecs to the loss-minimizing value
    ##### TODO(f): Your Code Here #####

    #iterate over each user
    for i in range(user_vecs.shape[0]):

        #save the movie_vecs for only the movies the ith person
        #rated in a matrix
        Y = movie_vecs[user_rated_idxs[i]]

        #Get just the ratings for the movies the ith person
        Ri = R[i][user_rated_idxs[i]]

        #compute Y^TR_i
        YTRi = np.matmul(Y.T,Ri)

        #compute the inverse matrix (Y^TY+I)^(-1)
        inv = np.linalg.inv(np.matmul(Y.T,Y)+np.identity(Y.shape[1]))

        #compute the update for x_i
        user_vecs[i] = np.matmul(inv,YTRi)

    return user_vecs

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

    # Update movie_vecs to the loss-minimizing value
    ##### TODO(f): Your Code Here #####

    #iterate over each movie
    for j in range(movie_vecs.shape[0]):
```

```
        #save the user_vecs for only the users that rated the jth movie
        #all in a matrix
        X = user_vecs[movie_rated_idxs[j]]

        #Get just the ratings for the users who watched the jth movie
        Rj = R[movie_rated_idxs[j],j]

        #compute the matrix X^TR_j
        XTRj = np.matmul(X.T,Rj)

        #compute the inverse matrix (X^TX+I)^(-1)
        inv = np.linalg.inv(np.matmul(X.T,X)+np.identity(X.shape[1]))

        #compute the update for y_j
        movie_vecs[j] = np.matmul(inv,XTRj)

    return movie_vecs
```

```
PS C:\Users\Colin\Desktop\CS289A23\hw7> python movie_recommender.py
Start optim, train MSE: 27574866.30, train accuracy: 0.5950, val
   accuracy: 0.5799
Iteration 1, train MSE: 13421216.24, train accuracy: 0.7611, val
   accuracy: 0.6431
Iteration 2, train MSE: 11474959.41, train accuracy: 0.7876, val
   accuracy: 0.6789
Iteration 3, train MSE: 10493324.86, train accuracy: 0.8007, val
   accuracy: 0.6989
Iteration 4, train MSE: 10040997.98, train accuracy: 0.8069, val
   accuracy: 0.7084
Iteration 5, train MSE: 9792296.83, train accuracy: 0.8098, val
   accuracy: 0.7100
Iteration 6, train MSE: 9649312.88, train accuracy: 0.8117, val
   accuracy: 0.7100
Iteration 7, train MSE: 9561491.69, train accuracy: 0.8130, val
   accuracy: 0.7060
Iteration 8, train MSE: 9503837.41, train accuracy: 0.8138, val
   accuracy: 0.7117
Iteration 9, train MSE: 9463660.97, train accuracy: 0.8144, val
   accuracy: 0.7111
Iteration 10, train MSE: 9434168.95, train accuracy: 0.8147, val
   accuracy: 0.7087
Iteration 11, train MSE: 9411512.64, train accuracy: 0.8150, val
   accuracy: 0.7119
Iteration 12, train MSE: 9393397.49, train accuracy: 0.8152, val
   accuracy: 0.7103
Iteration 13, train MSE: 9378404.19, train accuracy: 0.8155, val
   accuracy: 0.7125
Iteration 14, train MSE: 9365635.88, train accuracy: 0.8156, val
   accuracy: 0.7122
Iteration 15, train MSE: 9354518.75, train accuracy: 0.8157, val
   accuracy: 0.7125
Iteration 16, train MSE: 9344681.51, train accuracy: 0.8158, val
   accuracy: 0.7136
Iteration 17, train MSE: 9335879.18, train accuracy: 0.8159, val
   accuracy: 0.7144
Iteration 18, train MSE: 9327944.20, train accuracy: 0.8160, val
   accuracy: 0.7146
Iteration 19, train MSE: 9320755.69, train accuracy: 0.8161, val
   accuracy: 0.7149
Iteration 20, train MSE: 9314221.76, train accuracy: 0.8163, val
   accuracy: 0.7160
```

Listing 1: Terminal output after running movie_recommender.py

Final output:

training MSE: 9314221.76
training accuracy: 0.8163
validation accuracy: 0.7160

Compared to the results from part e) for d=10, there was a marginal improvement to the training accuracy, and possibly a very small improvement to to the validation accuracy.

4

4.1 a)

```
# Part A: PCA (modify plot_pca method in world_values_utils) #
plot_pca(values_train, hdi_train)
```

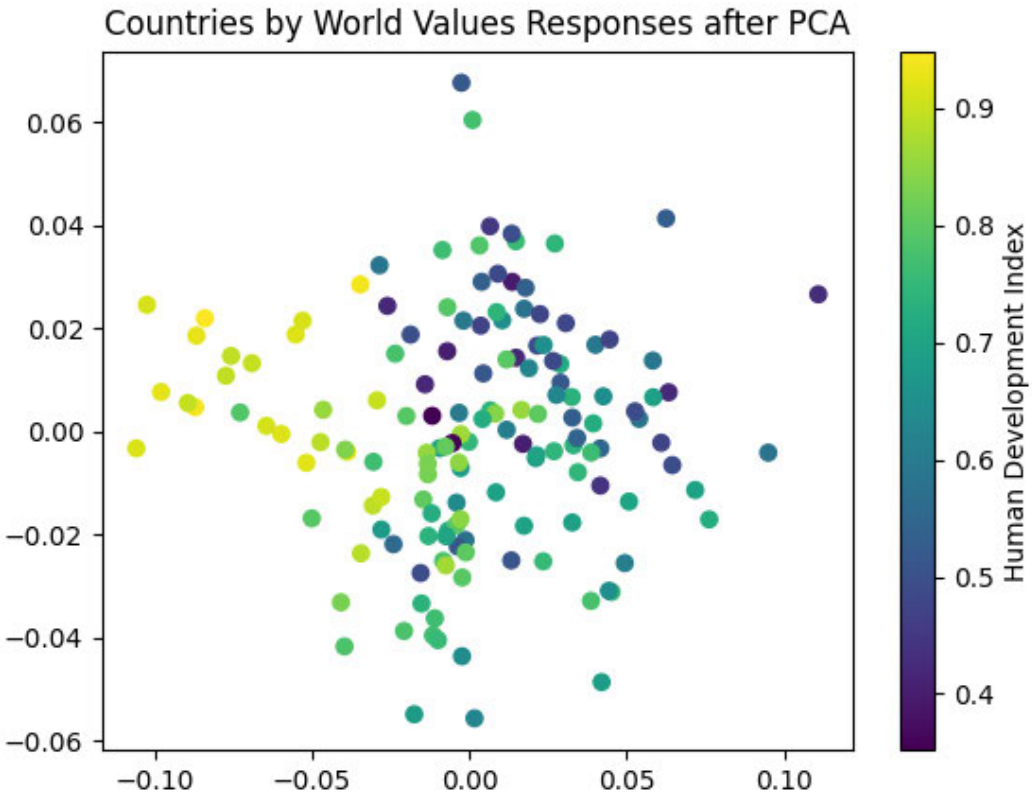


Figure 3: First two principle components of World Values Survey data

4.2 b)

Regression with k-NN could be done several ways. Once the k-nearest neighbors for a test point are identified using some distance metric, a prediction for the test point's value could be made by taking some aggregate of the values of its k-nearest neighbors, such as a weighted mean, or by assigning the prediction to be the median or mode of the nearest neighbors. Additionally, instead of taking some aggregate of the neighbors' values, linear regression could be done on the nearest neighbors' values and the input features, then the best linear predictor could be used to predict a value for the test point. Even beyond simple linear regression, a random forest or neural network could be used for non-linear regression.

4.3 c)

```
    # Part C: Find the 7 nearest neighbors of the U.S.
    nbrs = NearestNeighbors(n_neighbors=8).fit(values_train)
#     us_features = values_train.iloc[45].to_numpy().reshape(1, -1)
    us_features = values_train.iloc[[45]]

    # Use nbrs to get the k nearest neighbors of us_features
    # & retrieve the corresponding countries
    ##### TODO(c): Your Code Here #####

    distances, indices = nbrs.kneighbors(us_features, 8)
    seven_nearest = countries[indices[0]].tolist()[1:]

    # Calculate the length of the longest name in seven_nearest
    max_len = max(len(name) for name in seven_nearest)

    print('Seven nearest neighbors to the US ranked:')
    print("COUNTRY ", "       DISTANCE ")

    for a, b in zip(seven_nearest, distances[0][1:]):
        print("{:<{}}   {:.4f}".format(a, max_len, b))
```

```
Seven nearest neighbors to the US ranked:
COUNTRY          DISTANCE
Ireland           0.0177
United Kingdom    0.0216
Belgium           0.0301
Finland           0.0302
Malta             0.0335
Austria           0.0358
France            0.0390
```

Listing 2: Values: Seven nearest countries to the US

4.4 d)

```
# Use GridSearchCV to create and fit a grid of search results
##### TODO(d): Your Code Here #####
grid = GridSearchCV(pipeline, parameters, scoring='neg_root_mean_squared_error').


print("RMSE:", -grid.best_score_)
print(grid.best_estimator_, "\n")

# Plot RMSE vs k for k Nearest Neighbors Regression
plt.plot(grid.cv_results_['param_knn__n_neighbors'],
         (-grid.cv_results_['mean_test_score']))
plt.xlabel('k')
plt.ylabel('RMSE')
plt.title('RMSE versus k in kNN')
plt.show()
```

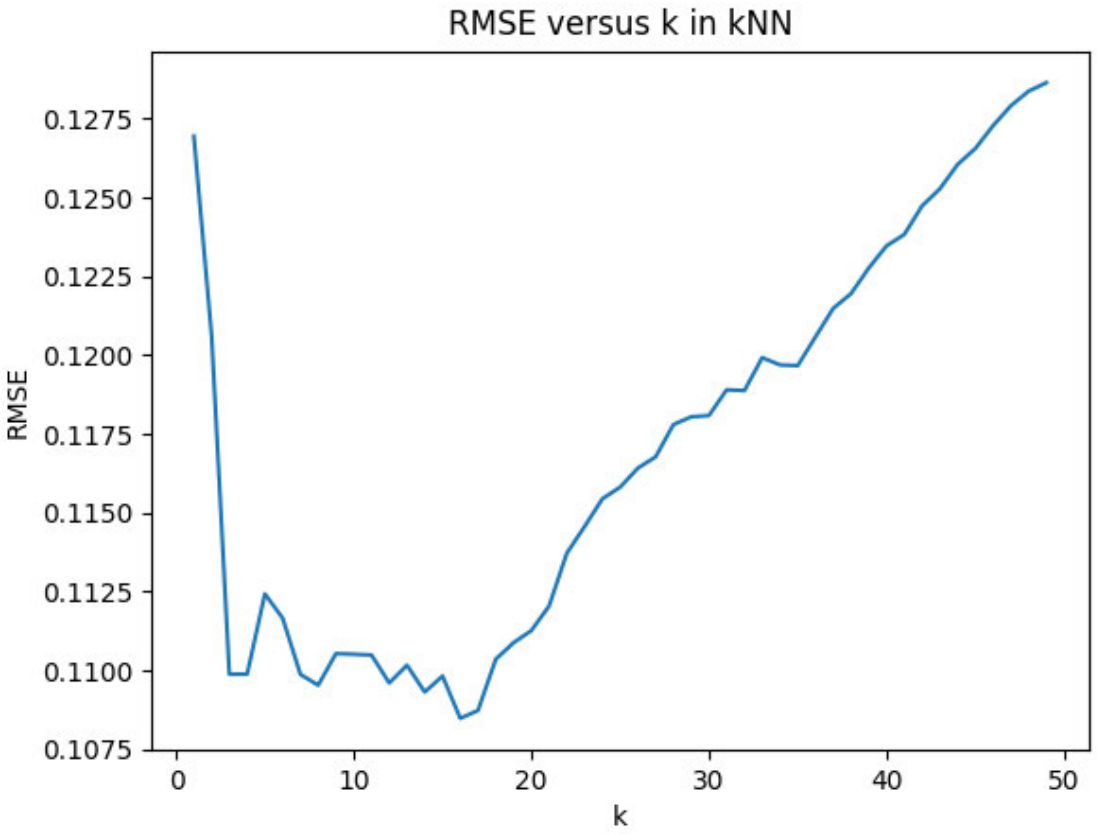


Figure 4: The root-mean-squared error of the k-NN regression vs. k

```
RMSE: 0.10848529119381482
Pipeline(steps=[('knn', KNeighborsRegressor(n_neighbors=16))])
```

Listing 3: Terminal output after running world_values_starter.py showing the RMSE of the optimum value for k

4.5 e)

RMSE starts high with k=1, likely due to very low bias, leading to high variance. As the number of nearest neighbors increases to 16, the bias increases as well, and in exchange the variance drops, indicating the model is fitting less to the noise of the training data. However, as k increases the bias keeps increasing to the point that the model is just a poor fit to both the training data and any test data, so the variance begins to increase, almost monotonically.

4.6 f)

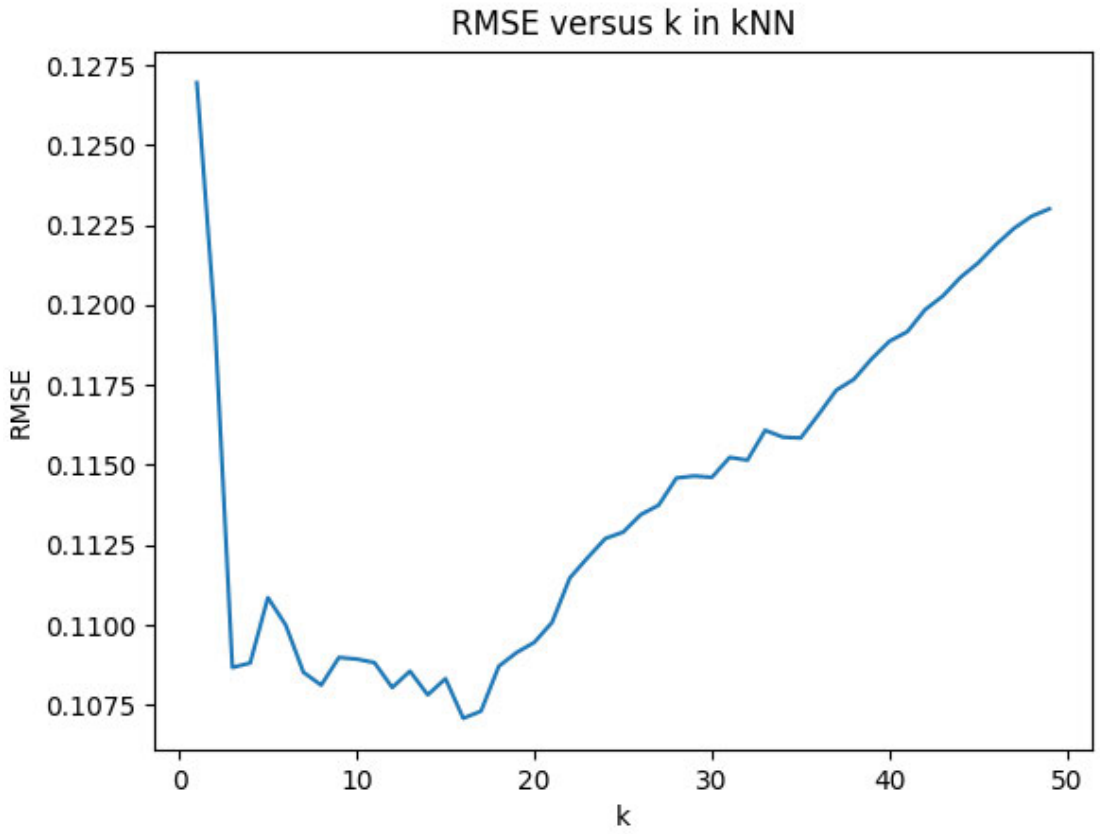


Figure 5: The root-mean-squared error of the k-NN regression vs. k with weight inverse to distance

```
RMSE: 0.10708230222935769
Pipeline(steps=[('knn',
                 KNeighborsRegressor(n_neighbors=16, weights='distance
    '))])
```

Listing 4: Terminal output after running world_values_starter.py showing the RMSE of the optimum value for k with weight inverse to distance

With weights of the neighbors being inversely proportional to distance from the test point the shape of the RMSE vs. k plot is similar, and the optimum k is the same as when the neighbors were weighted equally. However, the overall RMSE is slightly lower, and as k balloons, the RMSE increases more gradually than it did for the equal weighting. This makes sense because, while the number of neighbors increases, more of the included points will be at a further distance, and their weight falls off proportionally.

4.7 g)

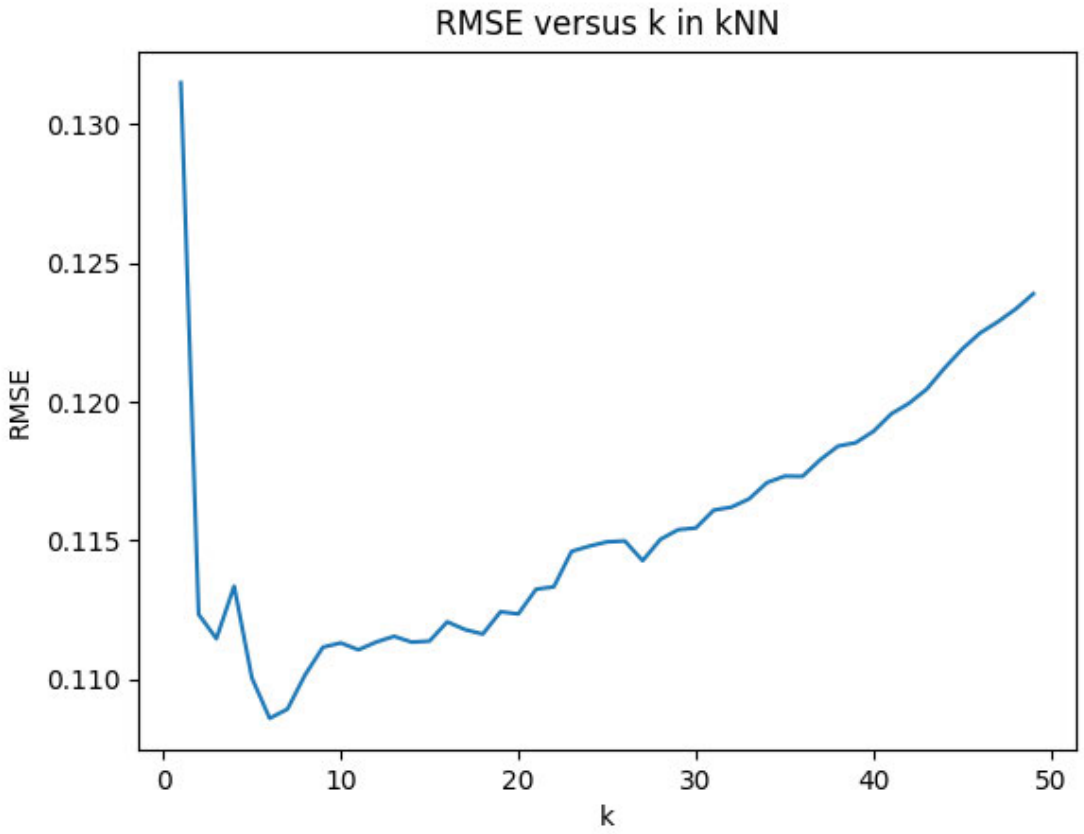


Figure 6: The root-mean-squared error of the k-NN regression vs. k with weight inverse to distance and with standardization

```
RMSE: 0.108594469882015
Pipeline(steps=[('scale', StandardScaler()),
                ('knn',
                 KNeighborsRegressor(n_neighbors=6, weights='distance'
   ))])
```

Listing 5: Terminal output after running world_values_starter.py showing the RMSE of the optimum value for k with weight inverse to distance, and with the features standardized

Standardizing the data causes the optimal k to shrink to six, but the best RMSE does not decrease. Instead, it very slightly increased.

5 Code Appendix

movie_recommender.py

```
import os
import scipy.io
import numpy as np
import scipy.linalg as spl
import matplotlib.pyplot as plt

# Load training data from MAT file
R = scipy.io.loadmat('movie_data/movie_train.mat')['train']

# Load validation data from CSV
val_data = np.loadtxt('movie_data/movie_validate.txt', dtype=int, delimiter=',')

# Helper method to get training accuracy
def get_train_acc(R, user_vecs, movie_vecs):
    num_correct, total = 0, 0
    for i in range(R.shape[0]):
        for j in range(R.shape[1]):
            if not np.isnan(R[i, j]):
                total += 1
                if np.dot(user_vecs[i], movie_vecs[j])*R[i, j] > 0:
                    num_correct += 1
    return num_correct/total

# Helper method to get validation accuracy
def get_val_acc(val_data, user_vecs, movie_vecs):
    num_correct = 0
    for val_pt in val_data:
        user_vec = user_vecs[val_pt[0]-1]
        movie_vec = movie_vecs[val_pt[1]-1]
        est_rating = np.dot(user_vec, movie_vec)
        if est_rating*val_pt[2] > 0:
            num_correct += 1
    return num_correct/val_data.shape[0]

# Helper method to get indices of all rated movies for each user,
# and indices of all users who have rated that title for each movie
def get_rated_idxs(R):
    user_rated_idxs, movie_rated_idxs = [], []
    for i in range(R.shape[0]):
        user_rated_idxs.append(np.argwhere(~np.isnan(R[i, :])).reshape(-1))
    for j in range(R.shape[1]):
```

```
        movie_rated_idxs.append(np.argwhere(~np.isnan(R[:, j])).reshape(-1))
    return np.array(user_rated_idxs, dtype=object), np.array(movie_rated_idxs, dtype=

# Part (c): SVD to learn low-dimensional vector representations
def svd_lfm(R):

    # Fill in the missing values in R
    ##### TODO(c): Your Code Here #####
    R = np.nan_to_num(R)

    # Compute the SVD of R
    ##### TODO(c): Your Code Here #####
    U, d, Vh = spl.svd(R, full_matrices=False)

    # Construct user and movie representations
    ##### TODO(c): Your Code Here #####
    user_vecs = user_vecs = U*d
    movie_vecs = Vh.T

    return user_vecs, movie_vecs

# Part (d): Compute the training MSE loss of a given vectorization
def get_train_mse(R, user_vecs, movie_vecs):

    # Compute the training MSE loss
    ##### TODO(d): Your Code Here #####

    mse_loss = 0
    for i in range(R.shape[0]):
        for j in range(R.shape[1]):

            if not np.isnan(R[i][j]):
                mse_loss += (np.dot(user_vecs[i],movie_vecs[j])-R[i][j])**2

    return mse_loss


# Part (e): Compute training MSE and val acc of SVD LFM for various d
d_values = [2, 5, 10, 20]
train_mses, train_accs, val_accs = [], [], []
user_vecs, movie_vecs = svd_lfm(np.copy(R))
for d in d_values:
    train_mses.append(get_train_mse(np.copy(R), user_vecs[:, :d], movie_vecs[:, :d]))
    train_accs.append(get_train_acc(np.copy(R), user_vecs[:, :d], movie_vecs[:, :d]))
    val_accs.append(get_val_acc(val_data, user_vecs[:, :d], movie_vecs[:, :d]))
```

```
plt.clf()
plt.plot([str(d) for d in d_values], train_mses, 'o-')
plt.title('Train MSE of SVD-LFM with Varying Dimensionality')
plt.xlabel('d')
plt.ylabel('Train MSE')
plt.savefig(fname='train_mses.png', dpi=600, bbox_inches='tight')
plt.clf()
plt.plot([str(d) for d in d_values], train_accs, 'o-')
plt.plot([str(d) for d in d_values], val_accs, 'o-')
plt.title('Train/Val Accuracy of SVD-LFM with Varying Dimensionality')
plt.xlabel('d')
plt.ylabel('Train/Val Accuracy')
plt.legend(['Train Accuracy', 'Validation Accuracy'])
plt.savefig(fname='trval_accs.png', dpi=600, bbox_inches='tight')


# Part (f): Learn better user/movie vector representations by minimizing loss
# begin solution
best_d = 10 # TODO(f): Use best from part (e)
# end solution
np.random.seed(20)
user_vecs = np.random.random((R.shape[0], best_d))
movie_vecs = np.random.random((R.shape[1], best_d))
user_rated_idxs, movie_rated_idxs = get_rated_idxs(np.copy(R))

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

    # Update user_vecs to the loss-minimizing value
    ##### TODO(f): Your Code Here #####

    #iterate over each user
    for i in range(user_vecs.shape[0]):

        #save the movie_vecs for only the movies the ith person
        #rated in a matrix
        Y = movie_vecs[user_rated_idxs[i]]

        #Get just the ratings for the movies the ith person
        Ri = R[i][user_rated_idxs[i]]

        #compute Y^TR_i
        YTRi = np.matmul(Y.T,Ri)

        #compute the inverse matrix (Y^TY+I)^(-1)
```

```
        inv = np.linalg.inv(np.matmul(Y.T,Y)+np.identity(Y.shape[1]))

        #compute the update for x_i
        user_vecs[i] = np.matmul(inv,YTRi)

    return user_vecs

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

    # Update movie_vecs to the loss-minimizing value
    ##### TODO(f): Your Code Here #####

    #iterate over each movie
    for j in range(movie_vecs.shape[0]):

        #save the user_vecs for only the users that rated the jth movie
        #all in a matrix
        X = user_vecs[movie_rated_idxs[j]]

        #Get just the ratings for the users who watched the jth movie
        Rj = R[movie_rated_idxs[j],j]

        #compute the matrix X^TR_j
        XTRj = np.matmul(X.T,Rj)

        #compute the inverse matrix (X^TX+I)^(-1)
        inv = np.linalg.inv(np.matmul(X.T,X)+np.identity(X.shape[1]))

        #compute the update for y_j
        movie_vecs[j] = np.matmul(inv,XTRj)

    return movie_vecs

# Part (f): Perform loss optimization using alternating updates
train_mse = get_train_mse(np.copy(R), user_vecs, movie_vecs)
train_acc = get_train_acc(np.copy(R), user_vecs, movie_vecs)
val_acc = get_val_acc(val_data, user_vecs, movie_vecs)
print(f'Start optim, train MSE: {train_mse:.2f}, train accuracy: {train_acc:.4f}, val
for opt_iter in range(20):
    user_vecs = update_user_vecs(user_vecs, movie_vecs, np.copy(R), user_rated_idxs)
    movie_vecs = update_movie_vecs(user_vecs, movie_vecs, np.copy(R), movie_rated_idx
    train_mse = get_train_mse(np.copy(R), user_vecs, movie_vecs)
    train_acc = get_train_acc(np.copy(R), user_vecs, movie_vecs)
    val_acc = get_val_acc(val_data, user_vecs, movie_vecs)
```

```
    print(f'Iteration {opt_iter+1}, train MSE: {train_mse:.2f}, train accuracy: {trai
```

world_values_parameters.py

```python
import numpy as np

regression_knn_parameters = {
    'knn__n_neighbors': np.arange(1, 50),

    # Apply uniform weighting vs k for k Nearest Neighbors Regression
    ##### TODO(f): Change the weighting #####
    'knn__weights': ['distance']
}
```

world_values_pipelines.py

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor

k_nearest_neighbors_regression_pipeline = Pipeline(
        [
            # Apply scaling to k Nearest Neighbors Regression
            ##### TODO(g): Add a 'scale' parameter that applies StandardScaler() ##
            ('scale', StandardScaler()),

            ('knn', KNeighborsRegressor())
        ]
    )
```

world_values_starter.py

```
"""
The world_values data set is available online at http://54.227.246.164/dataset/. In
    residents of almost all countries were asked to rank their top 6 'priorities'.
    they were asked "Which of these are most important for you and your family?"

This code and world-values.tex guides the student through the process of training s
    that predict the HDI (Human Development Index) rating of a country from the res
    citizens to the world values data.
"""

from math import sqrt
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import NearestNeighbors

from world_values_utils import import_world_values_data
from world_values_utils import plot_pca

from world_values_pipelines import k_nearest_neighbors_regression_pipeline
from world_values_parameters import regression_knn_parameters


def main():
    print("Predicting HDI from World Values Survey\n")

    # Import Data #
    print("Importing Training Data")
    values_train, hdi_train, countries = import_world_values_data()

    # Center the HDI Values #
    hdi_scaler = StandardScaler(with_std=False)
    hdi_shifted_train = hdi_scaler.fit_transform(hdi_train)

    # Data Information #
    print('Training Data Count:', values_train.shape[0])

    # Part A: PCA (modify plot_pca method in world_values_utils) #
    plot_pca(values_train, hdi_train)

    # Part C: Find the 7 nearest neighbors of the U.S.
    nbrs = NearestNeighbors(n_neighbors=8).fit(values_train)
```

```
#     us_features = values_train.iloc[45].to_numpy().reshape(1, -1)
    us_features = values_train.iloc[[45]]

    # Use nbrs to get the k nearest neighbors of us_features
    # & retrieve the corresponding countries
    ##### TODO(c): Your Code Here #####

    distances, indices = nbrs.kneighbors(us_features, 8)
    seven_nearest = countries[indices[0]].tolist()[1:]

    # Calculate the length of the longest name in seven_nearest
    max_len = max(len(name) for name in seven_nearest)

    print('Seven nearest neighbors to the US ranked:')
    print("COUNTRY ", "       DISTANCE ")

    for a, b in zip(seven_nearest, distances[0][1:]):
        print("{:<{}}   {:.4f}".format(a, max_len, b))

    # Part D: complete _rmse_grid_search to find the best value of k for Regression
    # Parts F and H: rerun this after modifications to find the best value of k for
    _rmse_grid_search(values_train, hdi_shifted_train,
                      k_nearest_neighbors_regression_pipeline,
                      regression_knn_parameters, 'knn')


def _rmse_grid_search(training_features, training_labels, pipeline, parameters, techn
    """
    Input:
        training_features: world_values responses on the training set
        training_labels: HDI (human development index) on the training set
        pipeline: regression model specific pipeline
        parameters: regression model specific parameters
        technique: regression model's name

    Output:
        Prints best RMSE and best estimator
        Prints feature weights for Ridge and Lasso Regression
        Plots RMSE vs k for k Nearest Neighbors Regression
    """
    # Use GridSearchCV to create and fit a grid of search results
    ##### TODO(d): Your Code Here #####
    grid = GridSearchCV(pipeline, parameters, scoring='neg_root_mean_squared_error').
```

```
    print("RMSE:", -grid.best_score_)
    print(grid.best_estimator_, "\n")

    # Plot RMSE vs k for k Nearest Neighbors Regression
    plt.plot(grid.cv_results_['param_knn__n_neighbors'],
             (-grid.cv_results_['mean_test_score']))
    plt.xlabel('k')
    plt.ylabel('RMSE')
    plt.title('RMSE versus k in kNN')
    plt.show()


if __name__ == '__main__':
    main()
```

world_values_utils.py

```
import pandas as pd
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt


def import_world_values_data():
    """
    Reads the world values data into data frames.

    Returns:
        values_train: world_values responses on the training set
        hdi_train: HDI (human development index) on the training set
        countries: countries corresponding to indices of values_train
    """
    values_train = pd.read_csv('world-values-train2.csv')
    countries = values_train['Country']
    values_train = values_train.drop(['Country'], axis=1)
    hdi_train = pd.read_csv('world-values-hdi-train2.csv')
    hdi_train = hdi_train.drop(['Country'], axis=1)
    return values_train, hdi_train, countries


def plot_pca(training_features,
             training_labels):
    """
    Input:
        training_features: world_values responses on the training set
        training_labels: HDI (human development index) on the training set
        training_classes: HDI class, determined by hdi_classification(), on the tra

    Output:
        Displays plot of first two PCA dimensions vs HDI
        Displays plot of first two PCA dimensions vs HDI, colored by class
    """
    # Run PCA on training_features
    ##### TODO(a): Your Code Here #####

    pca = PCA(n_components=2)
    transformed_features = pca.fit_transform(training_features)


    # Plot countries by first two PCA dimensions
    plt.scatter(transformed_features[:, 0],     # Select first column
```

```
                transformed_features[:, 1],     # Select second column
                c=training_labels['2015'])
    plt.colorbar(label='Human Development Index')
    plt.title('Countries by World Values Responses after PCA')
    plt.show()


def hdi_classification(hdi):
    """
    Input:
        hdi: HDI (human development index) value

    Output:
        high HDI vs low HDI class identification
    """
    if 1.0 > hdi >= 0.7:
        return 1.0
    elif 0.7 > hdi >= 0.30:
        return 0.0
    else:
        raise ValueError('Invalid HDI')
```