AMATH 482: HOMEWORK 1

SOPHIA BARBER

Department of Applied Mathematics, University of Washington, Seattle, WA
sophiab3@uw.edu

Abstract. Here, we study a robot which records the movements of 38 of its joints. It performs 3 movements: running, jumping, and walking. Our goal is to construct a classifier that takes a sample of movement and identifies which movement was performed. We do this by using Principal Component Analysis (PCA) to project our data onto a lower dimension, and classify each sample by assigning it to the class of movement whose centroid is closest to its position, in low-dimensional space. We will adjust the dimensionality of the problem and explore how many dimensions are necessary in our low-dimensional approximation.

1. Introduction and Overview

We consider a humanoid robot which can record the movements of 38 of its joints. The robot can perform 3 movements: running, jumping, and walking. We have a set of training data which contains 15 samples: 5 samples of each of the 3 movement types. Each sample records the (x, y, z) coordinates in all 38 joints for 100 time steps. This means that each sample is a 114 x 100 matrix, where each column (which contains 38*3 = 114 elements) records the spatial locations of the 38 joints at that time step. Our goal is to project the data onto a lower dimension (the original data has 114 spatial dimensions) and, based on this, design an algorithm that can take in any measurement of movement and identify whether the robot was running, jumping, or walking.

In order to do this, we will treat each time step as 1 sample (this means there will be 1500 "samples" in our training data). We will then create a vector of ground truth labels with an integer per class (0 = walking, 1 = jumping, 2 = running) where the first element of the vector is the label for our first "sample", etc. Then we will truncate the PC modes to k modes (see Section 2) and project our training data into k-modes PCA space. We will compute the centroid (mean) in k-modes PCA space of each movement (average element wise over all samples in k-modes PCA space that we know to be from a movement, resulting in a k x 1 vector; repeat for all 3 movements). Then we look at the projected point in k-modes PCA space for each sample. We take the distance between this point and each of the 3 centroids. We assign an integer (see above) to that sample that corresponds to the movement class whose centroid the point was closest to. Then we can compute the percent accuracy of the new, trained labels with the ground truth labels, to see how well our classification model fits our training data.

Finally, we will test how our classification model performs on classifying new samples, by loading our test data (a much smaller, new data set) and apply our classification model to it. For all of these tasks, we will compare what happens with different numbers of PCA modes (i.e. varying k) and discuss what k is best, i.e. how many spatial PCA modes are necessary to retain various degrees of classification accuracy.

2. Theoretical Background

Singular Value Decomposition (SVD) and Principal Component Analysis (PCA):

To understand PCA we must first understand SVD. Performing SVD on a matrix $A \in \mathbb{R}^{n\times m}$ is computing a factorization of the matrix A: there exists orthogonal matrices $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{m\times m}$ and diagonal matrix $\Sigma \in \mathbb{R}^{n\times m}$ with positive entries for which $A = U\Sigma V^T$. The matrix U contains the left singular vectors of A as its columns, the matrix V^T contains the right singular vectors of A as its columns, and the matrix Σ has the singular values of A as its diagonal entries, and 0 elsewhere, with the largest singular value σ_1 in the (1,1) entry of Σ, and with $\sigma_2, ...$ placed in decreasing order as you go down the diagonal of Σ [1].

Now that we have performed SVD on the matrix A, which will contain our data, we will use it to perform PCA. To do PCA, we must center and scale our data, i.e. subtract the mean and divide by the variance, before we compute the SVD above. Assume that A was already pre-processed as such before we computed

Date: February 14, 2025.

the SVD. The goal of PCA is to decompose the data into directions that maximize variance, and keep only so many of these directions. We want to build a low-rank projection of our data onto the directions of maximum variance that we kept [1]. Our left singular vectors (columns of U) are our PC modes, which we computed with SVD. These are the directions of most to least variance, with the first singular vector/PC mode being the direction of most variance, and the last singular vector/PC mode being the direction of least variance. Now we want to project our data, i.e. our matrix A, onto k of these vectors, i.e. into k-PCA space. Here, k is the number of singular vectors/PC modes we keep. This means that we are not keeping all of the information, only the information in the k directions of most variance, i.e. in the k directions that capture the most information. In effect what we are doing is truncating the SVD. From [1], if we write the SVD as

$$A = \sum_{i=1}^{min(m,n)} \sigma_i \vec{u_i} \vec{v_i}^T \tag{1}$$

where $\vec{u_i}$ is the i^{th} column of U and $\vec{v_i}$ is the i^{th} column of V^T, then the projection of A into k-modes PCA space (where $k \leq min(m,n)$) is the truncated SVD

$$A_k = \sum_{i=1}^{k} \sigma_i \vec{u_i} \vec{v_i}^T$$

which is effectively equation 1, but where $\sigma_{k+1}, ..., \sigma_{min(m,n)} = 0$.

Measuring Retained Energy:

If we construct a low-rank approximation of A, we would like to be able to have a way to measure how much information from the original data we have retained. We do this by measuring energy (where "energy" is how much variance/information is captured) via the Frobenius norm. In effect, by measuring how much energy is retained, we are measuring how much variance is captured when we only project onto the first k directions of most variance (i.e. onto the first k spatial PC modes). Keeping more PC modes means that more energy will be captured. So as we increase k, we capture more variance, and more variance means more information/energy. From [1], we can calculate the percent of energy retained when we project our data matrix A onto k-modes PCA space and construct a low rank approximation by

$$\frac{||A_k||_F}{||A||_F}$$

where A_k is our low rank approximation and $||\cdot||_F$ is the Frobenius norm, i.e.

$$||A||_F = (\sum_{i=1}^{min(m,n)} \sigma_i^2)^{1/2} \quad \& \quad ||A_k||_F = (\sum_{i=1}^{k} \sigma_i^2)^{1/2}$$

Thus we can see how many PC modes are necessary to keep a desired amount of information/energy when we build our low rank approximation. We use the Frobenius norm to do this because is sums the squares of the first k singular values, thus giving an accurate measure of energy.

Projecting data into k-modes PCA space:

When we project our data into k-modes PCA space (i.e. build a projection of our data onto the first k PCA modes), as discussed above, we can visualize our data by keeping only 2 or 3 PCA modes. In the case where k is 2, we can take the coefficients of the projection of our data into 2-mode PCA space, and plot them as (PC1, PC2) coordinates. This allows us to see trends in the data, and visualize our data set [1]. In this particular case (as seen in 4) we can see that different groups of data live in different places in 2d PCA space. PCA coefficients corresponding to the running samples are grouped together, and likewise for the other two movements. This indicates that our method of classification by proximity to centroids, as explained below, will be good because our data is well separated.

Building Classification Method/Machine Learning:

Our goal is to build a classification system that can take in samples of movements performed by the robot and identify them. To do this we use supervised learning. We have two data sets: a training set and a testing set. After pre-processing our training set (explained in Section 3), we want to build a model that will compute (1) the first k PCA modes and project our data onto them, and (2), the centroids for each movement (explained in Section 3), and use this to give us a set of *trained* labels that classify our samples.

We assign to each sample in our training set a ground truth integer corresponding to its type of movement. We then use our model to classify each sample in our testing set according to proximity to centroids (see Section 3). To do this, we consider the projected point of each sample of data in k-modes PCA space and compute the distance between this projected point and each of the centroids. We assign that sample to the class of movement whose centroid its projection was closest to. In principle, since our data is well separated, if a sample is close to the centroid of a particular movement, then we expect it to satisfy the same class of movement. We then measure the accuracy of applying this method to our training data to see how well our model fits the data. However, we also have a testing set, a smaller data set which the model has not "seen" before. We process our test set in the same way we did the training set, and then apply the same model to our test data. Notice that our model is "fitted" to the training data (see Section 3). This is so that we avoid fitting our model to any noise that we might have in our test set, allowing us to measure how well our model generalizes to new data. Applying our model to the test set lets us measure how well our model performs on recognition of new samples.

3. Algorithm Implementation and Development

Pre-processing: When we load our training data, we have 15 samples which are each 114 x 100, where the rows are the spatial dimension and the columns are the time dimension. However, we want to apply PCA such that the PC modes are spatial modes and the coefficients of the projection of our data onto the PC modes are the time-dependent coefficients. This means that we want each time slice to have its own PCA coefficients, and we want the modes (i.e. singular vectors) to be spatial. So, we should transpose each sample so that the columns are the spatial dimension and the rows are the time dimension. Thus when we compile our samples into `X_train`, we get a 1500 x 114 matrix. Furthermore, we will center and scale `X_train` using `StandardScaler` from `sklearn [4]`, which subtracts the mean and divides by the variance.

Task I: Now that we've loaded and processed our training data, we want to apply PCA. For each $k \in \{1, 2, ...114\}$, we'll get our first k PCA spatial modes, project our data `X_train` onto these k PCA modes, and then get a k-dimensional approximation of `X_train`. We do this by using the `PCA` library in python's `sklearn` [4]. We initialize an instance of PCA with k modes by `pca = PCA(k)`. Then we use `pca.fit_transform(X_train)` to project our data into k-modes PCA space, and then we get our low-dimensional approximation by using `pca.inverse_transform` [4].

For each k, we can measure what % of information/energy we retained in our k-dimensional approximation of `X_train` by finding the ratio of the Frobenius norm of our k-d approximation, to the Frobenius norm of `X_train`, as shown in Section 2, using `np.linalg.norm` [2]. This could also be considered the % accuracy of our low-dimensional approximation to `X_train`. After doing this for all k, we can see how many PCA modes are necessary to retain various thresholds of energy in our low-dimensional approximations of `X_train`.

Task II: To visualize our data, we will truncate the PCA modes to 2 and 3 modes and plot the projected `X_train` in 2- and 3- modes PCA space. To do this, initialize PCA as above with `k=2`. Then we use `pca.fit_transform(X_train)` [4] which gives us the projection of our data in 2-mode PCA space; i.e. it gives us the coefficients of this projection. This will be a 1500 x 2 matrix, since each sample will have 2 coefficients for its projection into 2-mode PCA space. We can plot these as (PC1, PC2) points [3], where column 1 of our matrix is PC1 and column 2 is PC2. We repeat with `k=3` and plot as (PC1, PC2, PC3) points [3].

Task III: To establish a ground truth classification of our 1500 "samples", we assign an integer to each sample based on its movement class, which is known to us: 0 = walking, 1 = jumping, and 2 = running. For some k, we project `X_train` into k-modes PCA space (as in task II). Then, for each movement, we compute the centroid/mean in k-modes PCA space. We do this by taking the element-wise mean over all the projections into k-modes PCA space of the samples corresponding to that movement, each of which will be a 1 x k vector. This gives us the centroid for that movement, which is also a 1 x k vector. We do this by `np.mean(data, axis=0)` [2], where `"data"` is the matrix whose rows are all of the 1 x k projections of samples of that movement into k-modes PCA space. We do this process for all movement types, resulting in one centroid/mean per movement: `j_mean, r_mean`, and `w_mean`.

Task IV: Now we want to build a model that will create a vector of *trained* labels that classify our data. We do this by first projecting `X_train` into k-modes PCA space, as explained above, and then take each row of this projection, which is the projected point in k-modes PCA space of the sample that lives in the corresponding row of `X_train`, and compute its distance to each of the 3 centroids. We do this using

`np.linalg.norm(sample_pca - centroid)` [2] where `sample_pca` is the projected point of the sample in k-modes PCA space, and for each sample, we do this for all three different centroids (`j_mean, r_mean`, and `w_mean`). To each sample, we assign an appropriate integer corresponding to the movement class whose centroid was closest to the projected sample in k-modes PCA space, producing our trained labels. We can measure the percent accuracy between the trained labels and the ground truth labels using `accuracy_score` from `sklearn` [4]. I repeat this process for all $k \in \{1, 2, ..., 114\}$ so I can compare accuracy with different k.

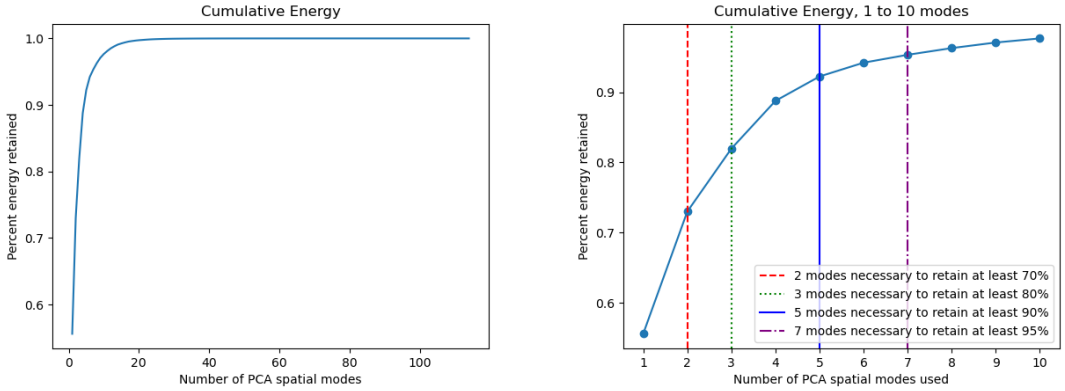
Task V: Now we want to test how well this model performs on recognition of new samples. We load our test data and pre-process it in the exact same way as above, except that we have fewer samples in our test data so our resulting matrix `X_test` will be 300 x 114 instead of 1500 x 114. Note that by "exact same" we mean that we center it by subtracting the mean of `X_train` and dividing by the variance of `X_train`. Then we project `X_test` into k-modes PCA space by initializing PCA as above, and then using the commands `pca.fit(X_train)` and then `pca.transform(X_test)` [4]. We already have our centroids for each movement in k-modes PCA space from above, and we repeat the same classification process for each sample in `X_test`. We fit `pca` to `X_train` and use the centroids from `X_train` so that our model will be fitted to our training data and we will thus avoid fitting to any noise in `X_test` and we can thus determine how well our model performs on recognition of *new* samples. Once we have got our trained labels for `X_test` using our model from Task IV, we can again use `accuracy_score` [4] to compute the accuracy of the model's classification of the `X_test` samples. Note: I repeated this entire process again, centering but *not* scaling both `X_train` and `X_test`. This means that I subtracted the mean of `X_train` from both but didn't divide by the variance. I compare these results to the ones from the above algorithm in Section 4.

Task VI: We use logistic regression to implement an alternative classifier. This is still a discrete classifier method like the centroid method above, but now we're using a logistic regression classifier rather than a "nearest-neighbor" classifier. We create a logistic regression classifier model using the `LogisticRegression` class from `linear_model` in `sklearn` [4]. We then fit our model to the training data by giving it the ground truth labels of our training data, and the projection of our training data `X_train` into k-modes PCA space. Then we give our model the projection of our test data, `X_test`, into k-modes PCA space (with `pca` fit to `X_train`, as in task V), and it uses the model to predict for us the classification of the samples in our test data. It does this by `logres.predict(Xk_test)` [4], where `logres` is the instance of our classifier model, and `Xk_test` is the projection of `X_test` into k-modes PCA space. We can then measure the accuracy of the classification (i.e. how well the model performs on recognition of new samples) by comparing the trained labels to the ground truth labels in the usual way. We can also find the training accuracy by giving our model the projection of `X_train` where we gave it the projection of `X_test`, above.

4. Computational Results

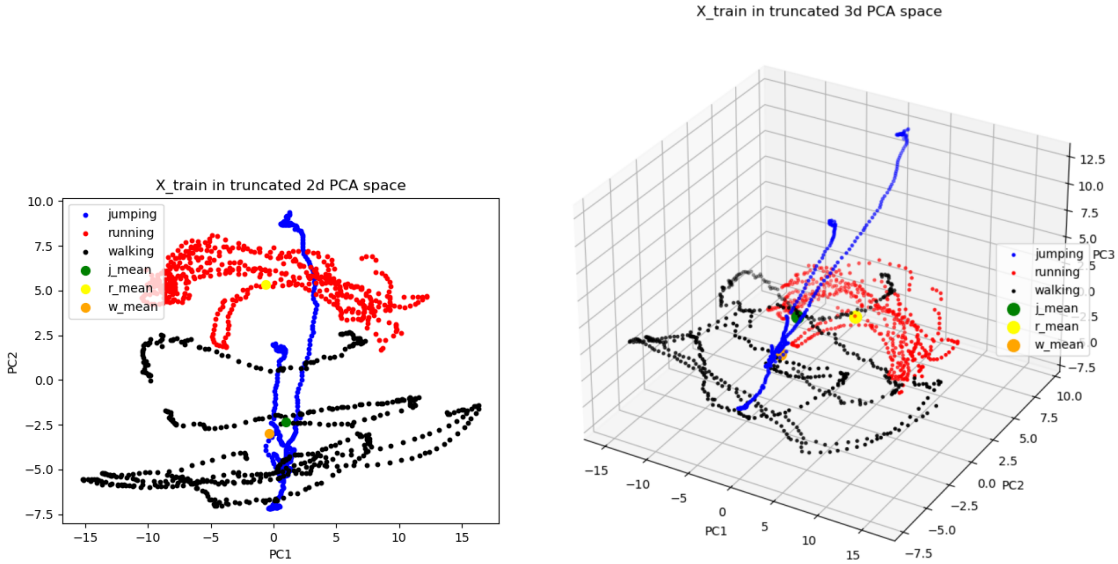
Task I: After applying the algorithm outlined in Section 3, we are able to see how well we approximate `X_train` with k spatial PCA modes. In Figure 1, we can see the cumulative energy vs k. In Figure 1a, we can see the percent of energy that is retained when we project `X_train` into k-modes PCA space and build a k-dimensional approximation of `X_train`, for all $k \in \{1, 2, ...114\}$. Figure 1b, zooms in on $k \in \{1, ..., 10\}$, and we can see the minimum number of PCA modes we need to keep to retain at least 70%, 80%, 90%, and 95% of the energy in `X_train`. We see that to retain 70% of the energy (i.e. approximate `X_train` by at least 70%), it is necessary to keep 2 spatial PCA modes; to retain 80%, we need 3 PCA modes; to retain 90%, we need 5 PCA modes; and to retain 95%, we need 7 PCA modes. As we expected, the more modes we keep, the more energy is captured because for higher k, we are keeping more directions which capture variance, and thus we expect to capture more energy/information when we build a higher-dimensional approximation of `X_train`, which means that `X_train` will be approximated to a greater accuracy with higher k.

Task II: As discussed in Section 2, when we truncate PCA space to 2 PCA modes, we can plot the projection of `X_train` onto the first 2 PC modes as 2d coordinates, (PC1, PC2), allowing us to visualize our data and see groupings. In Figure 2 below, we have plotted (PC1, PC2) in 2a, and the projection of `X_train` onto the first 3 PC modes as (PC1, PC2, PC3) in 2b. We can see that our data is in fact well separated, since the points corresponding to the samples of each movement are grouped together. This indicates that our method of classification via centroids, as explained in Sections 3, will work. Furthermore, we have plotted the centroids for each movement in 2- and 3- modes PCA space on each of the respective plots, where j_mean is the centroid for jumping, r_mean is the centroid for running, and w_mean is the centroid for walking. We can see that, as we would expect, the centroids are right in the middle of their respective data groups.



(A) (B)

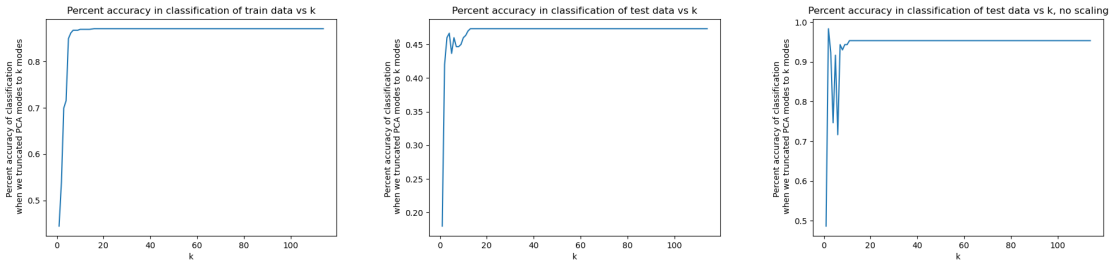
FIGURE 1. Cumulative Energy plots, showing the percent of energy that is retained when we approximate `X_train` with k PCA spatial modes. (A) shows the percent energy retained for all possible $k \in \{1, 2, ...114\}$. (B) shows this for only the first 10 k, and shows the minimum number of modes we need to retain various thresholds of energy. [3]



(A) (B)

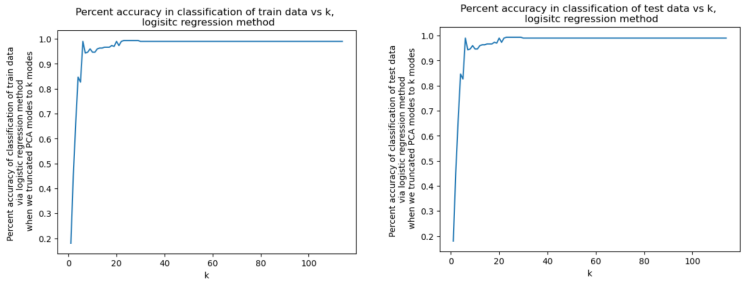
FIGURE 2. Plots of `X_train` projected into truncated PCA space with 2 PCA modes (A) and with 3 PCA modes (B), both including the centroids. This allows us to see that our data is well separated by movement type. [3]

Tasks III, IV, V: After we have used our centroid model to classify our movement samples in both the train data and the test data, and have measured the accuracy of both, for all possible k, we are able to determine what the optimal k is in terms of classifier accuracy for both the model fitting the data and for the model classifying new samples. Figure 3a shows the percent accuracy in classification of training data vs k. We see that the accuracy increases as we increase k, as we would expect, because we saw from the results of task I that adding more directions that capture variance means that more information is being captured, and our data is thus better approximated when we project it into k-modes PCA space and then build a low-dimensional approximation of the data. Since we have more information captured with higher k, we expect our classifier to be more accurate, and our model to fit the data better, which we see in the



(A) (B) (C)

FIGURE 3. Plots of accuracy in classification vs k. (A) shows accuracy in classification of train data, (B) shows accuracy in classification of test data, and (C) again shows accuracy in classification of test data, but where we did not scale our data before classifying. [3]



(A) (B)

FIGURE 4. Plots of accuracy in classification via logistic regression method vs k. (A) shows accuracy in classification of train data; (B) shows accuracy in classification of test data. [3]

trend of training accuracy. We see that the maximum accuracy is about 87%, and 6 modes is enough to get 85%, so we would say that $k = 6$ is optimal for classifier accuracy on training data. In Figure 3b, we see the accuracy in our classification of the test data vs k. We actually get rather poor accuracy: $\sim 47\%$ at most. We hypothesize that this is due to the fact that our model is not robust and very sensitive to scaling. We hypothesize that this means that all of our train samples had similar "length" in k-modes PCA space, and the samples in the test data had very different "length" from the train samples, thus resulting in low accuracy when we use our model to classify the test data. When we pre-processed our data sets before using the model, we subtracted the mean and divided by the variance. If we were to only subtract the mean (i.e. we center, but don't scale, our data) then we find that we get much better accuracy with our test set, but fairly similar results on our training set. This supports our hypothesis. Figure 3c shows the test accuracy vs k when we don't scale our data. We see that it is much better, surpassing 90% accuracy. In these plots, we see some oscillations in accuracy levels. This may be due to the fact that adding more information at some point might make the accuracy go down because that information might be not very correlated to the existing information, or it might be more noisy, giving the model more opportunities to make mistakes.

Task VI: After using our new, logistic regression model to classify our samples in both our train and test sets, for all k, we see that this model is much more robust than the one we built in task IV. We got almost virtually identical accuracy in the classification of both our train and test sets, and we were able to reach a higher level of accuracy for both sets. For the training set, we were able to get up to almost perfect accuracy (Figure 4a), whereas with the previous centroid method we could only get $\sim 87\%$ accuracy at best. In classification of the testing set, the logistic regression model vastly outperformed the centroid model in terms of classification accuracy, even when I scaled the data, meaning that this model is more robust and not as sensitive to scaling (to produce figures 4a and 4b, I scaled my data as well as centering it). We see this in Figure 4b: we get up to almost 100% accuracy in classifying test samples when using scaled data, rather than only $\sim 47\%$ accuracy from the centroid method. I conclude that this method is much better.

5. Summary and Conclusions

In this paper, we were able to build projections of our data into low-dimensional space, and see how many dimensions were necessary to retain a desired amount of the original information, or "energy". We were able to visualize how our data set was separated by movement type by projecting it into 2- and 3- modes PCA space. Then we implemented a classifier method using the centroid of each movement in k-modes PCA space, where we classified each sample by assigning it to the movement class whose centroid the projection of the sample in k-modes PCA space was closest to, out of the three centroids. We calculated our training and testing accuracy for all k, allowing us to see how well our model fit the data and how well it generalized to classification of new samples when we kept any number of PCA modes, prompting further analysis. We then implemented a new classification method based on logistic regression and compared it's accuracy to the centroid method, concluding that it was much more robust. In future, we could try some other classifier methods such as support vector machines or random forests.

Acknowledgements

The author is thankful to Dr. Frank for her lectures on SVD theory and other related topics. We are also thankful to Rohin Gilman for explaining how PCA works conceptually, as well as how to implement PCA in python. He was also helpful in understanding the machine learning concepts behind the classifier methods we implemented, and in answering other general questions.

References

[1] N. Frank. Amath 482 lecture notes. *Department of Applied Mathematics, University of Washington*, Jan/Feb 2025.
[2] C. R. Harris, K. J. Millman, S. J. van der Walt, et al. Array programming with NumPy. *Nature*, 585:357–362, 2020.
[3] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
[4] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.