AMATH 582: HOMEWORK 2

HUNTER LYBBERT

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

Abstract. In this analysis of the joint movement data of OptimuS-VD we use Principal Component Analysis (PCA) for dimensionality reduction. We visualize the movements in this new lower dimension. Using the lower dimensional projection we design an algorithm that recognizes which movement OptimuS-VD is performing. Furthermore, we first implemented a clustering-esque classifier and evaluate it's accuracy using the training and test sets.

1. Introduction and Overview

There are many computationally intensive computer vision models (primarily using convolutional neural-networks) that perform well on object and movement recognition. This was all kickstarted by a paper published in *Advances in Neural Information Processing Systems* by Krizhevsky et. al. [2]. In our analysis we do not make use of neural networks or deep learning, we will primarily use the Singular Value Decomposition (SVD) via PCA. In order to understand the data and attempt to classify the movements of the robot we aimed to complete the 5 tasks outlined in the assignment document. We made extensive use of several important Python packages. Namely, Matplotlib was used to create all plots and animations [1]. Additionally, Scikit-learn was the primary source of using the PCA algorithm and other classification methods [3]. Moreover there are important theoretical underpinnings behind the algorithm we implemented which will be cited and expounded upon in the following section.

2. Theoretical Background

The primary theoretical tools we used in our analysis were the Singular Value Decomposition (SVD) of a matrix X and principal component analysis (PCA) an application of the SVD. For the purposes of this report it suffices to describe the SVD at a high level and briefly describe how it is used in PCA. The SVD is indeed a decomposition of a matrix A into 3 matrices each encapsulating information about how the matrix A transforms a vector $\boldsymbol{x}$ when performing matrix vector multiplication such as $A\boldsymbol{x}$. The decomposition is denoted as follows

$$A = U\Sigma V^T. \tag{1}$$

where U and V^T are unitary matrices meaning they preserve magnitudes or norms and only project a vector into a new basis. Furthermore, Σ is a diagonal matrix with nonnegative entries on it's diagonal. The matrix, Σ, represents the scaling that A would do on the vector $\boldsymbol{x}$.

The diagonal entries of Σ are ordered singular values $\sigma_1 \geq \sigma_2 \geq \sigma_3 ... \geq 0$. In class we discussed the relationship between these singular values and the covariance matrix of A if A were our data of interest. This connection leads to PCA where we use the SVD to project our data to a lower

Date: March 11, 2025.

dimensional space but we preserve the dimensions with the greatest variance. Specifically, as discussed in class, retain the k-vectors from U that correspond to the k largest σ_i's

$$U_k^T X = \tilde{X}_{pc_k}. \tag{2}$$

Using this equation we can project our original data into k pc modes space. We make use of **scikit-learn**'s PCA class, which under the hood uses **scipy**'s SVD solver. Let's get into the actual implementation now.

3. Algorithm Implementation and Development

The training data we were working with was provided in 15 *.npy* files each containing 100 frames of the robot's 38 joint positions while performing one of three movements. We were provided with 5 samples of each movement, *walking, running,* and *jumping.* Additionally, it should be noted that the 38 joints' positions were provided in terms of 3 dimensional spatial coordinates. So each single sample was of the shape $(114, 100)$.

We want to perform PCA on the provided data using scikit-learn's implementation which expects the data input in the form of *(n_samples, n_dimensions).* In order to use sklearn, we needed to decide what shape made sense for our data. Initially I felt it made sense to reshape each of our 15 samples from $(114, 100)$ to $(11400,)$ which we could stack on top of one another and then we would have a data matrix of shape $(15, 11400)$ where each row corresponds to a single sample of robot movement. After some experimentation and consulting with classmates, the decision was made to retain the spatial dimension of 114 and to horizontally stack each of the 15 samples along the second dimension giving us a data matrix of shape $(114, 1500)$. With this shape we can simply transpose the data and we are ready to utilize sklearn's PCA class.

As more of an exploratory step we performed an analysis of how much information or energy is retained in our data matrix given a certain number of components are retained in the PCA transformation. To measure the energy or information of the data we used the following energy metric

$$E = ||A||_F^2 = \sum_{j=0}^{\min(m,n)} \sigma_j^2$$

where σ_j are the singular values of the (m, n) matrix A. This is the total energy of the matrix A. So when trying to calculate pc modes needed to preserve a given percentage (%) of total energy we calculate the partial cumulative sum over the singular values and normalize by dividing by the total energy E. See Figure 1 for a visualization of the percent of energy preserved by a given number of PCA components.

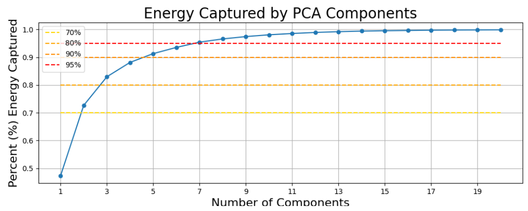

Figure 1. An analysis of how much information or energy is retained in our data matrix given a certain number of components are used in the PCA transformation.

Throughout the analysis we treat each frame of each sample as a single data point which has a label and is thus classified individually as well. We applied PCA to our compiled training data matrix twice first preserving 2 principal components and second, preserving 3 principal components. We visualize the points projected into these lower dimensions The projected data points (frames from the movement samples) are colored according to movement types. See Figure 2 for the visuals described. Notice, even in this low dimensional representation of our data the classes are well separated with minimal overlap between walking and jumping.

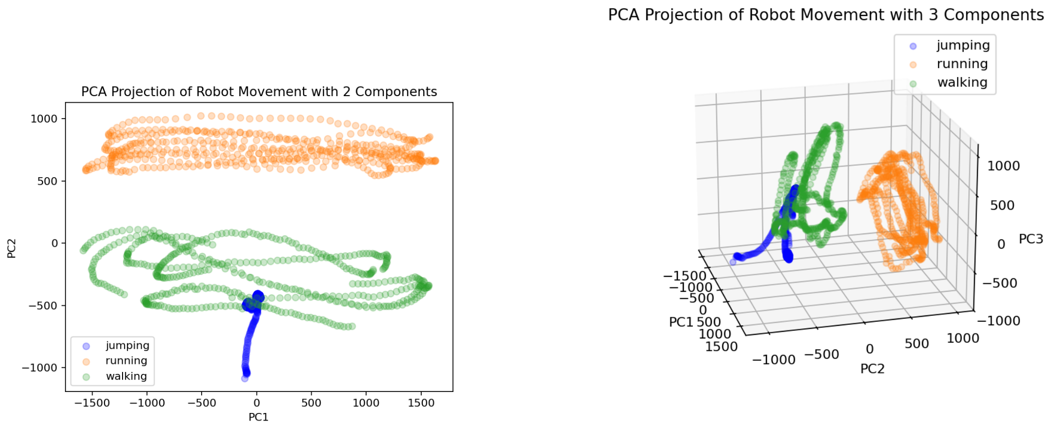

Figure 2. We have visualized the lower dimensional projection of the robot movement data (colored by movement) in both 2 and 3 dimensions. Notice, in this low dimensional representation the movements are well separated.

In order to plot according to color or to perform the next step in our algorithm, we made sure to carefully construct a ground truth array of labels which contained the corresponding movement type for each of our 1500 frames in the training data matrix of shape $(1500, 114)$ (after transposing for sklearn use). The next step was to calculate the centroid or sample mean of the collection of frames for each movement type. After calculating these centroids for each cluster of movement type, we implemented a basic classifier which classifies each point as the movement type based on which movement type's centroid it is closest to. This was easy to implement by writing a child class inheriting from sklearn's **ClassifierMixin** and **BaseEstimator** classes. See code for further details. This classifier was applied to the training dataset itself as well as freshly applied to the held out test dataset to evaluate the performance of this classifier algorithm. Additionally, we have included an implimentation of the k-nearest neighbors classifier. It's performance is compared to that of the mean centroid based classifier in the next section.

4. Computational Results

This section is only going to be a brief discussion of the results of this naive classifier based on training data centroids being calculated after PCA projections have been applied. First and foremost, take a look at the projection and classification visualizations. Figures 3 and 4 are for the training set but projected to 2 and 3 dimensions respectively. In both of these cases you can see immediately the walking and jumping clusters overlap a lot resulting in a lot of misclassification based on which centroid a given point is closest to.

We evaluated the accuracy of this classification method on the training set for various number of components to keep when using PCA. According to our results visualized on the left in Figure 5, the optimal number of components for accuracy would be any value of k greater than 11. However, we would likely want to choose the smallest k possible since this reduces the amount of data we are storing. So with that consideration we would choose an optimal k of 11.

Furthermore, we tested this method of classification on the held out test set with interesting results. Figures 6 and 7 are for the training set but projected to 2 and 3 dimensions respectively. In

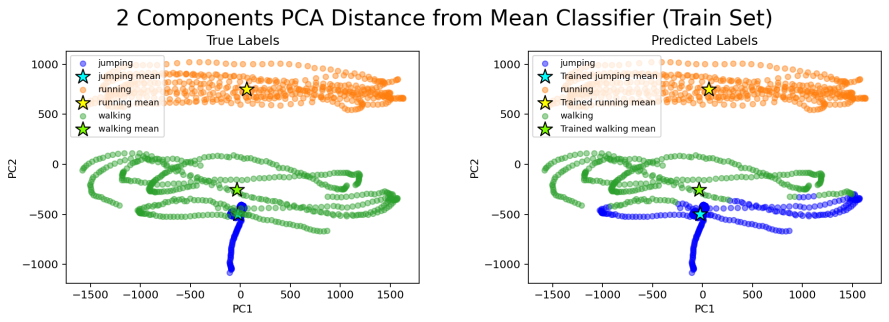

FIGURE 3. On the left is the training set projected into 2 dimensions. The centroids we calculated have also been plotted as stars. On the right we have recolored each projected point by the predicted label from the mean centroid based classifier.

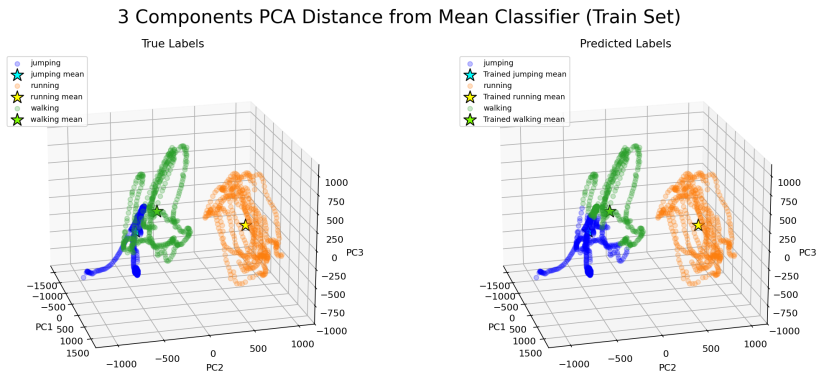

FIGURE 4. This contains the same information as Figure 3 with the exception that this is 3d.

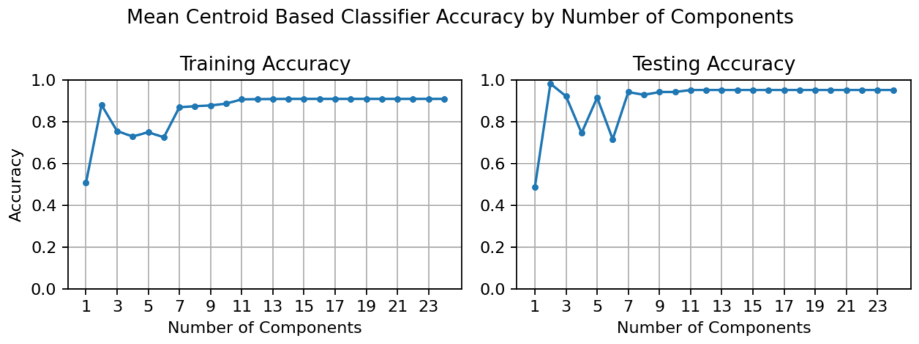

FIGURE 5. We performed this projection and classification for each possible number of PCA components for both training and test sets. In both cases, level of accuracy levels off well before keeping only 24 components (hence truncating the plot).

both of these cases we again see the walking and jumping clusters have issues. In only 2 dimensions there are only a few miscalculated walking points interpreted to be jumping points. This anomaly of high accuracy with only 2 components is largely due to the fact that in this dimension there is little overlap. This could easily be an artifact of just having a small sample and perhaps getting lucky that the particular walking sample didn't intersect directly with the jumping sample in the 2d projection.

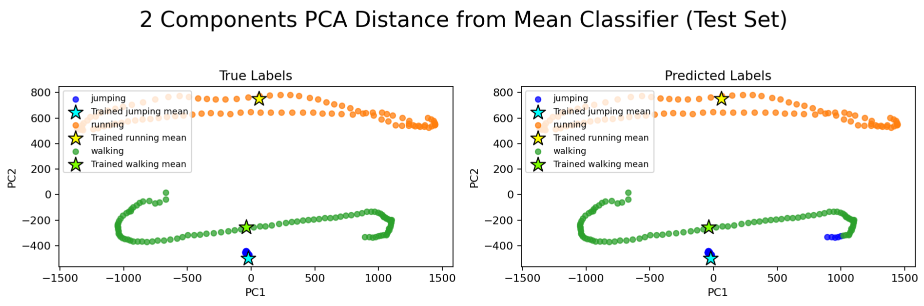

Figure 6. On the left is the test set projected into 2 dimensions. The centroids calculated from the training set are once again stars. On the right we have recolored each point by the predicted label from the mean centroid based classifier.

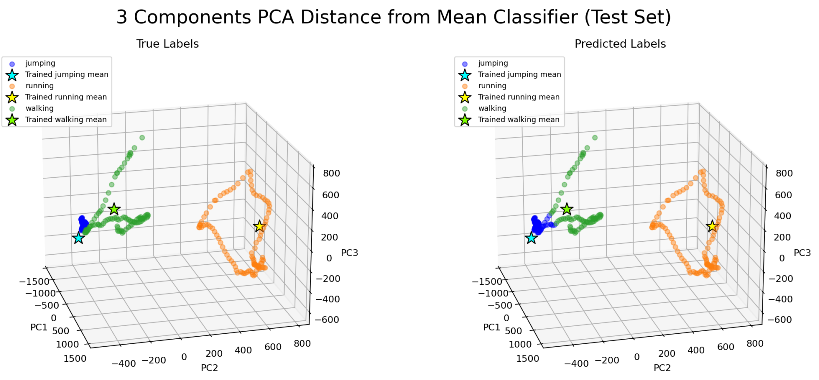

Figure 7. This contains the same information as Figure 6 with the exception that this is 3d.

Finally, we want to acknowledge an unexpected result which we found was that the maximum accuracy score on the test set was higher than that of the training set. We typically expect some overfitting of the training set and poor generalization which would mean high train accuracy and lower test accuracy. Once again, I suspect that this is an artifact of only training on 15 samples and only testing on 3 samples. This is a small training set and test set. If we could gather more data and perform these same experiments again, we would have a better idea of what the models accuracy is like.

Lastly, we discuss the results of using K-nearest neighbors as the classification method. As you can see in Figures 8 and 9 this algorithm proved to be very effective. This makes sense because we can visually see how separable these different movements are in the lower dimensional pc space. Therefore, K-nearest neighbors is well suited for the problem. Due to space constraints we have omitted the plot for the train set, but in Figure 9, we can see that it indeed performed very well on the test set even in 3d pc space. More plots and visuals are available on the author's GitHub page here.

5. Summary and Conclusions

Through our analysis we were able to see that some interesting structures in the data can be preserved even when reducing from 114 dimensions to 2 or 3. We saw the reasonable results of a

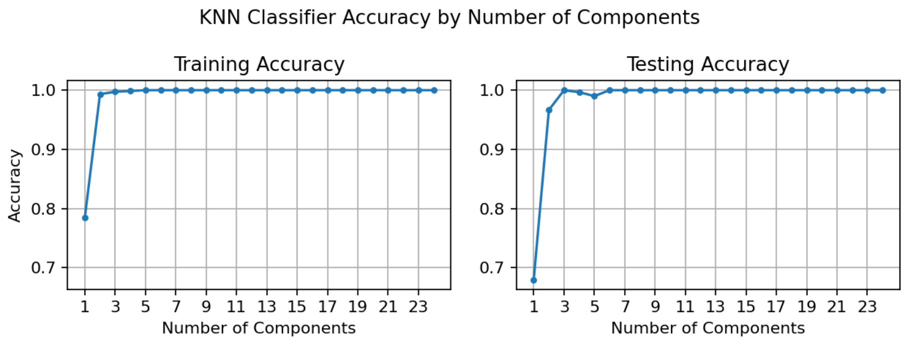

FIGURE 8. This plot should be compared with Figure 5. The only difference is this classification is based on sklearn's K-Nearest Neighbors classifier.

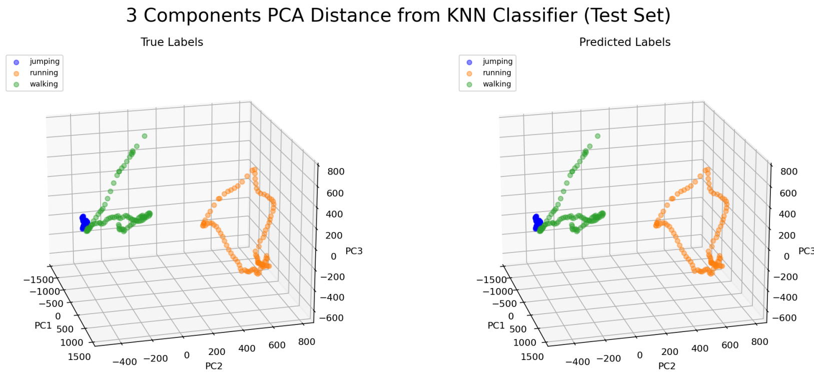

FIGURE 9. This plot should be compared with Figure 7. The only difference is this classification is based on sklearn's K-Nearest Neighbors classifier.

naive classification method and have ideas for further work. In the future we might analyze how implementing another classifier for example a support vector machine might impact the accuracy of the model. We would also like to try applying these methods to a larger number of samples to further expand our understanding of how these transformations and classifications generalize.

Acknowledgements

The author is thankful to Professor Eli Shlizerman for carefully instructing us in class. Finally, we thank Jaxon Tuggle, Nate Ward, Sophie Kamien, Anja Vogt, John Murphy, and Hailey Sparks who were available to share and discuss ideas throughout.

References

[1] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.
[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.