Cheng Chang, Yuan Tang, Mincheng Wang

Professor Eran Mukamel

COGS 109

04 June 2021

Classification of Distorted English Letters Using Principal Component Analysis and K-Nearest Neighbor

1. Introduction

Motivation: Recognition of human-written letters has always been a heated topic for the machine learning community. As the first step of investigating this topic, we are interested in whether, given a large set of letter images and the English alphabet as the target prediction space, we can train a machine-learning algorithm to distinguish between the letters with high accuracy. The benefit that this postulated algorithm is immense; a relatively seamless conversion from written texts to electronic documents would be both useful for daily life and academic works, such as literature preservation projects that are much needed for various ancient manuscripts.

Dataset: To further simplify the problem, the dataset of letter images that we use does not represent handwritten letters but capitalized letters of 20 different fonts with randomly generated distortions [1, 2]. The dataset contains a set of 16 attributes of 20,000 black-and-white images of English letters with such distortions, along with the letter that the images represent. Each character image was firstly defined in the form of a vector graphic, a computer graphic determined by line-connected points on a Cartesian plane, whose vector coordinates of end-points of its constituent line segments were scaled and "warping" as attributes. Rasterization was applied to convert those line segments into 45×45 arrays of pixels. The smallest rectangular image with a unique character in it is called a "box". The "on" pixels are the pixels representing the writings of those characters, and the "off" pixels are the ones representing the background image within each box. Each of the 16 attributes was scaled and fit to an integer range of 0 to 15.[1] The dataset is well balanced among the 26 letters. [3]

[1] The 16 attributes are the following:

1. The horizontal position, the number of pixels starting from the left edge to the center, of the box (x-box);
2. The vertical position, the number of pixels starting from the bottom to the center, of the box (y-box);
3. The width, counting pixels from the left to the right edge, of the box (width);
4. The height, counting pixels from the bottom to the top, of the box (high);
5. The total number of pixels in the character image (onpix);
6. The mean of the total differences between all horizontal positions of "on" pixels relative to the center horizontal position of the box, divided by width (x-bar);

Hypothesis: With this dataset, we want to compare the performance of K Nearest Neighbor (KNN) algorithms with various k-values for classifying the 26 letters. We also want to compare the best result of the naive KNN approach with KNN classifications of the dataset after performing various degrees of dimensionality reductions using principal component analysis (PCA). We hypothesize that KNN will not yield desirable accuracy for this particular data because of its high dimensionality. We further hypothesize that using KNN or the PCA-KNN pipeline to classify the dataset will not generate a desirable result for all 26 letters, but it will be able to classify letters that are most distinguishable in shape from other letters. Nevertheless, we want to find the algorithm with specific hyperparameters that achieves the best classification accuracy for a testing dataset.

2. Method

General processing: The dataset is well organized, so not too much data cleaning is needed. Dataset is cast into a pandas DataFrame, with each attribute name manually input based on the dataset description. Since this project aims to classify 26 distorted English alphabets and make predictions over a test dataset, we first split the entire dataset in an 8:2 ratio. 80% of the data, 16000 observations in total, is used for training, and 20% of the data, 4000 observations, is reserved for testing. We use the Scikit-learn "train_test_split" method and stratified the split result to ensure that the even ratios among letters of different alphabets are properly maintained among the training and testing set.

K-Nearest Neighbor (KNN): The model we try to investigate for this project is the K-nearest neighbor. We choose this model because our dataset is labeled and multiclass, and we think KNN is a supervised learning algorithm that can efficiently perform multiclass

7. The mean of the total differences between all vertical positions of "on" pixels relative to the center vertical position of the box, divided by height (y-bar);
8. The mean squared deviation of all "on" pixels' horizontal positions from the center of the box (x2bar);
9. The mean squared deviation of all "on" pixels' vertical positions from the center of the box (y2bar);
10. The mean product of the horizontal and vertical distances for each "on" pixel (xybar).
11. The mean product of the squared horizontal distance and the vertical distance for each "on" pixel (x2ybr);
12. The mean product of the horizontal distance and the squared vertical distance for each "on" pixel (xy2br);
13. The mean number of edges where an "on" pixel is immediately at the right to either an "off" pixel or the image boundary (x-ege);
14. The sum of the vertical positions of edges encountered in 13 (xegvy);
15. The mean number of edges where an "on" pixels immediately above to either an "off" pixel or the image boundary (y-ege);
16. The sum of the horizontal positions of edges encountered in 15 (yegvx). [3]

classification. We notice that our dataset has 16 features, which may cause the "curse of dimensionality", meaning that the data may be too sparse and their Euclidean distances may be uniformly distributed, which may diminish the performance of KNN. However, the extensive amount of total observations may remedy the issue.

We tune two hyperparameters of KNN: The weight and the k-values. For the weight, we choose between "distance" and "uniform". Weighing on distance means that the further the neighbor is to the data point, the less the deciding power it has to impact the label of the data point. Conversely, weighing uniformly means that regardless of the distance between the neighbors and the data point, all neighbors have equal deciding power over the data point. For the k-values, we go through all of the odd numbers ranging from 1 to 55 (28 in total). A smaller k-value stands for a more flexible KNN model. We achieve these processes by using the Scikit-learn “KNeighborsClassifier” function.

Cross-Validation (CV) of KNN without PCA: We use 5-fold stratified cross-validation to select the best model. To do this, we first shuffle the data index of the training dataset. Then the training set is divided into five different groups without repetition. The cross-validation is performed in a stratified manner, meaning that the proportion of each letter remains the same ratio as the original data. Each time, we treat one group of observations (with 3200 data points) as the validation set and the other four groups as the CV-training set for cross-validation. Every time we train the model in the CV-training set, we use the model to make predictions on its corresponding validation set to get the accuracy score. By going through all the CV-training-validation combinations, we get five different validation accuracy scores in total. In the end, we take the average of the five accuracy scores as the cross-validation performance of each model and select the hyperparameters of the KNN model with the highest average validation accuracy as the hyperparameters for our output KNN algorithm. The output KNN is then trained with the entire training dataset and used to generate an accuracy score for the testing set. We use the “KNeighborsClassifier”, “Pipeline”, and “GridSearchCV” functions of Scikit-learn to implement the cross-validation process.

Principal Component Analysis (PCA) and KNN: We want to further consider the results generated by KNN algorithms that are provided with different numbers of primary principal components after PCA. Using “Minka’s MLE” feature of the PCA function provided by Scikit-learn, we find that the first 14 principal components can explain around 98.83% of the total variance of the data (Figure 1). Therefore, we construct the pipeline of feeding in the first 2-14 principal components of PCA over the training dataset to the cross-validation machinery for KNN described above. The 13 CV processes, each performing cross-validations for KNN with the 28 k-values and the weighting scheme (56 in total), try to find the KNN models with the best average validation accuracy when using 2-14 principal components. The best KNNs from each group of principal component usage are then compared by training them over the entire training dataset and generating test accuracy scores for them with the test dataset. They are also compared with the KNN result without PCA.

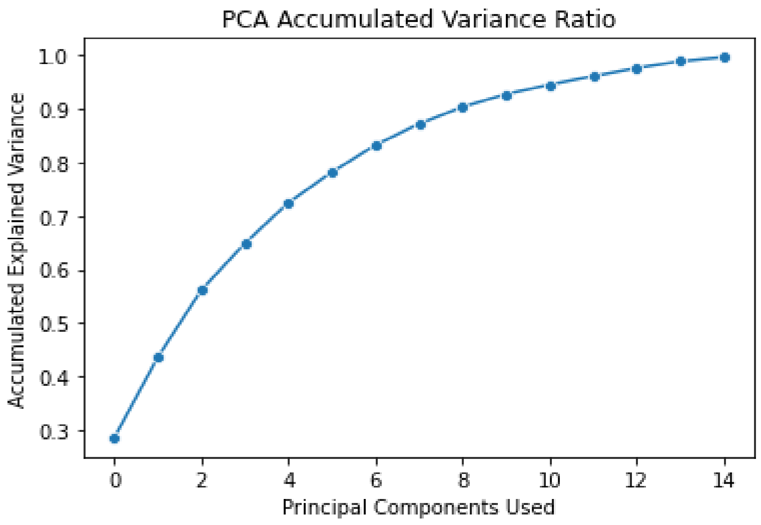

Figure 1

3. Results

Model selection of KNN without PCA: We choose our best KNN model based on the score of average validating accuracy through cross-validation. A higher accuracy score stands for better KNN performance. The best KNN model without PCA with weight "distance" and K value of 3, with a mean testing accuracy of 95.4%.

Figure 2a illustrates the average performance result of each model based on the validation accuracy metric. The y-axis represents the mean validating accuracy scores across five trials of cross-validation. The blue line represents KNN with weight "distance," while the red line shows KNN with weight "uniform" across all K values. The "distance" KNN triumphs the "uniform" for every K value. And the highest performance of 94.4% validation accuracy is obtained by KNN with weight "distance" and K of 3.

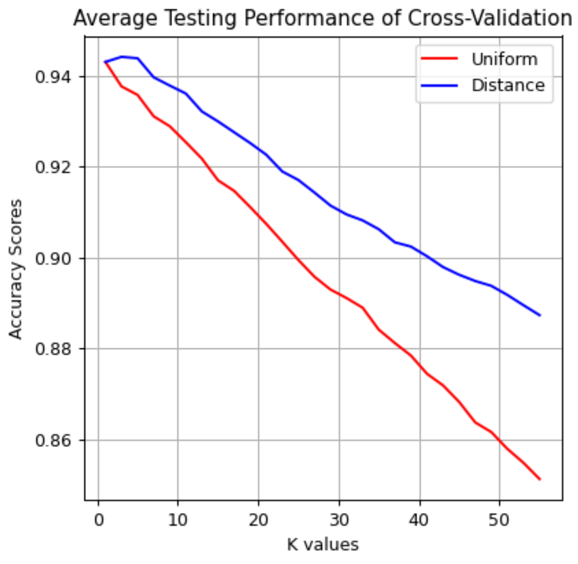

Figure 2a

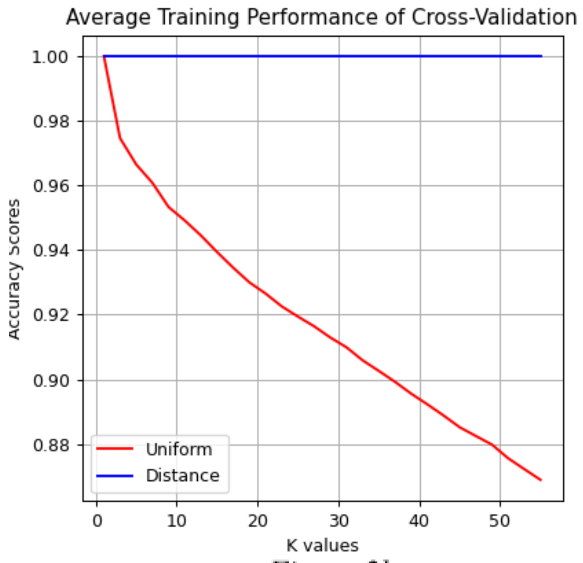

Figure 2b

Figure 2b shows the mean training accuracy of cross-validation. When KNN's weight is based on "distance," The training accuracy is always 1, while for the "uniform" weight KNN, the mean accuracy performance drops faster when the K value increases.

Model estimation of KNN without PCA: The best model selected is KNN with weight "distance" and K value of 3. Since KNN is a nonparametric supervised machine learning method algorithm, we cannot report the parameter estimates. After refitting the model to 16000 training observations, we get an Accuracy of 1 for the training score and 0.954 for the testing prediction. The model correctly classified 100% of the training data and accurately predicted 95.4% of the testing set.

Figure 3 shows the confusion matrix when the best KNN model predicts the testing dataset. The darker the color, the more predictions are made between the actual letter and the predicted letter. Most predictions are made on the diagonal of the matrix, meaning that all individual letters are properly predicted, or the testing accuracy for each letter is very high.

In the end, to prepare the model for new data, we train the model by using the whole dataset and get a training accuracy score of 1.

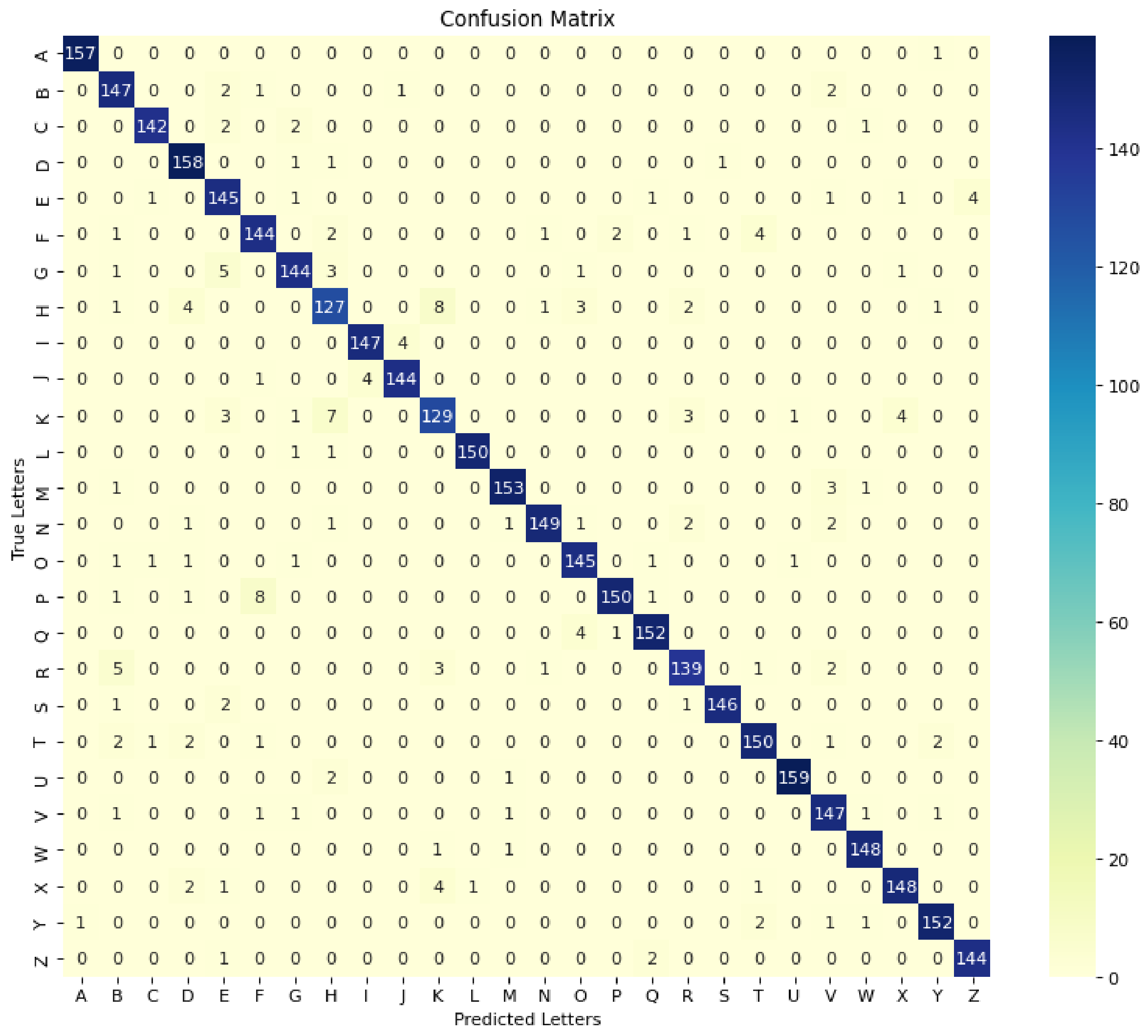

Figure 3

Results of KNN after PCA: Figure 4a shows the KNN cross-validation time needed after selecting 2-14 principal components from PCA for classifications. The time varies each time the pipeline is run, but the overall trend remains the same. KNN algorithms with 14 principal components use the longest time for multiple model fitting, while KNNs with 3 principal components use the least time.

Figure 4b shows the k-values of each PCA-KNN algorithm with the best validation results for different principal components used. There is a general downward trend; for more principal components used, the k-values of the best KNN algorithm decrease.

Figure 4c shows the test accuracy of the best KNN within each PCA group. As more components are used, the accuracy score increases logarithmically and caps at around 12-14 principal components.

The selection of the desirable PCA-KNN model is more complex. 6 models achieve a final accuracy score of over 90%. Two of the models (k=1, weight=uniform, principal components used=13 and k=5, weight=distance, principal components used=14) achieve a score of over 95%. For faster computation time, PCA-KNN with k=3, weight=distance, and principal components used=9 achieves a test accuracy score of 92.3% in around 20 seconds. For overall best performance, the test accuracy of the KNN algorithm with the "uniform" weighting scheme, a k-value of 1, and using 13 principal components has a final test accuracy of 95.6%. Using the PCA-KNN cross-validation pipeline over the entire dataset shows the same optimal hyperparameter assignment and a training accuracy of 1.

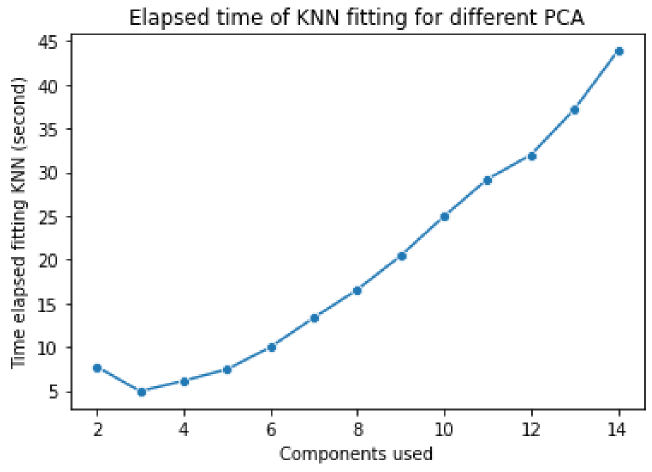

Figure 4a

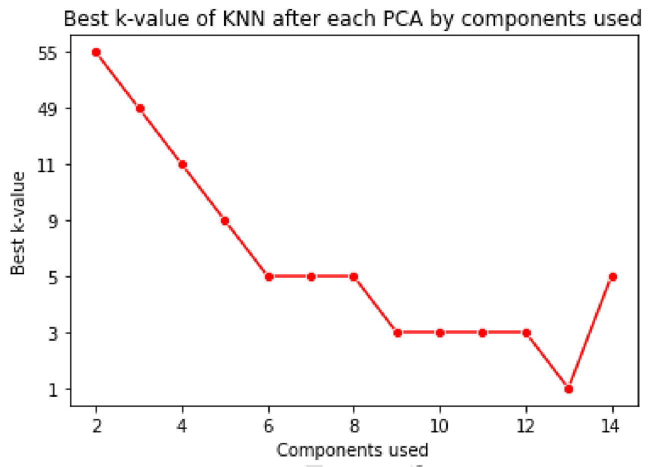

Figure 4b

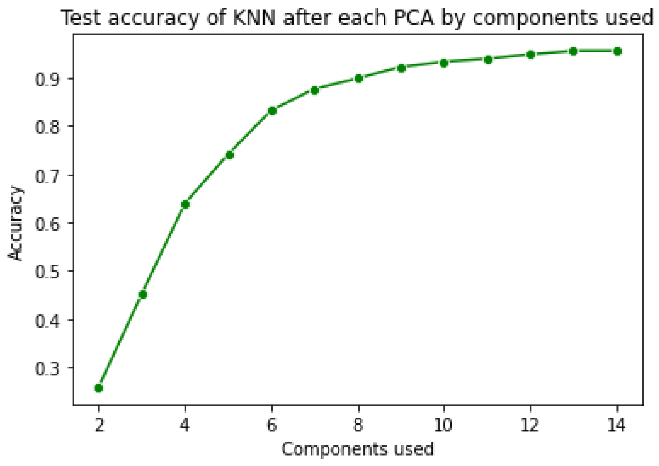

Figure 4c

4. Conclusion and discussion

KNN without PCA: KNN performs extraordinarily well for classifying English alphabets in this particular dataset, proving that our hypothesis is too conservative. We think the large number of observations in our dataset may increase the density of the data space, which remedies the negative cost carried forth by the high dimensionality.

Figure 3 shows that the algorithm makes most of the mistakes when distinguishing letters that are very similar to each other in shapes, such as “P” and “F” (10 total bi-directional misclassifications), “H” and “K” (15 misclassifications), and “I” and “J” (8 misclassifications). Nevertheless, we are also surprised that KNN successfully distinguishes between a lot of the ambiguous letters, such as "O" and "Q."

The increment of K stands for the reduction of model flexibility. For the uniformly weighted KNN, models seem never to overfit the data because testing accuracy drops along with training accuracy (Figure 2). The "distance" weighted KNN shows a different pattern; if K is lower than 3, the testing performance increases when model flexibility decreases, meaning that when the model is more flexible, the model overfits the training set (Figure 2). For all k-values larger than 3, the model seems to underfit the data. The model prediction result seems to favor models with high flexibility in this case.

PCA-KNN: One benefit of conducting PCA before classification tasks that is demonstrated by Figure 4a is that it helps reduce computational complexity. The computation time of KNN reduces linearly as fewer principal components are used for classification. However, when using too few principal components for KNN classification, PCA drastically reduces the model strength of KNN, which suggests that finding the right number of principal components to use is very important.

As fewer principal components are used, the optimal k-values increase (Figure 4b). This seems to suggest that as less information about the true cluster of data is passed into KNN, a more inflexible decision boundary would generate better classifications. This is an interesting finding that seems to conform to our understanding of model complexity. For example, for fewer data points provided in a 2D plane, linear regression will tend to have better test performance than polynomial regression due to its lower model complexity.

However, when more dimensions of data are passed in, KNN performs better with smaller k-values. This seems to confirm that, in a high dimensional space, the uniform distribution of the Euclidean distance of points makes it harder for KNN to predict the category of a new data point by its neighbors that are farther away, because all data points are within similar proximities. It seems that the large data set size helps KNN to find a solution in this case, which is to form small local clusters of data points of the same category and predict new data points only by comparing them with local neighbors.

For further study, researchers can perform comparisons between KNN and other supervised machine learning algorithms. For example, linear discriminant analysis (LDA), neural networks, and decision trees are suitable for multiclass classification tasks. More sophisticated comparisons can be made by randomly splitting training and testing sets and implementing cross-validation for each split. More performance metrics can be tested: F1, ROC_AUC, recall, sensitivity, etc. Paired t-test and ANOVA may be used to show the statistical difference between different algorithms.

5. Code

All source code of this project is hosted in the GitHub repository: https://github.com/chc012/cogs109_finalproject_2021.

6. References

[1] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[2] Letter Recognition Data Set. https://archive.ics.uci.edu/ml/datasets/Letter+Recognition

[3] P. W. Frey and D. J. Slate. "Letter Recognition Using Holland-style Adaptive Classifiers". (Machine Learning Vol 6 #2 March 91)

[4] Scikit-learn: Machine Learning in Python, Pedregosa et.al., JMLR 12, pp. 2825-2830, 2011.