Hiva Mohammadzadeh

CS 189 Introduction to Machine Learning
Spring 2023 Jonathan Shewchuk

HW7

Due: Friday, May 5 at 11:59 pm

Deliverables:

1. Submit a PDF of your homework, **with an appendix listing all your code**, to the Gradescope assignment entitled "Homework 7 Write-Up". In addition, please include, as your solutions to each coding problem, the specific subset of code relevant to that part of the problem. You may typeset your homework in LaTeX or Word (submit PDF format, **not** .doc/.docx format) or submit neatly handwritten and scanned solutions. **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.

   - In your write-up, please state with whom you worked on the homework.
   - In your write-up, please copy the following statement and sign your signature next to it. (Mac Preview and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make it *extra* clear so that no one inadvertently cheats.
     "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."

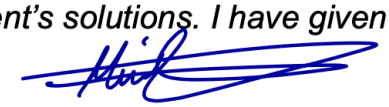
2. Submit all the code needed to reproduce your results to the Gradescope assignment entitled "Homework 7 Code". Yes, you must submit your code twice: once in your PDF write-up following the directions as described above so the readers can easily read it, and once in compilable/interpretable form so the readers can easily run it. Do **NOT** include any data files we provided. Please include a short file named README listing your name, student ID, and instructions on how to reproduce your results. Please take care that your code doesn't take up inordinate amounts of time or memory to run. If your code cannot be executed, your solution cannot be verified.

Hiva Mohammadzadeh

3036919598

Question 1: Honor Code

“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.”



Question 2: The raining Error of AdaBoost

1 Honor Code

Declare and sign the following statement:

"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."

Signature : _____

While discussions are encouraged, *everything* in your solution must be your (and only your) creation. Furthermore, all external material (i.e., *anything* outside lectures and assigned readings, including figures and pictures) should be cited properly. We wish to remind you that consequences of academic misconduct are *particularly severe*!

2 The Training Error of AdaBoost

Recall that in AdaBoost, our input is an $n \times d$ design matrix X with n labels $y_i = \pm 1$, and at the end of iteration T the importance of each sample is reweighted as

$$w_i^{(T+1)} = w_i^{(T)} \exp(-\beta_T \, y_i \, G_T(X_i)), \quad \text{where} \quad \beta_T = \frac{1}{2} \ln\left(\frac{1 - \text{err}_T}{\text{err}_T}\right) \quad \text{and} \quad \text{err}_T = \frac{\sum_{y_i \neq G_T(X_i)} w_i^{(T)}}{\sum_{i=1}^n w_i^{(T)}}.$$

Note that err_T is the weighted error rate of the classifier G_T. Recall that $G_T(z)$ is ± 1 for all points z, but the metalearner has a non-binary decision function $M(z) = \sum_{t=1}^T \beta_t G_t(z)$. To classify a test point z, we calculate $M(z)$ and return its sign.

In this problem we will prove that if every learner G_t achieves 51% accuracy (that is, only slightly above random), AdaBoost will converge to zero training error. (If you get stuck on one part, move on; all five parts below can be done without solving the other parts, and parts (c) and (e) are the easiest.)

(a) We want to change the update rule to "normalize" the weights so that each iteration's weights sum to 1; that is, $\sum_{i=1}^n w_i^{(T+1)} = 1$. That way, we can treat the weights as a discrete probability distribution over the sample points. Hence we rewrite the update rule in the form

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

for some scalar Z_T. Show that if $\sum_{i=1}^n w_i^{(T)} = 1$ and $\sum_{i=1}^n w_i^{(T+1)} = 1$, then

$$Z_T = 2\sqrt{\text{err}_T(1 - \text{err}_T)}. \tag{2}$$

Hint: sum over both sides of (1), then split the right summation into misclassified points and correctly classified points.

(b) The initial weights are $w_1^{(1)} = w_2^{(1)} = \cdots = w_n^{(1)} = \frac{1}{n}$. Show that

$$w_i^{(T+1)} = \frac{1}{n \prod_{t=1}^T Z_t} e^{-y_i M(X_i)}. \tag{3}$$

(c) Let B (for "bad") be the number of sample points out of n that the metalearner classifies incorrectly. Show that

$$\sum_{i=1}^n e^{-y_i M(X_i)} \geq B. \tag{4}$$

Hint: split the summation into misclassified points and correctly classified points.

(2) $w_i^{(T+1)} = w_i^{(T)} \exp(-\beta_T y_i G_T(x_i))$, where $\beta_T = \frac{1}{2}\ln\left(\frac{1-err_T}{err_T}\right)$ and $err_T = \frac{\sum_{y_i \neq G_T(x_i)} w_i^{(T)}}{\sum_{i=1}^n w_i^{(T)}}$

a) $w_i^{(T+1)} = \frac{w_i^{(T)} \exp(-\beta_T y_i G_T(x_i))}{z_T}$. Show that if $\sum_{i=1}^n w_i^{(T)} = 1$ and $\sum_{i=1}^n w_i^{(T+1)} = 1$, then $z_T = 2\sqrt{err_T(1-err_T)}$

Using the hint to sum over both sides of the equation, we get and the fact that $\sum_{i=1}^n w_i^{(T+1)} = 1$,

$$z_T = \sum_{i=1}^n w_i^{(T)} \exp(-\beta_T y_i G_T(x_i)) = \sum_{y_i = G_T(x_i)} w_i^{(T)} \underbrace{e^{-\beta_T}}_{\text{scalar}} + \sum_{y_i \neq G_T(x_i)} w_i^{(T)} \underbrace{e^{\beta_T}}_{\text{scalar}}$$

$$= e^{-\beta_T} \sum_{y_i = G_T(x_i)} w_i^{(T)} + e^{\beta_T} \sum_{y_i \neq G_T(x_i)} w_i^{(T)} = e^{-\beta_T}(1-err_T) + e^{\beta_T} err_T$$

$$= \sqrt{\frac{err_T}{1-err_T}}(1-err_T) + \sqrt{\frac{1-err_T}{err_T}}\, err_T = \boxed{2\sqrt{err_T(1-err_T)}} \checkmark$$

b) Show that $w_i^{(T+1)} = \frac{1}{n\prod_{t=1}^T z_t} e^{-y_i M(x_i)}$

$$w_i^{(T+1)} = w_i^{(1)} \prod_{t=1}^T \frac{e^{-\beta_t y_i G_t(x_i)}}{z_t} = \frac{1}{n\prod_{t=1}^T z_t} e^{-y_i \sum_{t=1}^T \beta_t G_t(x_i)}$$

$$= \boxed{\frac{1}{n\prod_{t=1}^T z_t} e^{-y_i M(x_i)}} \checkmark$$

c) Show that $\sum_{i=1}^n e^{-y_i M(x_i)} \geq B$.

Using the hint and spliting the sumation, we get:

$$\sum_{i=1}^n e^{-y_i M(x_i)} = \sum_{y_i M(x_i) \leq 0} e^{-y_i M(x_i)} + \sum_{y_i M(x_i) > 0} e^{-y_i M(x_i)}$$

$$\geq \sum_{y_i M(x_i) \leq 0} 1 + \sum_{y_i M(x_i) > 0} 0 = \boxed{B}$$

② cont.

d) Show that if $err_t \leq 0.49$ for every learner G_t, then $B \to 0$ as $T \to \infty$.

Using the hint that $z_t < 0.9998$, we get:

$err_t \leq 0.49 \to z_t \leq 2\sqrt{0.49(0.51)} = 0.9998 < 1$. So,

$\lim_{T\to\infty} \prod_{t=1}^{T} z_t = 0$ $\quad \sum_{i=1}^{n} w_i^{(k)} = 1$ for all k. $e^{-y_i M(x_i)} \to 0$ because $\frac{1}{n\prod_{T=1} z_t} \to \infty$

And,

$B \leq \sum_{i=1}^{n} e^{-y_i M(x_i)} \to 0$ therefore $0 \leq B \leq \sum_{i=1}^{n} e^{-y_i M(x_i)}$. So, $B \to 0$ as $T \to \infty$

---

e) Explain why AdaBoost with short decision trees is a form of subset selection when the number of features is large.

This is because short decision trees don't look at a lot of features. If they don't improve the predictive power enough, it won't be used in any of the decision trees.

(d) Use the formulas (2), (3), and (4) to show that if $\text{err}_t \leq 0.49$ for every learner G_t, then $B \to 0$ as $T \to \infty$. Hint: (2) implies that every $Z_t < 0.9998$. How can you combine this fact with (3) and (4)?

(e) Explain briefly why AdaBoost with short decision trees is a form of subset selection when the number of features is large.

3 Movie Recommender System

In this problem, we will build a personalized movie recommender system! Suppose that there are $m = 100$ movies and $n = 24{,}983$ users in total, and each user has watched and rated a subset of the m movies. Our goal is to recommend more movies for each user given their preferences.

Our historical ratings dataset is given by a matrix $R \in \mathbb{R}^{n \times m}$, where R_{ij} represents the rating that user i gave movie j. The rating is a real number in the range $[-10, 10]$: a higher value indicates that the user was more satisfied with that movie. If user i did not rate movie j, $R_{ij} = \texttt{NaN}$.

The provided `movie_data/` directory contains the following files:

- `movie_train.mat` contains the training data, i.e. the matrix R of historical ratings specified above.
- `movie_validate.txt` contains user-movie pairs that don't appear in the training set (i.e. $R_{ij} = \texttt{NaN}$). Each line takes the form "`i, j, s`", where `i` is the user index, `j` is the movie index, and `s` indicates the user's rating of the movie. Contrary to the training set, the rating here is binary: if the user liked the movie (positive rating), `s` $= 1$, and if the user did not like the movie (negative rating), `s` $= -1$.

We also provide `movie_recommender.py`, containing starter code for building your recommender system.

The singular value decomposition (SVD) is a powerful tool to decompose and analyze matrices. In lecture, we saw that the SVD can be used to efficiently compute the principal coordinates of a data matrix for PCA. Here, we will see that SVD can also produce dense, compact featurizations of the variables in the input matrix (in our case, the m movies and n users). This application of SVD is known as Latent Semantic Analysis (Wikipedia), and we can use it to construct a Latent Factor Model (LFM) for personalized recommendation.

Specifically, we want to learn a feature vector $x_i \in \mathbb{R}^d$ for user i and a feature vector $y_j \in \mathbb{R}^d$ for movie j such that the inner product $x_i \cdot y_j$ approximates the rating R_{ij} that user i would give movie j.

(a) Recall the SVD definition for a matrix $R \in \mathbb{R}^{n \times m}$ from Lecture 21: $R = UDV^\top$. Write an expression for R_{ij}, user i's rating for movie j, in terms of only the contents of U, D, and V.

(b) Based on your answer above, what should we choose as our user and movie feature vector representations x_i and y_j to achieve 100% training accuracy (correctly predict all known ratings in R)?

(c) In the provided `movie_recommender.py`, complete the code for part (c) by filling in the missing parts of the function `svd_lfm`. Start by replacing all missing (`NaN`) values in R with 0. Then, compute the SVD of the resulting matrix, and follow your above derivations to compute the feature vector representations for each user and movie. Note: do **not** center the data matrix; this is not PCA.

Once you are finished with the code, the **rows** of the `user_vecs` array should contain the feature vectors for users (so the ith row of `user_vecs` is x_i), and the **rows** of `movie_vecs` should contain the feature vectors for movies (so the jth row of `movie_vecs` is y_j).

Hint: we recommend using `scipy.linalg.svd` to compute the SVD, with `full_matrices = False`. This returns U ($n \times m$), D (as a vector of m singular values in descending order, **not** a diagonal matrix), and $V^\top$ ($m \times m$) in that order.

(3)

a) $R = UDV^T$. write an expression for R_{ij}, user i's rating for movie j, in terms of only the contents of U, D, and V.

$R_{ij} = (x_i D) \cdot y_j$

b) What should we choose as our user and movie feature vector representations x_i, y_j to achieve 100% training accuracy?

User feature vector representations → Rows of UD
movie feature vector representation → rows of V

c) Coding.

Code in appendix.

d) Training performance of the model → MSE → $\sum_{(i,j) \in S} (x_i \cdot y_j - R_{ij})^2$ where $S := \{(i,j) : R_{ij} \neq NaN\}$.

Coding.

Code in appendix.

e) Coding. Comment on which value of d leads to optimal performance.

Code in appendix.

Bud $d = 10$ → val accuracy of 72%.

(d) To measure the training performance of the model, we can use the mean squared error (MSE) loss,

$$\text{MSE} = \sum_{(i,j)\in S} (x_i \cdot y_j - R_{ij})^2 \quad \text{where } S := \{(i,j) : R_{ij} \neq \texttt{NaN}\}.$$

Complete the code to implement the training MSE computation within the function `get_train_mse`.

(e) Our model as constructed may achieve 100% training accuracy, but it is prone to overfitting. Instead, we would like to use lower-dimensional representations for x_i and y_j to approximate our known ratings closely while still generalizing well to unknown user/movie combinations. Specifically, we want each x_i and y_j to be d-dimensional for some $d < m$, such that only the top d features are used to make predictions $x_i \cdot y_j$. The "top d features" are those corresponding to the d largest singular values: use this as a hint for how to prune your current user/movie vector representations to d dimensions.

In your code, compute pruned user/movie vector representations with $d = 2, 5, 10, 20$. Then, for each setting, compute the training MSE (using the function you implemented in part (d)), the training accuracy (using the provided `get_train_acc`), and the validation accuracy (using the provided `get_val_acc`). Plot the training MSE as a function of d on one plot, and the training and validation accuracies as a function of d together on a separate plot. The code for this part is already included in the starter code, so if your training MSE function from part (d) is implemented correctly, the required plots should be saved to your project directory.

Comment on which value of d leads to optimal performance.

Hint: as a sanity check, if implemented correctly, your best validation accuracy should be about 71%.

(f) For sparse data, replacing all missing values with zero, as we did in part (c), is not a very satisfying solution. A missing value in the training matrix R means that the user has not watched the movie; this does not imply that the rating should be zero. Instead, we can learn our user/movie vector representations by minimizing the MSE loss, which only incorporates the loss on rated movies ($R_{ij} \neq \texttt{NaN}$).

Let's define a loss function

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

where S has the same definition as in the MSE. This is similar to the original MSE loss, except with two additional regularization terms to prevent the norms of the user/movie vectors from getting too large.

Implement an algorithm to learn vector representations of dimension d, the optimal value you found in part (e), for users and movies by minimizing $L(\{x_i\}, \{y_j\})$.

We suggest employing an alternating minimization scheme. First, randomly initialize x_i and y_j for all i, j. Then, minimize the above loss function with respect to the x_i by treating the y_j as constant vectors, and subsequently minimize the loss with respect to the y_j by treating the x_i as constant vectors. Repeat these two steps for a number of iterations. Note that when one of the x_i or y_j are constant, minimizing the loss function with respect to the other component has a closed-form solution. **Derive this solution first in your report, showing all your work**.

The starter code provides a template for this algorithm. Start by inputting your best d value from part (e) to initialize the user and movie vectors, and then implement the functions to update the user and movie vectors (holding the other constant) to their loss-minimizing values.

(3) cont.

f) Loss function → $L(\{x_i\},\{y_j\}) = \sum_{(i,j)\in S} (x_i \cdot y_j - R_{ij})^2 + \sum_{i=1}^{n} \|x_i\|_2^2 + \sum_{j=1}^{n} \|y_j\|_2^2$. Derive this solution, showing all your work.

$$\nabla_{x_i} L(\{x_i\},\{y_j\}) = \sum_{(i,j)\in S} 2(x_i \cdot y_j - R_{ij}) y_j + 2x_i$$

$$= 2\sum_{(i,j)\in S} (y_j y_j^T x_i - R_{ij} y_j) + 2x_i$$

To find the optimal solution $2\sum_{(i,j)\in S} (y_j y_j^T x_i - R_{ij} y_j) + 2x_i = 0$.

$$\sum_{(i,j)\in S} (y_j y_j^T x_i - R_{ij} y_j) + x_i = 0$$

$$\sum_{(i,j)\in S} (y_i y_j^T + I) x_i = \sum_{(i,j)\in S} R_{ij} y_j$$

$$\boxed{x_i = \left(\sum_{(i,j)\in S} (y_i y_j^T + I)\right)^{-1} \left(\sum_{(i,j)\in S} R_{ij} y_j\right)}$$

And Now for y_j:

$$\nabla_{y_j} L(\{x_i\},\{y_j\}) = \sum_{(i,j)\in S} 2(x_i \cdot y_j - R_{ij}) x_i + 2y_j$$

$$= 2\sum_{(i,j)\in S} (x_i x_i^T y_j - R_{ij} x_i) + 2y_j$$

Setting to zero and solving:

$$2\sum_{(i,j)\in S} (x_i x_i^T y_j - R_{ij} x_i) + 2y_j = 0$$

$$\sum_{(i,j)\in S} (x_i x_i^T y_j - R_{ij} x_i) + y_j = 0$$

$$\sum_{(i,j)\in S} (x_i x_i^T + I) y_j = \sum_{(i,j)\in S} R_{ij} x_i$$

$$\boxed{y_i = \left(\sum_{(i,j)\in S} (x_i x_i^T + I)\right)^{-1} \left(\sum_{(i,j)\in S} R_{ij} x_i\right)}$$

Question 3: Movie Recommender System

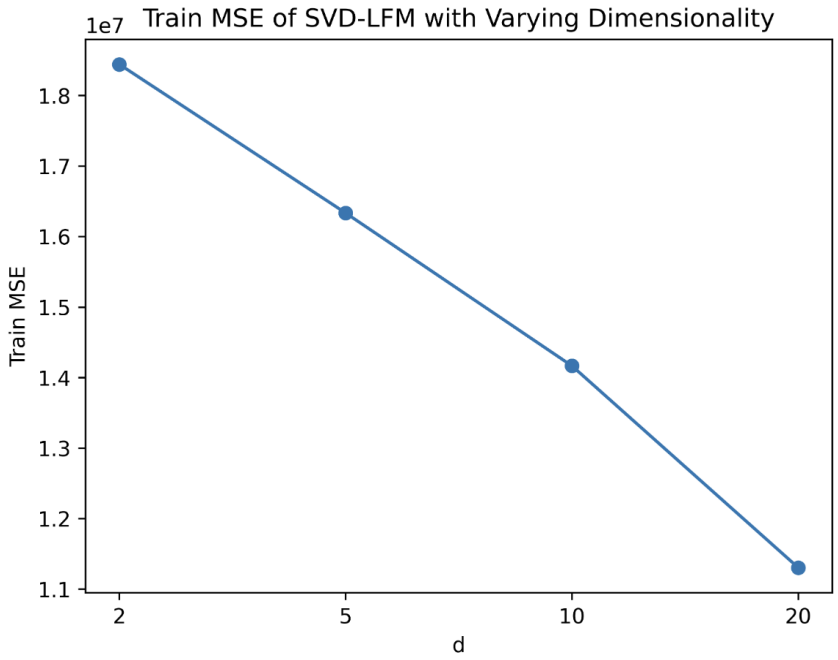
Part a) Expression

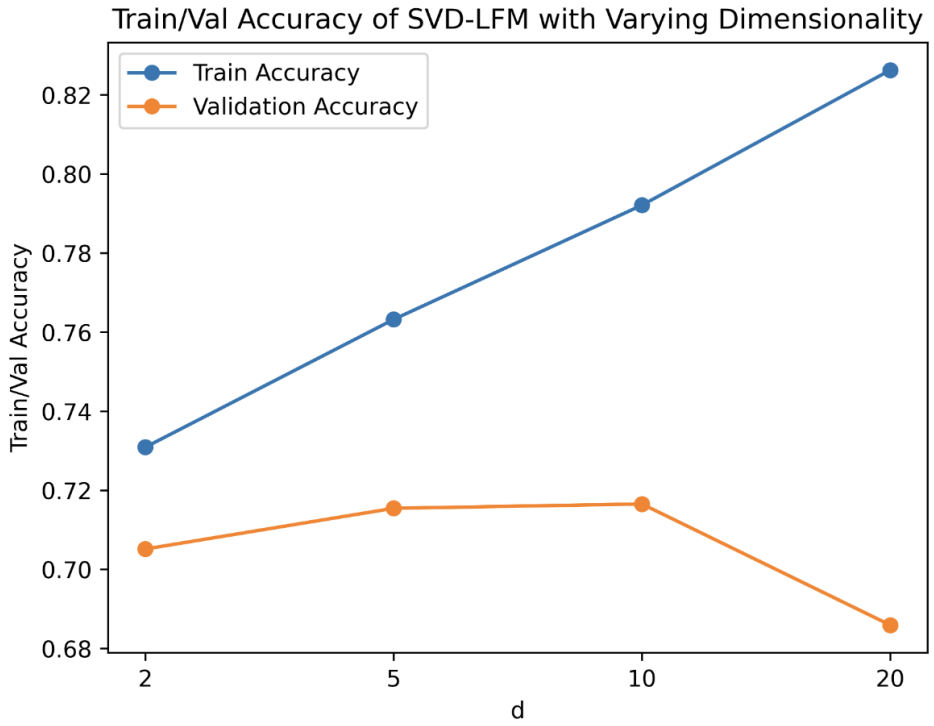
Part b) What should we choose as our user and movie feature vector representations?

Part c) Coding

Part d) Coding

Part e) Comment on which value of d leads to optimal performance.





From the plot, we can see that the best validation accuracy is around 72% at d=10.

Part f) Derive the solution first. Coding.

```
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
```

At iteration 20, I get training MSE = 9314221.76 = 9.31 e^7 , Training accuracy = 0.8163 = 81.63% , and Validation accuracy = 0.7160 = 71.60% which is close to the validation accuracy in part e) which was around 72%

- To improve efficiency, we recommend using the `user_rated_idxs` and `movie_rated_idxs` arrays provided, which contain the indices of movies that each user rated and the indices of users that rated each movie (respectively), to iterate through the non-`NaN` values of R in the update functions.
- Run these 2 update steps for 20 iterations. Include your final training MSE, training accuracy, and validation accuracy on your report, and compare these results with your best results from part (e).

4 Nearest Neighbors for Regression, from A to Z

For this problem, we will use data from the UN to have some fun with the nearest neighbors classifier. You'll be modifying starter code in the provided `world_values` directory.

We are using the "World Values Survey" dataset, collected over several years from many countries. The survey asked, "Which of these are most important for you and your family?" There were 16 possible responses, including needs like "Freedom from Discrimination and Persecution" and "Better Transport and Roads." The data reported is the fraction of responses in each country that chose each option.

We would like to use these 16 features of each country (citizens' responses to the survey) to predict that country's HDI (Human Development Index), a value between 0 and 1. In reality, the HDI is a complex measure that accounts for factors like a country's life expectancy, education, and per capita income. Intuitively, though, you would expect citizens of countries with different HDI to have different priorities. For that reason, it may be reasonable to predict the HDI from survey data. (Note: throughout the problem we use RMSE, which stands for Root Mean Squared Error.)

(a) Let's visualize the data. Using sklearn, apply PCA to the data in the plot_pca method of world_values_utils.py. Plot the data in its first two PCA dimensions, colored by HDI.

(b) In lecture, we covered k-nearest neighbors algorithms for classification problems. We decided that the class of a test point would be the plurality of the classes of the k nearest training points. That algorithm makes sense when the outputs are discrete, so we can vote. Here, the outputs are continuous. How would you adapt the k-nearest neighbors classifier for a regression problem? (This is an open-ended question with several possible answers.)

(c) Modify the starter code in world_values_starter.py to find the 7 nearest neighbors of the USA. Which countries are the USA's 7 nearest neighbors (in order) from the data given?

(d) The main hyperparameter of k-nearest neighbors is k itself. Use grid search in world_values_starter.py to create a plot of the RMSE of k-NN regression versus k, where k is the number of neighbors. Include your plot in your write-up. What is the best value of k? What is the RMSE?

(e) Explain your plot in (d) in terms of bias and variance. Think about the spirit of bias and variance more than their precise definitions.

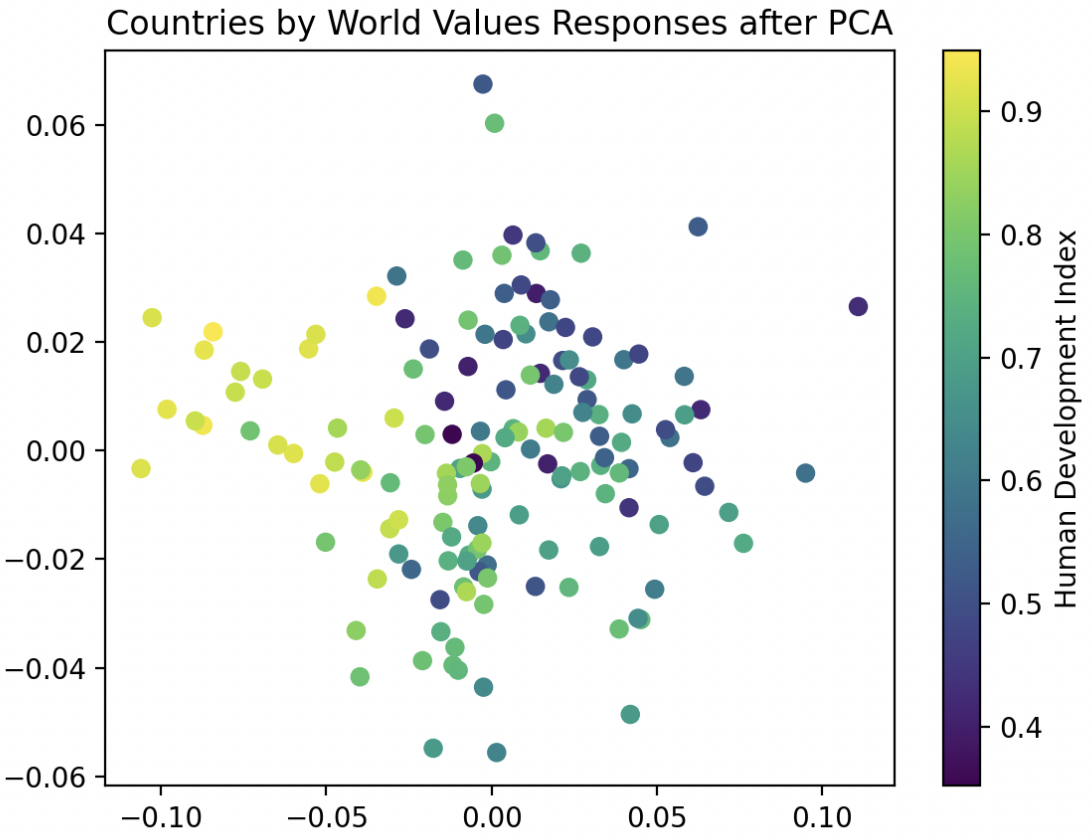
(f) We do not need to weight every neighbor equally: closer neighbors may be more relevant. For this problem, weight each neighbor by the inverse of its distance to the test point by modifying world_values_parameters.py. Plot the RMSE of k-NN regression with distance weighting vs. k, where k is the number of features. What is the best value of k? What is the RMSE? What happens as k gets very large, compared to part (d)?

(g) One of the drawbacks of the k-nearest neighbors classifier is that it is very sensitive to the scale of the features. For example, if one feature takes on values 0 or 0.1 and another takes on values 0 or 10, then neighbors will almost certainly agree in the second feature.

Add normalization to your k-nearest neighbors pipeline (continue to use distance weighting). Plot RMSE versus k. What is the best value of k? What is the RMSE?

Question 4: Nearest Neighbors for Regression, from A to Z

Part a) Plot data.



Part b) How would you adapt the k-nearest neighbors classifier for a regression problem?

Part c) Coding. Which countries are the USA's 7 nearest neighbors from the data given?

USA's 7 nearest neighbors:

1 Ireland

2 United Kingdom

3 Belgium

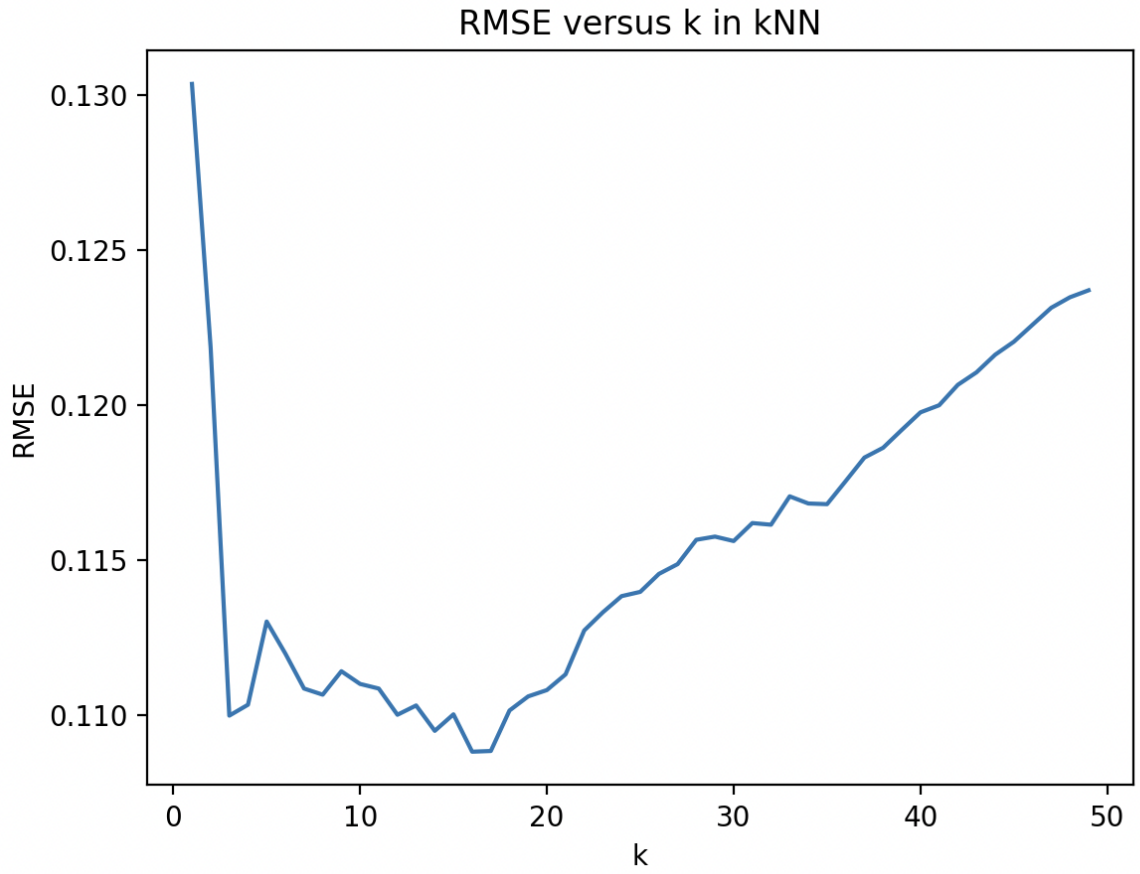
4 Finland

5 Malta

6 Austria

7 France

Part d) Coding. Plot the RMSE. What is the best value of k? What is RMSE?



RMSE: 0.10881912358683599

Best k value is the value that balances the bias and variance, with the lowest RMSE. So, it will be k = 16.

Part e) Explain the plot in d) in terms of bias and variance.

The RMSE decreases as k increases, but eventually levels off and starts to increase again as k gets too large.

In terms of bias and variance: When k is small, the model is overfitting, so it has a low bias, but high variance which leads to high RMSE.

As k increases, the models becomes more simplified, causing the bias to start increasing and variance to decrease, which leads to lower RMSE.

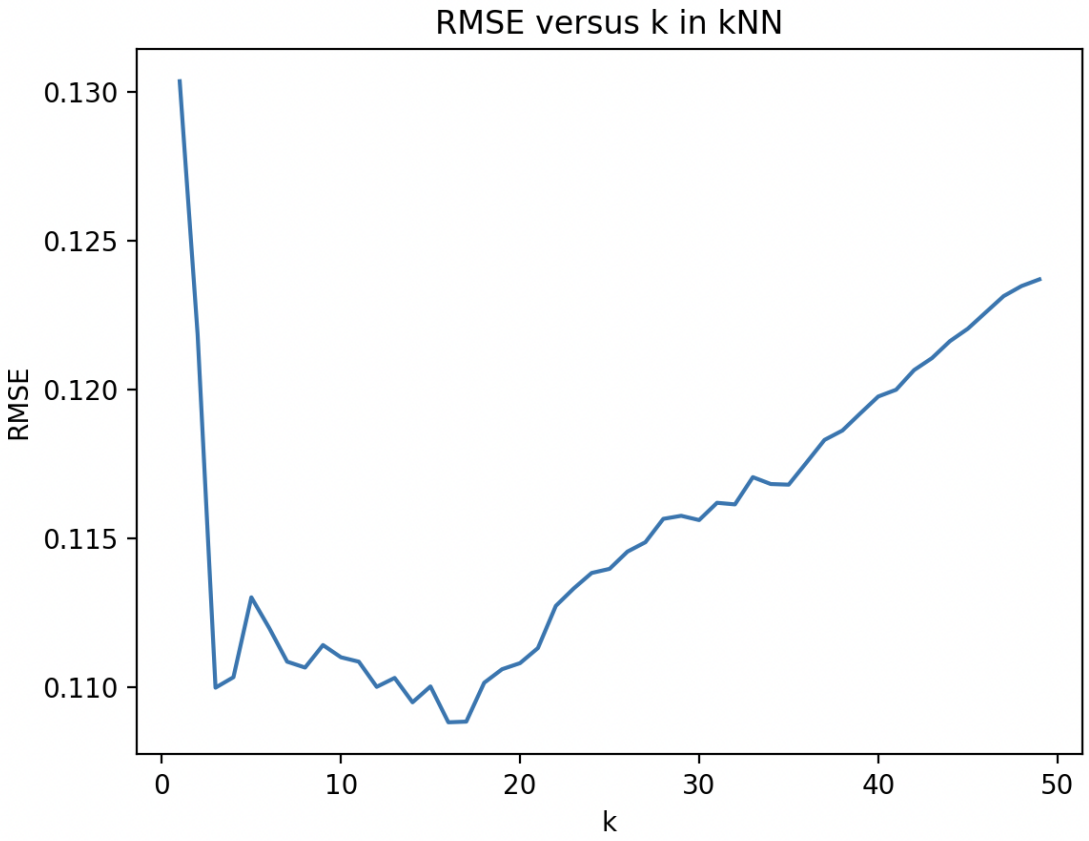
Eventually, the bias becomes too high and the RMS starts to increasing again, causing high bias and low variance again which shows that the model is underfitting.

This plot shows that for the optimal value of k, we want the bias and variance to balance each other out and that is when the RMSE is at its lowest point.

Smaller k → lower bias , high variance

Big k → High bias, low variance

Part f) Coding. Plot the RMSE. What is the best value of k? What is RMSE? What happens as k gets very large, compared to d)?

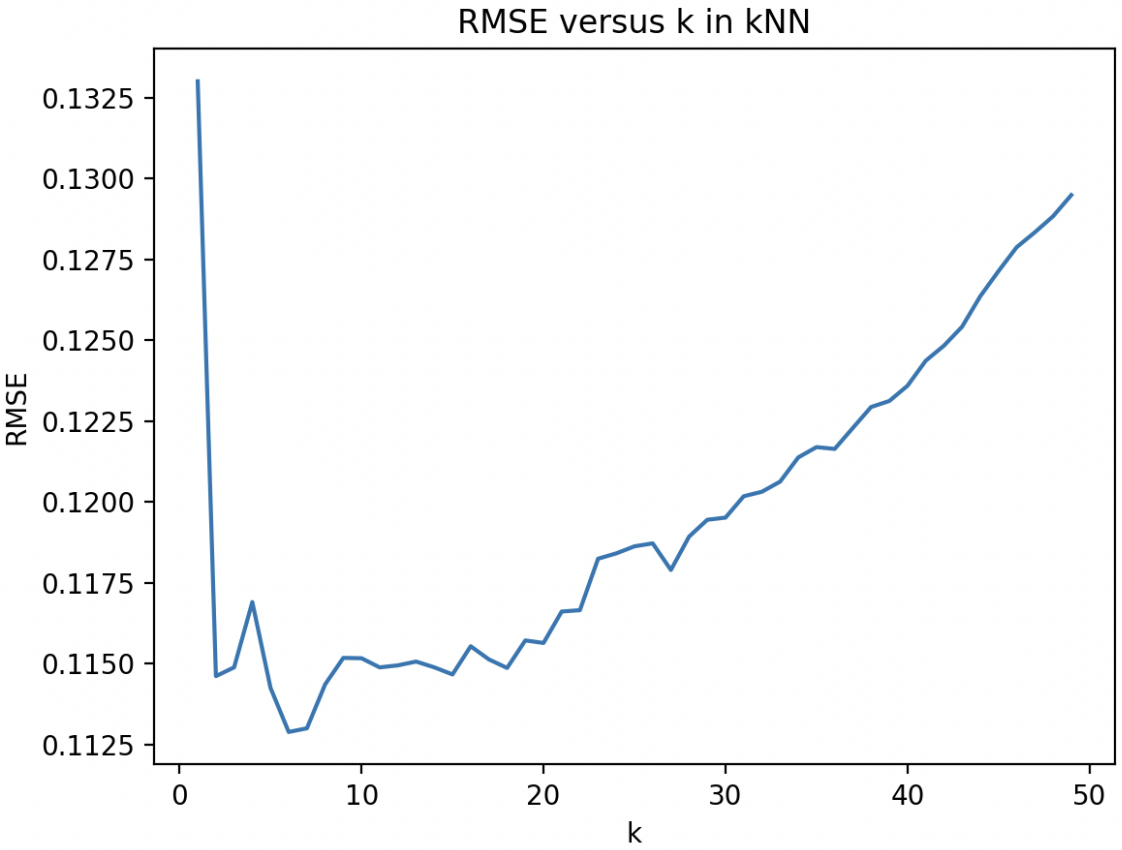


RMSE: 0.10881912358683599

Best k value is the value with the lowest RMSE. So, it will be k = 16.

The RMSE increases at a slower rate in this case compared to part d)

Part g) Coding. Plot the RMSE. What is the best value of k? What is RMSE?



RMSE: 0.11289612128761553

Pipeline(steps=[('scale', StandardScaler()),

('knn', KNeighborsRegressor(n_neighbors=6))])

The best value for k is the value with the lowest RMSE. So, it will be k = 8 now.

Appendix:

Question 2: The raining Error of AdaBoost

No coding.

Question 3: Movie Recommender System

Part a) No coding.

Part b) No coding.

Part c)

```
import os
import scipy.io
import numpy as np
import scipy.linalg
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:
```

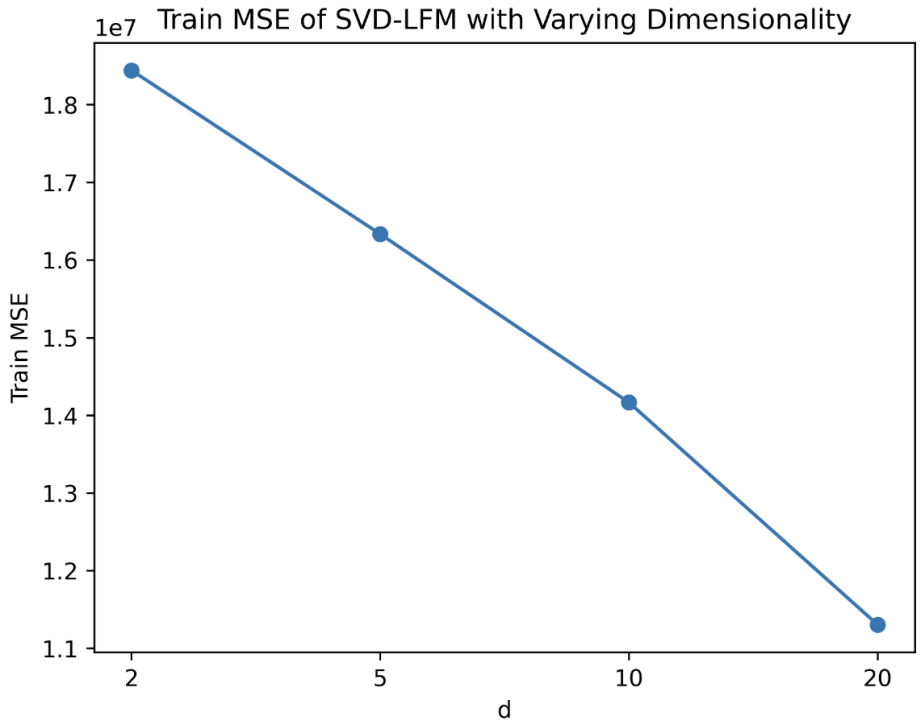
```
            num_correct += 1
    return num_correct/val_data.shape[0]

# 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=object)

# 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 #####
    # Start by replacing all missing (NaN) values in R with 0
    R[np.isnan(R)] = 0

    # Compute the SVD of R
    ##### TODO(c): Your Code Here #####
    # compute the SVD of the resulting matrix
    # we recommend using scipy.linalg.svd to compute the SVD, with full matrices =
False.
    U, D, Vt = scipy.linalg.svd(R, full_matrices=False)

    # Construct user and movie representations
    ##### TODO(c): Your Code Here #####
    #  follow your above derivations to compute the feature vector representations
for each user and movie.
    user_vecs = np.dot(U, np.diag(D))
    movie_vecs = Vt.T

    return user_vecs, movie_vecs
```

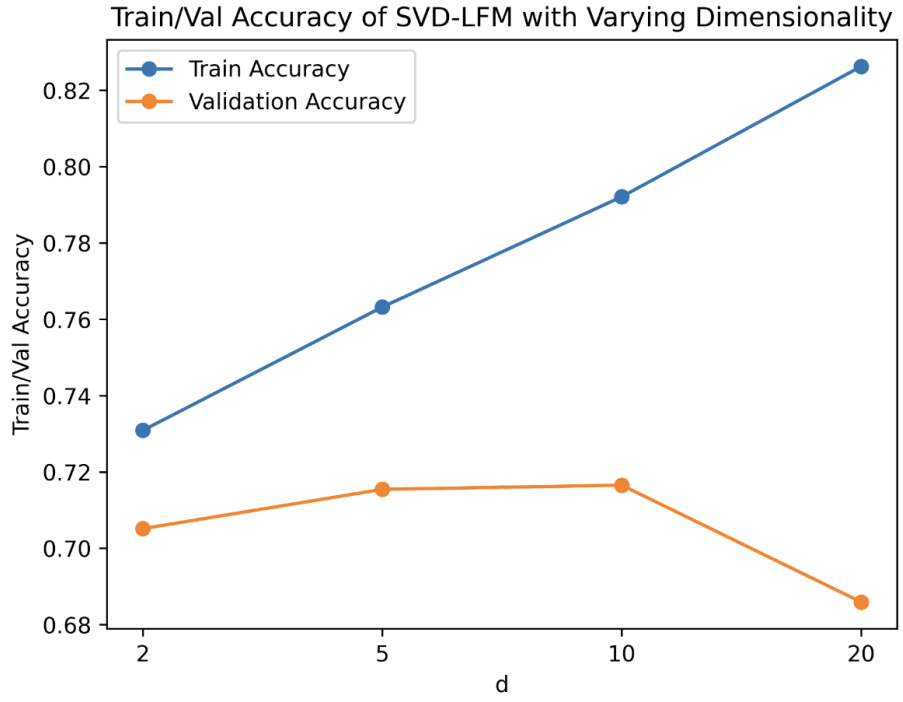
Part d)

```
# 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 #####
    # Formula: MSE = (sumforall(i,j)inS)(xi·yj-Rij)^2 where S:={(i,j):Rij!=NaN}
    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 = (sumforall(i,j)in S)(xi·yj-Rij)^2
                mse_loss += (np.dot(user_vecs[i], movie_vecs[j]) - R[i, j])**2

    return mse_loss
```

Part e) Comment on which value of d leads to optimal performance.

```
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')
```





From the plot, we can see that the best validation accuracy is around 72% at d=10.

Part f) Derive the solution first. Coding.

```
# Part (f): Learn better user/movie vector representations by minimizing loss
# begin solution
best_d = 10 # TODO(f): Use best from part (e) --> From the plot of part e) 10 seems
to be the best d.
# 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 #####
    # My derived expression for user_vecs is xi = ((sumforall(i,j)inS (y_i(Y_i^T) +
I))^-1 ) ( (sumforall(i,j)inS (R_ij.Y_j)) )
    for i in range(R.shape[0]):
        x = np.eye(user_vecs.shape[1])
        y = np.zeros_like(movie_vecs[0])
        for j in user_rated_idxs[i]:
            x += np.outer(movie_vecs[j], movie_vecs[j])
            y += R[i, j] * movie_vecs[j]
        user_vecs[i] = np.dot(np.linalg.inv(x), y)
    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 #####
    # My derived expression for movie_vecs is yi = (sumforall(i,j)inS (x_i(x_i^T) +
I))^-1 (sumforall(i,j)inS (R_ij.x_j))
    for j in range(R.shape[1]):
        X = np.eye(user_vecs.shape[1])
        Y = np.zeros_like(user_vecs[0])
        for i in movie_rated_idxs[j]:
            X += np.outer(user_vecs[i], user_vecs[i])
            Y += R[i, j] * user_vecs[i]
        movie_vecs[j] = np.dot(np.linalg.inv(X), Y)

    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 accuracy: {val_acc:.4f}')
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_idxs)
   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:
{train_acc:.4f}, val accuracy: {val_acc:.4f}')
```

```
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
```

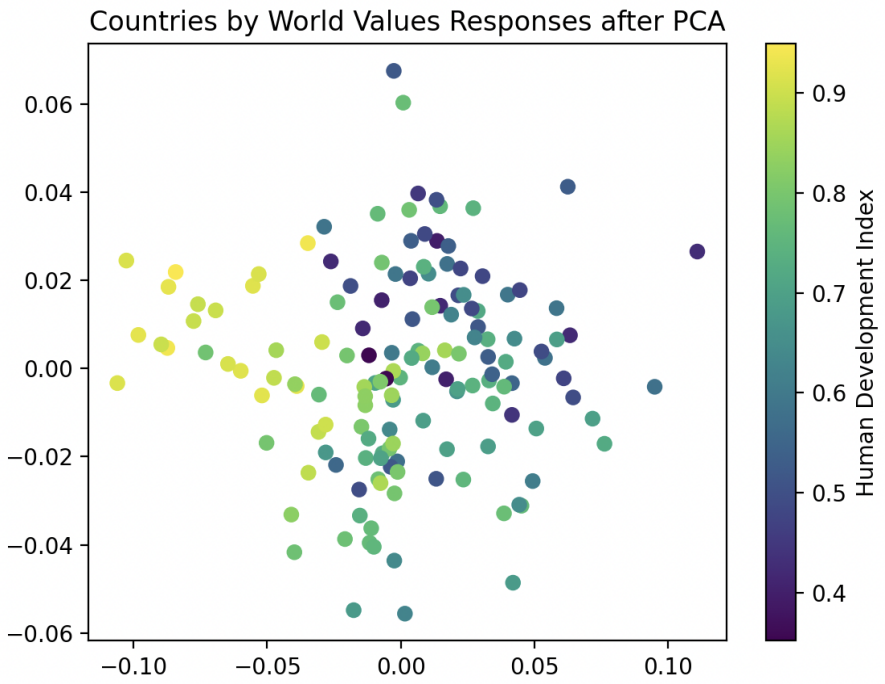
I get training MSE = 9314221.76 = 9.31 e^7 , Training accuracy = 0.8163 = 81.63% , and Validation accuracy = 0.7160 = 71.60% which is close to the validation accuracy in part e) which was around 72%

Question 4: Nearest Neighbors for Regression, from A to Z

Part a) Plot data.

```
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
training set

    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 #####

    # transformed_features = ...
    pca = PCA(n_components=2)
    transformed_features = pca.fit_transform(training_features)
    # print(transformed_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')
```



Part b) No coding.

Part c) Coding. Which countries are the USA's 7 nearest neighbors from the data given?

```
"""
The world_values data set is available online at http://54.227.246.164/dataset/. In
the data,
   residents of almost all countries were asked to rank their top 6 'priorities'.
Specifically,
   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
several models
   that predict the HDI (Human Development Index) rating of a country from the
responses of its
   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)

    # 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)
    print("USA's 7 nearest neighbors:")
    for i, idx in enumerate(indices[0][1:]):
        print(i+1, countries[idx])
    # Part D: complete _rmse_grid_search to find the best value of k for Regression
Grid Search #

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

USA's 7 nearest neighbors:

1 Ireland

2 United Kingdom

3 Belgium

4 Finland

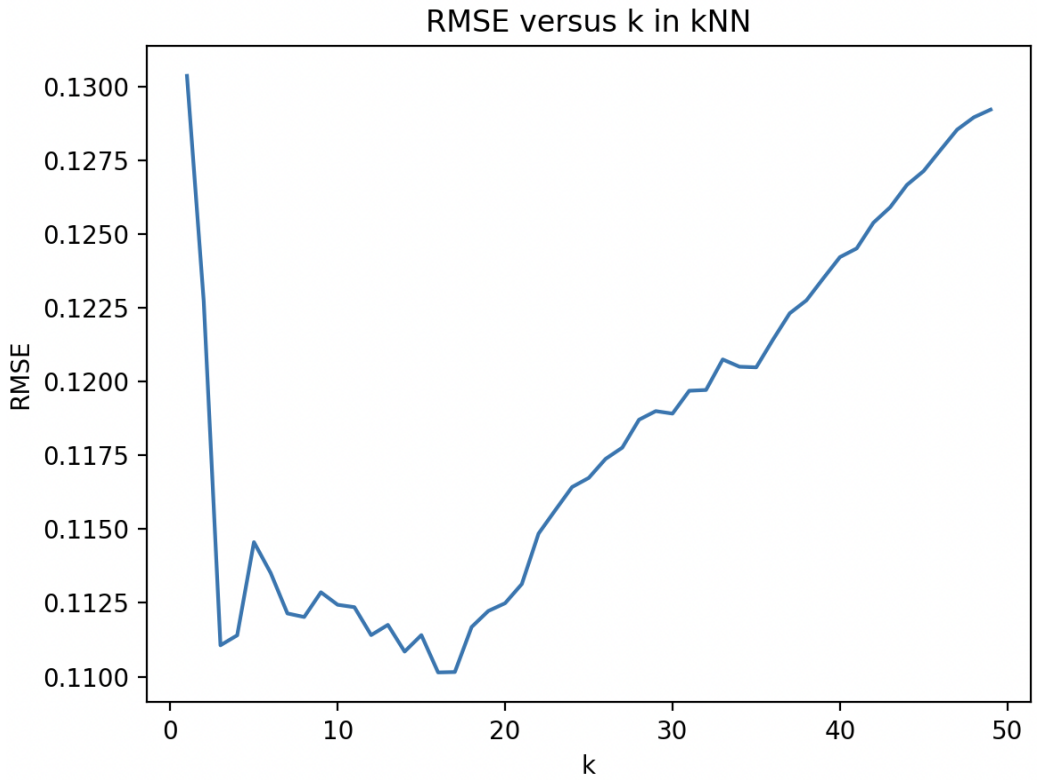
5 Malta

6 Austria

7 France

Part d) Coding. Plot the RMSE. What is the best value of k? What is RMSE?

```
def _rmse_grid_search(training_features, training_labels, pipeline, parameters,
technique):
    """
    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 = ...
    grid = GridSearchCV(pipeline, parameters, cv=5,
scoring='neg_mean_squared_error')
    grid.fit(training_features, training_labels)

    print("RMSE:", sqrt(-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'])**0.5)
    plt.xlabel('k')
    plt.ylabel('RMSE')
    plt.title('RMSE versus k in kNN')
    plt.show()


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



RMSE: 0.11014306152198958

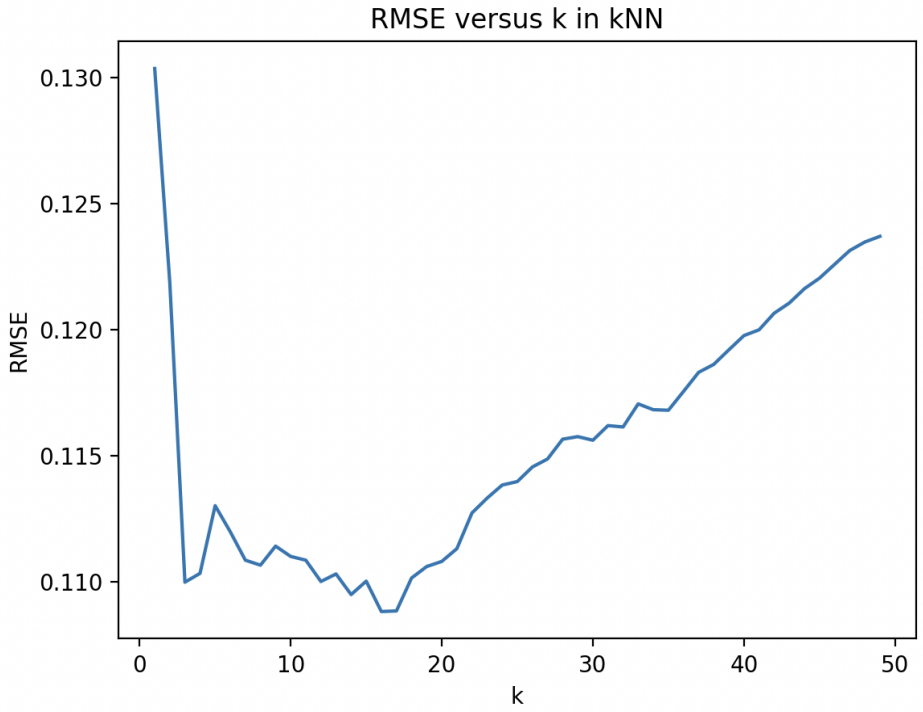
Best k value is the value with the lowest RMSE. So, it will be k = 16.

Part e) No coding.

Part f) Coding. Plot the RMSE. What is the best value of k? What is RMSE? What happens as k gets very large, compared to d)?

'knn__weights': ['distance'] instead of 'knn__weights': ['uniform'] for this part

```
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': ['uniform']
   # Uncomment the following code for part d)
   # 'knn__weights': ['distance']
}
```



RMSE: 0.10881912358683599

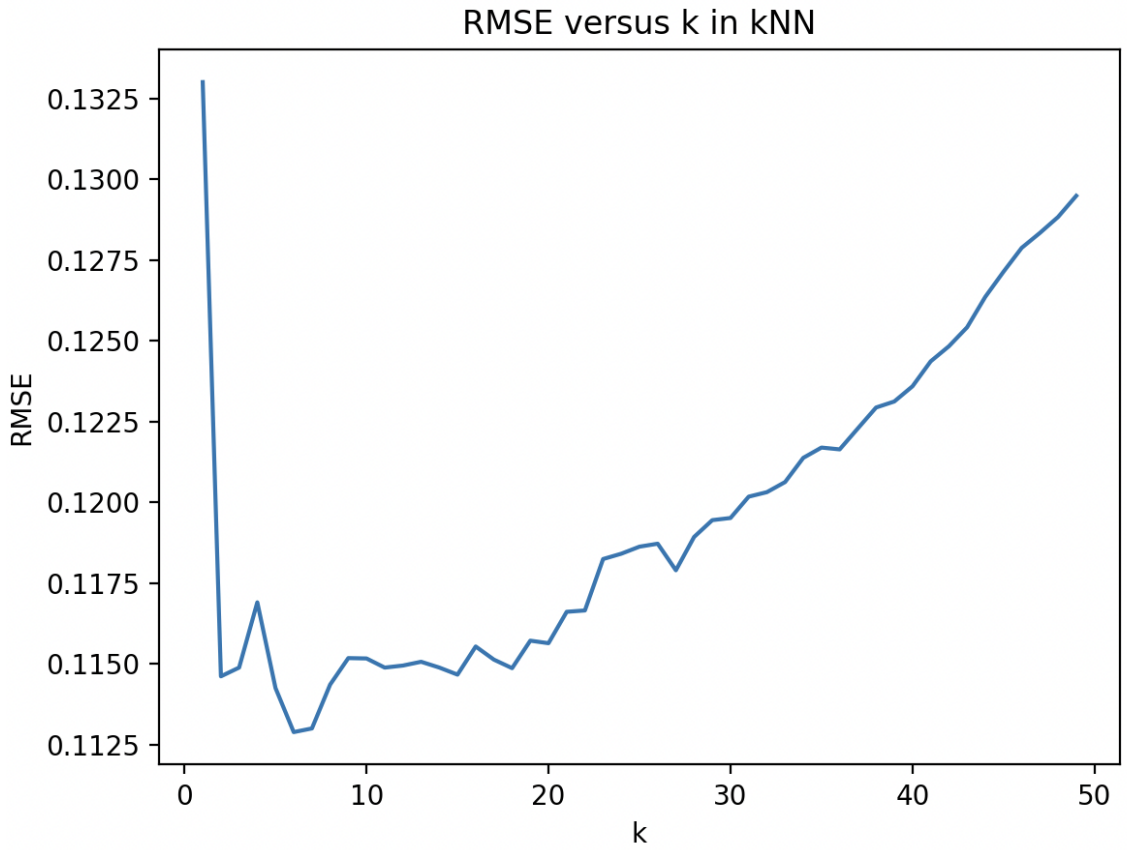
Best k value is the value with the lowest RMSE. So, it will be k = 16.

The RMSE increases at a slower rate in this case compared to part d)

Part g) Coding. Plot the RMSE. What is the best value of k? What is RMSE?

Just added the scale to add standardization ('scale', StandardScaler()),

```
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()
#####
          # Uncomment the following line for part g)
          # ('scale', StandardScaler()),
          ('knn', KNeighborsRegressor())
      ]
  )
```



RMSE: 0.11289612128761553

```
Pipeline(steps=[('scale', StandardScaler()),
                ('knn', KNeighborsRegressor(n_neighbors=6))])
```

The best value for k is the value with the lowest RMSE. So, it will be k = 8 now.