Homework 4: Unsupervised Learning

Principles of Machine Learning (Wentworth Institute of Technology)

Fall 2024 Semester

Without labels in our training data, unsupervised learning is more interested in finding hidden patterns that describe the data. Experts often look at those patterns to either interpret the data or use them in some way to train supervised learning models, which brings them all together (get it, clustering is an unsupervised learning task that groups related data together?) as a suite of tools. Please respond to each item below, which can be a question to answer and/or task to perform. *You must show your work in order to receive full credit for a correct response, and your work will determine partial credit for incorrect responses.* If you choose to work with others in the class, please remember that everyone must turn in their own version of the responses to the items in this assignment. You must mention with whom you collaborated as well (and for that matter, please remember to include your name on the submission).

When working on problems in the wild, be it for work or pleasure, the world will neither tell you the specific problem nor what to do to solve this problem. Customers and/or clients come to you because they do not know how to do it themselves, which means they can only give a lot of context attempting to describe what they want you to achieve. The world itself has lots of things you can observe and loves to throw life challenges at us. Some of the information from these sources will be useful, and the rest is either a red herring or irrelevant. These problems are phrased in a similar way, and your solutions will need you to identify both the problem(s) and which information is relevant to solve them.

1. Continuing to grow our suite of ML algorithms, we will implement a simple K-Means clustering learner. Continuing with your matrix-based programming language (Matlab, Octave, Julia, R, etc.) or package (Python's numpy and scipy, C++'s Boost, etc.) of choice, grab the general classes you prepared from Homework 2 as your starting point. It is highly encouraged you TEST that your code works as intended after each step so that any problems in the next step are easier to diagnose. *Good programming practices you should use in this assignment are listed in the back as an appendix.*

   (a) ***(30 points)*** Starting from a general class or data structure that should describe the properties and capabilities of any machine learning algorithm, we can implement a more specific (sub)class or data structure for K-Means clustering. Its training and prediction functions should follow these specifications:

   TO-DO: Implement a K-Means clustering class as specified below in the other TO-DO segments. When following the good programming practices, remember that you should include comments that document what parts of the code do. Because the TO-DO segments only describe what code to write, use the context provided before the TO-DO segments to determine what comments to provide that explain what the code does.

   Train Function

   i. The inputs to the train function should always be the hyperparameters (as an array or hash map based on your implementation in Homework 2) and the training data as inputs X (a matrix) and ground truth outputs $\overrightarrow{y}$ (a vector, which is effectively a matrix with exactly one column). This is not a typo; $\overrightarrow{y}$ is still a function input even though unsupervised learning does not provide training data labels.

   TO-DO: Implement the K-Means clustering class as a subclass of the generic ML algorithm class, and setup the train function that the subclass will overwrite. For now, this

overwrite is an empty function with only a name, the parameter list, and any additional syntax your programming language of choice requires to write a function (examples are curly braces, colons, keywords like `pass` or `return`, etc.). Answer the following question: Why does the class hierarchy we designed require us to include $\overrightarrow{y}$ as a function parameter for training? What are some things users can do to avoid this being a problem when calling the K-Means clustering train function[1]? If the training function never uses $\overrightarrow{y}$ in any of its lines of code, then why does it ultimately not matter what the user does?

ii. Let one of the hyperparameters the user sets be the number of clusters to learn K, let another hyperparameter the user sets be the threshold for clustering convergence τ, and let a third hyperparameter the user sets be the maximum number of iterations before forcing the clustering process to stop s. Check these hyperparameters to confirm that they are appropriate values.

TO-DO: Implement an if-then-else check within the train function that checks for the value assigned to each hyperparameter. These checks should make sure all the hyperparameters are non-negative numbers, and the number of clusters should be an integer.

iii. Before we can start the clustering process, we need to initialize and setup a few variables that will keep track of the cluster information. The parameters for the K-Means clustering model will store the centroids of each of the K clusters, and each sample/row of the dataset will need an "additional feature" that stores its current cluster assignment. We will also keep track of the number of samples that belong to each cluster in order to perform later computations. Lastly, in order to check for convergence, we will need a variable that stores the previous parameter assignment.

TO-DO: Set the class's learned parameters to a matrix with K rows and as many columns as X, but the values assigned to each entry do not yet matter. Define another matrix variable that will copy the matrix of learned parameters. After this, append a column to X that will keep track of the assigned cluster. We will also need to define a vector (matrix with one row/column) variable that stores K entries, and the values assigned to each entry should be 0. Answer these questions: Why do we care about initializing the variable representing the number of samples in a cluster to store all zeroes? Why do we not care about the values of the parameter matrix upon initialization? What does this tell us about initializing variables whose assigned values will be accumuluated vs. replaced?

iv. Before we can begin the iterative clustering process, we need a starting point from which we can iteratatively apply updates to clusters. Without any knowledge about the labels for each sample in X, randomly assign each sample to one of the K clusters.

TO-DO: Implement a loop that iterates over each sample/row of X. Within this loop, assign a number variable to an integer between 1 and K (or between 0 and $(K-1)$ if your programming language uses 0-indexing and you want to avoid having to subtract 1 when accessing cluster information later[2]). Still within the loop, assign the appended column of the current row to the random number stored in the previous variable. Then, still within the loop, increment the entry of the vector counting the number of samples per cluster that corresponds to the stored random number[3] by 1.

v. With every sample/row of X assigned to an initial cluster, we can now perform the iterative updating step. In each iteration, we will compute the centroid of each cluster as

[1] "Problems" refers to the compiler or interpreter complaining that the user called the train function without providing arguments for all the required parameters.

[2] The speed-up will be worth it to avoid subtracting 1 later because (1) many datasets are large and (2) we will perform this iteration a lot. Keep in mind from Homework 1 just how much time you can save by avoiding an arithmetic step inside a loop.

[3] See? We already have to subtract 1 to get the correct entry if you did not do it with the random number assignment and have a 0-indexing programming language.

the average of feature vectors in that cluster

$$\overrightarrow{\overrightarrow{x_k}} = \frac{1}{|C_k|} \sum_{\overrightarrow{x_i} \in C_k} \overrightarrow{x_i},$$

compute the distances between each sample and the centroids, and then reassign the samples' clusters based on the closest centroid to each sample. To keep things simple, we will use Euclidean distance (L_2-norm) for finding the closest centroid

$$dist\left(\overrightarrow{v_1}, \overrightarrow{v_2}\right) = ||v_1 - v_2||_2 .$$

We break down the bigger steps into smaller ones, and it will be good practice to think about how each high-level step became all these smaller steps because you will rarely have someone breaking it down for you.

TO-DO: (Find new centroids) After the previous loop, define a boolean variable that will check for convergence; initialize this variable to `False`. Then, implement a new loop that iterates until convergence evaluates to `True` or we performed the maximum number of iterations s. Store the previous parameter values into the other matrix variable you defined earlier in the code, and then assign all the parameters to 0 after backing them up. Within this loop, implement another loop that iterates over all the samples/rows of X. In this inner loop, add the current sample's feature vector (this excludes the assigned cluster column you appended earlier) to the row of the learned parameter matrix corresponding to the assigned column for the current sample. After this inner loop completes, still within the outer loop, implement another loop that runs K times, once per cluster. Within this inner loop, multiply the current row of the learned parameters matrix by the scalar that divides 1.0 by the number of samples currently assigned to the current cluster. Answer the following question: Why do we need to divide 1.0, rather than 1, by the number of current samples/rows assigned to a specific cluster?

TO-DO: (Reassign clusters) After the inner loop completes, still within the outer loop, implement another loop over the samples/rows of X. Within this inner loop, define a number variable assigned to the greatest possible value a number variable type in your programming language of choice can store. Then, still within the inner loop, implement a loop that runs K times, once per cluster. In this innermost loop, create an if-statement whose condition compares the stored value in the recently-defined variable against the magnitude of the difference between the current sample's feature vector (from the middle-nested loop, again excluding the appended column) and the centroid stored in the current cluster index's row of the learned parameter matrix (from the innermost loop). When the magnitude is less than the stored value, then store this magnitude into the variable (replacing the previous value) and change the current sample's cluster assignment to the current iteration of the innermost loop[4]. Answer the following question: Why does setting the newly defined variable to the greatest possible assignable value guarantee that we find the minimum distance between the current sample and all the centroids during if-statement comparisons?

vi. After computing cluster assignments and centroids (model parameters) for the current iteration, we need to check whether another iteration is likely to reduce the "error" further. This would require enough samples to change between clusters so that the centroids drastically change. Otherwise, this process repeats ad nauseam without any changes to the centroids or cluster assignments per sample. This lack of change between iterations is called ***convergence***, and we can terminate the program early to avoid redundant calculations. Check for convergence using a comparison of the centroids (learned parameters) between this iteration and the previous iteration.

[4]This is between 0 and $(K-1)$ or between 1 and K, as discussed in previous footnotes.

TO-DO: After the two inner loops complete, still within the outer loop, set the boolean variable checking for convergence to the product of `True` and the current iteration of the outer loop[5]. Then, implement a loop that runs K times, once per cluster. Within this inner loop, set the boolean variable to the conjunction (`and` operator) of itself and a comparison that checks whether the threshold τ is greater than the magnitude of the difference between the current row (centroid) of the learned parameter matrix and the current row (centroid) of the other matrix variable storing the previously learned parameter matrix. Answer the following question: What properties of checking for convergence justify the default assessment being true—that is, why do we change the assessment to false rather than start with false and then change it to true? How is this similar to lazy boolean evaluation that most programming languages use to save computation time when evaluating conjunctions and disjunctions?

Predict Function

i. The input to the predict function should always be the inputs X.

TO-DO: Setup the predict function that the subclass will overwrite. For now, this overwrite is an empty function with only a name, the parameter list, and any additional syntax your programming language of choice requires to write a function (examples are curly braces, colons, keywords like `pass` or `return`, etc.).

ii. An important sanity check, if you did not implement this for the generalized template, is that the parameters have been learned already. If not, be sure to inform the user and terminate the program gracefully (throw an error/exception or trigger an assertion violation).

TO-DO: At the start of the predict function, check whether the class instance's value for the parameters is set to a non-default value (typically null). If it is the default value, then throw an error/exception or trigger an assertion violation that provides a message about needing to train the model first.

iii. Without any formal matrix operations to perform in K-Means clustering classification, we must predict each row of X independently. For a single input row $\overrightarrow{x}$, compare its distance to all the learned clusters' centroids and select the cluster whose centroid is closest as the prediction. Include all the predicted labels in a vector $\overrightarrow{z}$ and return it.

TO-DO: After the check of the parameters, define a vector or matrix variable that will represent $\overrightarrow{z}$. Scoping requires this variable's definitions to appear before other blocks such as loops if we plan to use them after that block of code executes. Implement a loop over the rows $\overrightarrow{x}$ of X, and define a number variable at the start of this loop assigned to the greatest possible value a number variable type in your programming language of choice can store. Then, still within the loop, implement a loop that runs K times, once per cluster. In this inner loop, create an if-statement whose condition compares the stored value in the recently-defined variable against the magnitude of the difference between $\overrightarrow{x}$ and the centroid stored in the current cluster index's row of the learned parameter matrix. When the magnitude is less than the stored value, then store this magnitude into the variable (replacing the previous value) and change the corresponding entry of $\overrightarrow{z}$ to the current iteration of the inner loop[6]. After the both nested loops complete, return $\overrightarrow{z}$. Answer these questions: If we had access to a GPU, then could we perform each prediction in parallel; why or why not? How much does the prediction function have in

[5]Most programming languages support `False` and 0 as interchangeable, and this also supports `True` and any non-0 number as interchangeable. If this is not the case for your programming language of choice, then please adjust accordingly so that the first iteration cannot converge. Ideally, this is doable without an if-statement because it will be a disruption to control flow in the remaining iterations of the loop, which will greatly slow down your program's execution.

[6]Time to sound like a broken record; you are not experiencing déjà vu. This is between 0 and $(K-1)$ or between 1 and K, as discussed in previous footnotes.

common with the training function for K-Means clustering? How could we take advantage of these overlaps to reduce the amount of code we have to write?

2. ***(20 points)*** Because it is very little effort, why not add another classifier to our suite of learning algorithms? This will be a semi-supervised learning algorithm based on K-Means clustering. Create a subclass of the K-Means clustering class you just programmed, and override the train function with a change to the step that performs the initial cluster assignments (they are no longer random because you have some labels)—we compute centroids using the labeled data during the first iteration, and we do not reassign labels to the samples with provided labels. The rest of the train function is the same, and the prediction function is identical. Yes, that's it!

   TO-DO: Implement the semisupervised K-Means clustering class as a subclass of the generic ML algorithm class, and setup the train function that the subclass will overwrite. To start, simply copy-paste the code from the train function from the unsupervised version (superclass's). Then, include a new vector variable at the start of the function with as many entries as the rows of X that will store boolean values determining for which samples/rows $\overrightarrow{y}$ included a label. During the initial cluster assignment loop, replace the random assignment with the value from $\overrightarrow{y}$[7]. When looping over the samples/rows to compute each centroid, use the `continue` keyword to skip the current sample/row when its assigned cluster does not exist. Then, when looping over the samples/rows to assign their new cluster, use the `continue` keyword to skip the current sample/row when $\overrightarrow{y}$ assigned its label. Answer the following question: To avoid editing a copy-paste of the code from the unsupervised K-Means clustering superclass, how could we replace some parts of the train function in the superclass with helper functions to reduce the amount of overridden code in the semi-supervised K-Means clustering subclass?

3. ***(40 points)*** Wrapping up our suite of ML algorithms for this semester, we will implement Principal Components Analysis (PCA) to compress the representation of datasets. This one is not efficient to run with loops, and we must use matrix computations that will compute things over multiple entries with a single command[8]. Continuing with your matrix-based programming language (Matlab, Octave, Julia, R, etc.) or package (Python's numpy and scipy, C++'s Boost, etc.) of choice, grab the general classes you prepared from Homework 2 as your starting point. It is highly encouraged you TEST that your code works as intended after each step so that any problems in the next step are easier to diagnose. *Good programming practices you should use in this assignment are listed in the back as an appendix.*

   TO-DO: Implement a PCA class as specified below in the other TO-DO segments. When following the good programming practices, remember that you should include comments that document what parts of the code do. Because the TO-DO segments only describe what code to write, use the context provided before the TO-DO segments to determine what comments to provide that explain what the code does.

   Train Function

   (a) The inputs to the train function should always be the hyperparameters (as an array or hash map based on your implementation in Homework 2) and the training data as inputs X (a matrix) and ground truth outputs $\overrightarrow{y}$ (a vector, which is effectively a matrix with exactly one column). Like with the unsupervised version of K-Means clustering, $\overrightarrow{y}$ is still a function input even though unsupervised learning does not provide training data labels.

[7] Following good programming practices, we should define some constant/variable assigned to a special number that encodes "no label exists." You can use this variable in your code to check for unlabeled samples/rows, and people using your software can use it in their setup of $\overrightarrow{y}$ to guarantee consistency. Imagine what could go wrong if their "no label exists" number is different from yours... For future-proofing purposes, you should put this constant/variable in the general machine learning superclass to make this available and consistent across all your future semi-supervised learning classifiers.

[8] This comes down to the matrix operations being very well programmed and optimized beyond anything we could do in this course. Think of it like Homework 1, but on steroids.

TO-DO: Implement the PCA class as a subclass of the generic ML algorithm class, and setup the train function that the subclass will overwrite. For now, this overwrite is an empty function with only a name, the parameter list, and any additional syntax your programming language of choice requires to write a function (examples are curly braces, colons, keywords like `pass` or `return`, etc.).

(b) Let the hyperparameter the user sets be the number of principal components to learn K. Check this hyperparameter to confirm that it is an appropriate value.

TO-DO: Implement an if-then-else check within the train function that checks for the value assigned to the hyperparameter for the number of principal components. It should be both positive and less than or equal to the smaller of the number of samples/rows or features/columns of X.

(c) We must ***standardize*** the dataset before computing anything due to the spatial transformations being scale-dependent. That is, one feature being measured in miles and another being measured in degrees Celcius will cause each of their feature values to transform differently and distort the information. Converting X into the standardized dataset $X_{standardized}$ avoids this because all the features convert their unit of measurement to ***z-score***, how many standard deviations a value is from the mean assuming all the data follows a normal distribution. With a uniform unit of measurement, there is no risk of distortion because the transformations in each feature have identical impact. We will need to find the mean and standard deviation per feature in the dataset X in order to compute the standardized version—do this without using any loops.

TO-DO: After checking the hyperparameters, compute and store both the mean and standard deviation of the dataset X per column, which your matrix-based programming languages can compute to generate each vector in only one function call (no loops!). Some of these functions use the parameter name `axis` for choosing rows vs. columns. Next, define a matrix variable that will store the standardized dataset. In this case, we will create a Data Standardization Function later that we can use elsewhere (including other machine learning algorithms from past homework assignments), and it will take X, the vector of means per feature, and the vector of standard deviations per feature as input while returning $X_{standardized}$ as output. For now, this function should just print something unique like "Performing Data Standardization: (Inputs include...)" and return a dummy output that has the correct format (a matrix with the same number of rows and columns as X).

(d) With the standardized dataset, we will take advantage of there being many efficient algorithms for approximating the singular value decomposition (SVD) of any matrix to get

$$X_{standardized} \approx U\Sigma V^T.$$

This provides the information we need for PCA because (1) one of the U or V^T matrices will have the eigenvectors of the covariance matrix and (2) Σ's singular value entries correspond to the eigenvalues of the covariance matrix. Better yet, most matrix-based programming languages will compute the SVD and already order the eigenvectors to correspond to the eigenvalues from greatest-to-least!

TO-DO: Compute and store the SVD of $X_{standardized}$ as the U, Σ, and V^T matrices. Depending on your language of choice, this single SVD function might return a tuple/list/collection of values corresponding to each of these three matrices or some special data structure containing the three matrices as properties. Be sure to read your API documentation to find this out.

(e) Determine which of U or V^T has the eigenvectors you need. Specifically, there will be as many eigenvectors as there are columns in $X_{standardized}$. Return this matrix so that *each row is a normalized eigenvector.*

TO-DO: Check the sizes and U and V^T to determine which one has the same number of rows and columns (they are square matrices) as the number of columns of $X_{standardized}$. If U is the correct matrix, then transpose it because U has the eigenvectors as columns by definition (V^T is fine because its rows are the eigenvectors). Alternatively, you can directly choose the correct matrix without performing this check because Σ has the same number of rows and columns as $X_{standardized}$, and you can figure out the number of rows and columns for U and V^T mathematically (see the question at the end of this paragraph). Assign the desired matrix to a variable noting which one has the eigenvalues. Answer the following question: Based on how matrix multiplication works regarding the number of rows and columns for the multiplicand and product matrices, which of U or V^T should be the one with as many rows and columns as the number of columns of $X_{standardized}$?

(f) Take the first K eigenvectors out of the selected matrix (recall that the principal components happen to be the eigenvectors) and store them in the parameters as a matrix with each principal component being a row.

TO-DO: For whichever of U^T or V^T that you stored in the previous step, store a copy of that matrix's first K rows into the class's parameters property.

Predict Function

(a) The input to the predict function should always be the inputs X.

TO-DO: Setup the predict function that the subclass will overwrite. For now, this overwrite is an empty function with only a name, the parameter list, and any additional syntax your programming language of choice requires to write a function (examples are curly braces, colons, keywords like `pass` or `return`, etc.).

(b) An important sanity check, if you did not implement this for the generalized template, is that the parameters have been learned already. If not, be sure to inform the user and terminate the program gracefully (throw an error/exception or trigger an assertion violation).

TO-DO: At the start of the predict function, check whether the class instance's value for the parameters is set to a non-default value (typically null). If it is the default value, then throw an error/exception or trigger an assertion violation that provides a message about needing to train the model first.

(c) We must standardize the inputs like in the train function before computing anything due to the spatial transformations being scale-dependent. We will need to find the mean and standard deviation per feature of X in order to compute the standardized version—again, do this without using any loops.

TO-DO: Like in the train function, compute and store both the mean and standard deviation of the dataset X per column, which your matrix-based programming languages can compute to generate each vector in only one function call (no loops!). Some of these functions use the parameter name `axis` for choosing rows vs. columns. Next, define a matrix variable that will store the standardized dataset. In this case, we will call the same Data Standardization Function described in the train function (we will define it after the predict function is finished) that takes X, the vector of means per feature, and the vector of standard deviations per feature as input while returning $X_{standardized}$ as output. Recall that, for now, this function should just print something unique like "Performing Data Standardization: (Inputs include...)" and return a dummy output that has the correct format (a matrix with the same number of rows and columns as X).

(d) Perform the data compression and decompression, one after the other. This is not quite a prediction as we have done with other ML algorithms' predict functions, but it does perform the learned function (defined by the parameters after training) on the input data. This is

why some machine learning software libraries call this function `transform` instead, but they still have a predict function due to the superclass definition[9].

TO-DO: Multiply the learned parameter matrix by the transpose of $X_{standardized}$ and store the product—this is the compressed data $X_{compressed,standardized}$. Then, multiply the transpose of the learned parameter matrix by the compressed data $X_{compressed,standardized}$—this is the decompressed data $X_{decompressed,standardized}$. Make sure $X_{decompressed,standardized}$ has the same number of rows and columns as $X_{standardized}$, and transpose the matrix if that is not the case (the question at the end of this paragraph shows how we can figure this out mathematically rather than via code). Answer the following question: Based on how matrix multiplication works regarding the number of rows and columns for the multiplicand and product matrices, do we need to transpose $X_{decompressed,standardized}$ after computing it so that it has the same number of rows and columns as $X_{standardized}$?

(e) Because the compressed data should use the same units as the original data to properly represent it, we must undo the standardization of $X_{decompressed,standardized}$ to get $X_{decompressed}$ and then return it.

TO-DO: We will create a Data Undo Standardization Function later that we can use elsewhere (including other machine learning algorithms from past homework assignments), and it will take $X_{decompressed,standardized}$, the vector of means per feature that was used earlier in the predict function for the Data Standardization Function input, and the vector of standard deviations per feature (also used earlier) as input while returning $X_{decompressed}$ as output. For now, this function should just print something unique like "Performing Data Undo Standardization: (Inputs include...)" and return a dummy output that has the correct format (a matrix with the same number of rows and columns as X). Rather than storing the output from calling the Data Undo Standardization Function, we can return it directly.

Data Standardization Function

(a) This function requires the dataset X, the mean vector $\overrightarrow{\mu}$, and the standard deviation vector $\overrightarrow{\sigma}$. These should all be inputs to the function.

TO-DO: Setup the data standarization function within the PCA subclass, or set it up in the ML Algorithm superclass if you want this feature available for your other ML Algorithms. For now, this is an empty function with only a name, the parameter list, and any additional syntax your programming language of choice requires to write a function (examples are curly braces, colons, keywords like `pass` or `return`, etc.).

(b) First, each datapoint in X needs to be *centered in the coordinate space* by changing the mean of each feature to 0. Conveniently, subtracting the corresponding mean per feature of the datapoint can do this: $\overrightarrow{x}_n - \overrightarrow{\mu}$ for $n \in \{1, 2, \ldots, N\}$. However, the number of samples N can be large enough that looping could take too long. *YOU SHOULD NOT USE ANY LOOPS.*

TO-DO: We will take advantage of a process called ***tiling*** or ***repeating*** (depends on your chosen programming language) that will construct a matrix by appending copies of another matrix or vector. Thus, you can tile/repeat $\overrightarrow{\mu}$ as many times as the number of rows in X to get a matrix μ_{tiled} with the same number of rows and columns as X. Simply subtract that tiled matrix μ_{tiled} from X and store the difference in a matrix variable $X_{centered}$.

(c) Second (and lastly), each datapoint in $X_{centered}$ needs to be scaled so that all the features have the same unit length. This matters because different types of data can have different ranges, such as height (between 0 cm and 175 cm) vs. age (between 0 years and 100 years). Conveniently, dividing data that is centered at 0 (a.k.a. the previous step) by the data's standard deviation performs this scaling so that all features are now represented by their

[9]The exception to this is when software libraries use additional superclass options that are separate from other ML algorithms, such as dimensionality reduction algorithms.

statistical z-score, which is the same unit of measurement across all features. However, we are not going to divide entry-by-entry via $X_{centered:n,f}/\sigma_f$ because that looping will also take too long. *YOU SHOULD NOT USE ANY LOOPS*. The solution to this one is to remember that division is also multiplication as long as the denominator is not 0 (a standard deviation of 0 would mean that feature's values are the same for all the samples/rows) to get $X_{centered:n,f} \cdot (1/\sigma_f)$. *BE CAREFUL*: this is not a scalar multiplication times a matrix because there are different scalars (each entry in vector $\overrightarrow{\sigma}$) based on the data's column. Tiling/repeating also will not work because matrix multiplication involves both addition and multiplication when we want just multiplication. However, *diagonal matrices can accomplish scalar multiplication with a different scalar per vector*; having 0 along a row/column except for the diagonal means that only one row/column of the product matrix will be multiplied by the scalar in that diagonal entry!

TO-DO: Perform element-wise power (the matrix data structure or class in your programming language of choice should have a function that does this) to compute $\overrightarrow{\sigma}_{inverted}$ where each entry is raised to the -1 power (putting each standard deviation in the denominator), and store this vector in a new variable. Then, define a variable $\sigma_{inverted,diagonal}$ that stores a matrix with each entry of $\overrightarrow{\sigma}_{inverted}$ along its diagonal (your matrix-based programming languages should also have a function for creating a diagonal matrix from a vector); this should be a square matrix with as many rows and columns as the number of columns in X (the number of features). Multiply $X_{centered}$ by $\sigma_{inverted,diagonal}$ such that their rows and columns line up, and return this matrix product $X_{standardized}$.

Data Undo Standardization Function

(a) This function requires the standardized dataset $X_{standardized}$, the original dataset's (X's) mean vector $\overrightarrow{\mu}$, and the original dataset's standard deviation vector $\overrightarrow{\sigma}$. These should all be inputs to the function. We are effectively undoing the Data Standardization Function, which means many of these steps should look familiar (except in reverse).

TO-DO: Setup the data undo standarization function within the PCA subclass, or set it up in the ML Algorithm superclass if you want this feature available for your other ML Algorithms[10]. For now, this is an empty function with only a name, the parameter list, and any additional syntax your programming language of choice requires to write a function (examples are curly braces, colons, keywords like `pass` or `return`, etc.).

(b) First, each sample in $X_{standardized}$ needs to have its scale re-applied so that all the features have their original unit length. This matters because the z-scores are less readable to people who are used to the typical ranges for different types of data. Conveniently, multiplying z-scores by the original data's standard deviation re-applies this scaling to each feature. Like with the data standarization function, we are not going to multiply entry-by-entry via $X_{standardized:n,f} \cdot \sigma_f$ because that looping will take too long. *YOU SHOULD NOT USE ANY LOOPS*. Again, this is not a scalar multiplication times a matrix because there are different scalars (each entry in vector $\overrightarrow{\sigma}$) based on the data's column—we will use diagonal matrices.

TO-DO: Create a diagonal matrix with each entry of $\overrightarrow{\sigma}$ along its diagonal (your matrix-based programming languages have a function for this) and store it in a variable $\sigma_{diagonal}$. Then, multiply $X_{standardized}$ by $\sigma_{diagonal}$ such that their rows and columns line up, and store this solution in a variable X_{scaled}.

(c) Second (and lastly), each sample in X_{scaled} needs to be *repositioned in the coordinate space* by returning the center to the original mean of each feature instead of 0. Conveniently, adding the corresponding mean per feature of the datapoint can do this: $\overrightarrow{x}_{scaled:n} + \overrightarrow{\mu}$ for $n \in \{1, 2, \ldots, N\}$. However, the number of samples N can be large enough that looping could take too long. *YOU SHOULD NOT USE ANY LOOPS*. We will again take advantage of tiling/repeating to construct a matrix by appending copies of another matrix or vector.

[10]The choice should be consistent with where you chose to put the Data Standardization Function

TO-DO: Like in the data standardization function, you can tile/repeat $\overrightarrow{\mu}$ as many times as the number of rows of X_{scaled} to get a matrix μ_{tiled} with the same number of rows and columns as X_{scaled}. Store μ_{tiled} and return the sum of μ_{tiled} and X_{scaled}.

4. ***(10 points)*** Flubsy decided not to try anything slick this time. "If I keep my mouth shut, then everyone can focus on the assignment and final project. Nothing can possibly go wrong!" For once, Flubsy is right! Nothing complicated to do here, just tell Flubsy goodbye.

   TO-DO: Write a brief message, which can be as simple as two words, that sends a parting to Flubsy. Please, address Flubsy and not some other person; some students in the past have ignored Flubsy and hurt their feelings—don't be that person!

Appendix: Good Programming Practices

When writing code, you must follow these good programming practices:

- When turning in code, keep in mind that Rick Freedman (the instructor who wrote this assignment) has a golden rule: *Good code with bad comments is bad code, PERIOD.* Make sure comments are clear and describe what will happen per significant moment in the code. If the code does something trivial over several lines, explain the idea just before the first of those lines. If the code does something complicated on a single line, then explain what that line does. Use your judgement, but also remember you might need to reuse and understand this code later—your future self will thank you if you document it better now while the content is fresh in your mind!

- Documenting functions you create (purpose, inputs, outputs, and possible error throws) is also helpful. There is a reason most APIs that you read include these details as part of the manual.

- Avoid "magic numbers" when encoding choices like the linear regression approaches. It is tempting to use an integer such as 0 meaning MLE, etc., and the idea works well if done carefully. If you simply write 0 everywhere in the code, then how will you know which 0's mean the MLE choice, which 0's mean the number 0, and which 0's mean some other choice between different options? Instead, set a variable (ideally constant, if your language supports them, to avoid accidentally changing the assignment) to the number, and then use that variable everywhere in the code. It is easier to understand (compare `approach = 0` to `approach = MLE_LINREG`) and easier to update later if something changes in your code. New option to add in that slot, which means you have to change the previous option's assignment? Just change the variable to another number and do not worry because the variable effectively replaced it everywhere else in the code. Found a bug in the code that requires you to change the option you chose somewhere in the code? Now you know which 0's are for the option and which 0's are not so you do not accidentally change the wrong things (creating more errors that were not broken before).

- Be careful naming the output file to which you write because the program could overwrite another file that your program previously wrote by mistake! It is usually good practice to add a date+timestamp in the filename so that the output filenames are unique (unless you call the function really fast multiple times in a row).

- Store intermediate computations that you are able to reuse rather than compute everything from scratch in each line of code. Although you can compute things multiple times (such as X^T), it costs *real-world time* in an era where computer memory is much cheaper on your average computer (satellites and embedded systems are a different story, as is time spent retrieving cached data... these are all beyond the scope of this assignment). Save the time by storing the intermediate computation steps, and then call the variable storing it in every case that you use it afterwards. Storing the intermediate computation also lessens the chance of making a typo when recomputing it next time. This is similar to the benefits of avoiding "magic numbers."

- Break out sequences of computations that you will reuse frequently into functions rather than rewrite those lines of code from scratch each time you need it. Similar to storing intermediate computations, this lessens the chance of a typo when rewriting all that code each time. Furthermore, the function makes your code easier to read if it is descriptive—consider a function name replacing a lot of redundant code like an abridged comment placed above its block of code.

- Make debugging accessible even after your program is finished. You never know when you will need to look at the code or modify it again, and you do not want to have to redo your entire debugging setup. In particular, if you have any print statements in your code to explore the control flow ("entering function F," "exiting loop L in function F," etc.) or inspect variables ("the value of variable V is currently:," "compare variable V's value to expected E:," etc.), then those should stay in your code. However, to avoid those print statements cluttering the screen, put them all inside conditional blocks (if statements) where the condition to check is for a global constant variable such as DEBUG. Make sure to define this global constant variable DEBUG at the start of your code, and then set it appropriately before your compile and run your code. If DEBUG is boolean, then use TRUE when you want the debug information to print and FALSE when you do not want the debug information on the screen. If DEBUG is an integer, then define a hierarchy of verbosity/priorities where 0 means 'print nothing,' 1 means 'print the basic information,' 2 means 'print some additional details,' etc. (make sure to define each verbosity/priority level with its own constants to avoid "magic numbers"). You can also create more elaborate debugging variables in more complex code, such as separate DEBUG_X constant variables for different parts of the program X. For an assignment like this, one boolean variable should be sufficient.

Rick has lots of other rules and pet peeves about writing good code, but you can talk about those at office hours sometime if you are interested. They do not likely apply in this assignment.