Answers for HW4

Question 1

(a)

Train Function ***i***: Why does the class hierarchy we designed require us to include $\vec{y}$ as a function parameter for training? What are some things users can do to avoid this being a problem when call the K-Mean cluster train function? If the training function never uses $\vec{y}$ in any of its lines of code, then why does it ultimately not matter what the user does?

- The class hierarchy is designed to generalize training functions across multiple models, including supervised and unsupervised learning. To maintain a consistent API, the training function might always require $\vec{y}$, even if it is not used.
- To avoid this being a problem, the user can pass in a dummy variable as $\vec{y}$.
- This ultimately does not matter because K-mean does not utilize $\vec{y}$ in in calculations.

---

iii: 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?

- The variable that represents the number of samples in each cluster (let's call it cluster_counts) is updated through accumulation during the clustering process. If we do not initialize this at 0, it can lead to an incorrect result as the program runs, depending on the programming language used, it can also crash the program.

- The parameter matrix (e.g., the centroids in K-Means) is typically replaced rather than accumulated during updates. Because of this, we do not have to care about what value it is initialized with.
- Variables that are updated through accumulation must be initialized to values that appropriately reflect the absence of prior updates. While variables that are replaced during updates do not require careful initialization since their values are overwritten before being used in calculations.

---

v: Why do we need to divide 1.0, rather than 1, by the number of current samples/rows assigned to a specific cluster?

- Depending on which programming language was used, if we use 1 (an int type value) instead of a 1.0 (a float type value) to divide the number of samples, the resulting value would be an int instead of a float. This can cause many problems, mainly precision problem, which could ruin our centroids calculations.

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?

- If the starting point is the largest number possible anything after it is guarantee to be smaller than that number.

---

vi: 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?

- Convergence is a state that can be disproven by a single counterexample. Starting with true and disproving convergence as needed reflects this one-directional nature of the check.
- This is similar to lazy boolean evaluation in that: *Conjunction*, If one condition is false, the result is immediately false, skipping the rest. Similarly, in convergence checks, finding non-convergence flips the state to false, avoiding unnecessary checks. *Disjunctions*, If one condition is true, the result is immediately true, skipping the rest. Similarly, convergence checks start with true and avoid repeatedly flipping back unless disproven.
- This save times because it allow the program to exit a condition check earlier. There also minimal state change, only flipping True to False or False to True when strictly necessary.

Predict Function ***iii***: 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 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?

- We can definitely parallelize this function. This is because each prediction is independent from each other, we can run each prediction on a different thread and combine it back together at the end.
- The prediction function share some similarity with the train function, in that they both perform distance calculations between each data points and the centroids. They both assign a centroid to each data point.
- To reduce code duplication, we can abstract distance calculations, create a functions for centroids assignments which can be access by both functions.

Question 2

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?

- To avoid overriding code in the semi-supervised K-Means clustering subclass, we can modularize components of the train function from unsupervised K-means clustering into helper functions. These helper can handle distinct, reusable parts of the training process, making it easier to override only the necessary parts.

Question 3

Train Function

c: 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 $X_{standardized}$?

- V^T should have as many rows and columns as the number of columns of $X_{standardized}$ because it should be in the same feature space, aligning with the features of $X_{standardized}$.

---

d: 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 $X_{standardized}$?

- We do not need to transpose $X_{decompressed,standardized}$ after computing it. This is because its dimension are inherently the same as $X_{standardized}$ due to how matrix multiplication works in SVD reconstruction.