Lab 6

Due: Friday 10/20/2023 @ 11:59pm EST

The purpose of labs is to give you some hands on experience programming the things we've talked about in lecture. This lab will focus on detecting outliers with SVD.

Task 0: Setup

Included with this file is a new file called `requirements.txt`. This file is rather special in Python: it is the convention used to communicate dependencies that your code depends on. For instance, if you open this file, you will see entries for `numpy` and `scikit-learn`: two pythn packages we will need in order to run the code in this lab. You can download and install these python packages through `pip`, which is python's package manager. While there are many ways of invoking `pip`, I like to do the following:

```
python3 -m pip install -r requirements.txt
```

(I let `python3` figure out which `pip` is attached to it rather than the more common usage: `pip install -r requirements.txt`)

Task 1: Finding a value of k (10 points)

In the file, at the top is a global variable called `K`. This variable should be set to the number of new features from the SVD of our data X that we want to keep. The way we figure this out in this lab is to run this file. A progress bar should populate and then a plot should appear. The y-axis of this plot shows the percentage of information retained (between a reconstruction of our data and the original data). The x-axis of this plot shows how many new features from the SVD of X were used to reconstruct X.

The way we read this plot is to find a point on the line. That point's x-coordinate x says that, if I use the top x best new features from the SVD of X to reconstruct X, then the reconstruction contains y (the y-coordinate) percentage of the info that X contains. Higher numbers are better, but in this plot we would need to keep most of the features in order to preserve 100% of the data.

The way we choose a value of `K` is to look for the inflection point on this graph (I see the primary inflection point around $x = 9$ or $x = 10$). Inflection points in graphs such as this determine the point it stops becoming "worth it" to keep more features. Anything before the inflection point is worth it, and anything afterwards is not (because of diminishing returns). Set your value of `K` using this methodology.

Task 2: Finding a value of `ZSCORE_STDDEV_THRESHOLD` (10 points)

The next constant to set is how far away a z-score value has to be away from the mean (i.e. 0) to be considered anomalous. We will be somehow converting points into z-scores and then thresholding those z-scores to determine who and who is not an outlier. Remember your background knowledge from statistics: a point is (rare) if it is more than 1 stddev away, extra rare if it is more than 2 stddevs away and extremely rare if it is more than 3 stddevs away. Set your value of `ZSCORE_STDDEV_THRESHOLD` using this methodology.

Task 3: `def reconstruct_X_using_k_features` (20 points)

This function should calculate the SVD of X and use the SVD of X to reconstruct X using only the best k features from the SVD. However, unlike the 99% use case of SVD where we want to create a matrix with k columns, this time we want to express the reconstruction in the original feature space (just using only the information contained within the top k new features). Here is how we do this:

- Calculate $U, \Sigma, V^T = \texttt{svd}(X)$
- Set all singular values outside of the top k to be zero.
- Reconstruct $\hat{X} = U\Sigma V^T$

The subtlety is in step 2. Normally we would truncate U and Σ by throwing away parts of the matrices that we don't want to keep. Here however we not truncating, but instead zeroing out singular values. Whenever we set a singular value to zero, we are effectively throwing it away when we do the matrix multiplication (i.e. those new features will have a coefficient of 0 in the matmul and will not contribute anything). This process however produces a matrix that has the same dimensionality as the original dataset, albeit containing less information than the original dataset.

Please implement this functionality into `reconstruct_X_using_k_features`

Task 4: `def calculate_distances` (20 points)

Given X and a reconstruction of X called $\hat{X}$ in the same feature space, we want to measure how closely each point corresponds to its reconstruction. This function should measure the distance between a point and its reconstruction using euclidean distance and return a numpy array containing these distances. The ordering of these distances should align with the ordering of the points themselves (i.e your output should be $[d(x_1, \hat{x}_1), d(x_2, \hat{x}_2, \cdots, d(x_n, \hat{x}_n)])$

Task 5: `def z_score` (20 points)

This function should take in a numpy array of distances and convert these distances into z-scores. Remember, the way we calculate a z-score is with the following formula:

$$z_i = \frac{x_i - \texttt{mean}(x)}{\texttt{stddev}(x)}$$

Your output should also be a numpy array

Task 6: `def get_outlier_idxs` (20 points)

Given a numpy array of z-scores, we want to figure out which of them are anomalous values. Using the value you set for `ZSCORE_STDDEV_THRESHOLD` earlier, figure out which indices in this array contain values that are anomalous (according to your threshold). Your output should also be a numpy array.

Note: anomalies can occur on either side of the mean!

Task 7: Submit Your Lab

To complete your lab, please **only turn in the `outliers.py` file** on Gradescope. You shouldn't have to worry about zipping it up or anything, just drag and drop it in.