Chwan-Hao Tung
861052182
12/6/2016
CS229

CS229 Final Project Report

1 Approach to the Problem

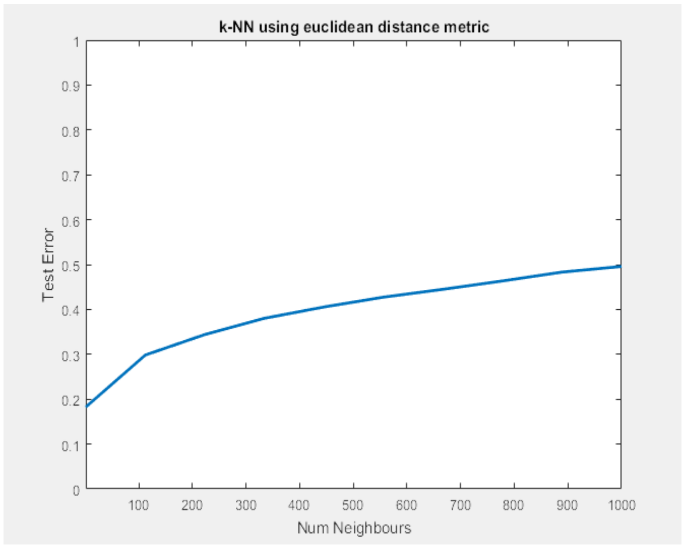
Handwriting recognition pertains to the problem of classification. We have 26 different labels used to classify different images of handwriting of lower-case letters. My first instinct is that the data will not be linearly separable considering that handwriting can have many different forms with different letters sharing similar characteristics (e.g letter 'i' and 'l').

For my initial classifier, I wanted a relatively simple classifier that can produce models quickly so I can check the error quicker when I adjust the classifier parameters. In addition, it's much easier to understand the deficiencies of the model or the data which allows me to come up with more concrete ideas to improve the classifier. The classifier must also need to be able to predict categories with labelled data with many features.

For those reasons, I chose to use k-nearest neighbors as my baseline classifier. It is conceptually very simple and easy to use. I can create many different models with different distance metrics or number of neighbors in a short time and compare their effectiveness. There is also a lot of room for improvements if k-NN by itself proves insufficient.

2 Initial Results and Analysis

To start off, I split 90% handwriting data into training data, and the remaining 10% into testing data using random indices. To pick the number of neighbors for the classifier, I compared the resulting testing error by using increasing number of neighbors while holding all other parameters on default.



k-NN testing error vs. number of neighbors

The trend appears to be that with higher number of neighbors, the testing error increases in a logarithmic fashion. So I narrowed down the range to k = [1,10] and picked k=5 as that gave me the lowest testing error of **16.64%** with default euclidean distance metric and nearest tiebreaker.

I wanted to also weight the ‘vote’ of the neighbors. Intuitively, the further away a neighbor is the less important its ‘vote’ should be. MATLAB’s fitcknn allows us to set the distance weight function for evaluating the nearest neighbors’ distances. So by changing the distance weight to squared inverse, I get a small improvement and reached **16.59%** testing error.

The testing error still remains high. One of the reasons probably had to do with how euclidean distance metric doesn’t work well in higher dimensions due to the curse of dimensionality[1]. So the next immediate step would be to choose a better distance metric. Since we’re dealing with mostly binary (black, white) data, Jaccard distance metric seemed like a good candidate as it ‘ignores’ zero values in its calculations and only cares about differences in nonzero coordinates.

So after training the data with k-NN using Jaccard distance metric and squared inverse distance weighting, I reach the initial result of **14.67%** testing error.

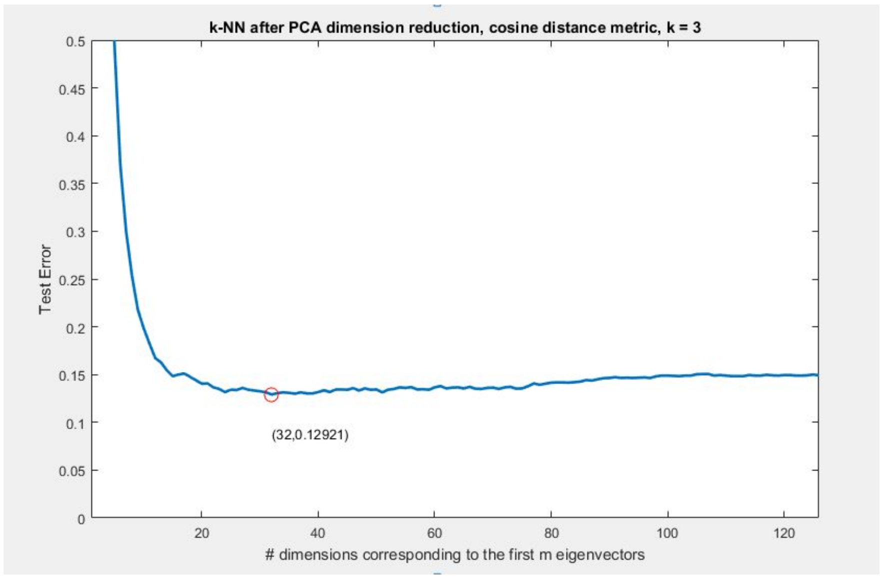
The testing error is still unsatisfactory. Looking at examples of misclassified handwritten letters, disregarding the capital letters, it would seem that a fair majority of the examples are cases where the letter ‘i’ gets mispredicted as ‘l’. In addition, those ‘i’s are written in a way such at the body stretches from top to bottom with 1 pixel gap near the top or sometimes even without the gap. Generally, examples where letters are written in a very elongated way seems to get misclassified more often. These cases can probably be categorized as the outliers of their letters. k-NN’s decision boundaries are very susceptible to noise and outliers. In addition, most of the the distance metrics k-NN don’t work well with high dimensions. It would be ideal to find an improvement that deals with these two issues.

3 Description of Improvement

My idea for improvement is to determine which features corresponds to pixels that are rarely used, such as the last bottom row of pixels and all the whitespace most letters share and make them ‘weigh’ less or maybe even be disregarded. Or to amplify the weights of the pixels that defines a specific letter. By doing this, hopefully it would reduce noise or the outliers impact on the performance of k-NN.

One of the first methods that came to mind was using PCA to do dimensionality reduction on the data and then do k-NN. By applying PCA on the training data we get back the coefficient matrix, the principal component variances, and the score matrix. By looking at the variances we can determine the features that give the highest variance which would affect classification the most. To determine how many dimensions I should include, I trained a k-NN model using the score matrix, reconstructed the test data using the coefficient matrix, and calculated testing error vs. the how many of the first m dimensions I included.

4 Final Results and Analysis



Testing error vs. number of features adopted

Training and testing on the first 32 features of the score matrix gives me the lowest testing error of **12.92%**.

By extracting the highest impact features of our data, PCA has slightly alleviated the issue of outliers and also helped with performance of k-NN distance metrics. By examining the misclassified examples, I have noticed that most cases are irregular variations of lower case letters such as capital letters. In addition, the letter ‘l’ is misclassified as ‘i’ quite often, which is the opposite of what was common before the improvement.

If I were to further improve the accuracy, I would focus on how to treat outliers such as capital letters in a better way. One idea that comes to mind is doing clustering and picking the most representative data for each label. Perhaps cluster the capital letters of each letter into their own group somehow. Another idea would be to use tangent distances as our distance metric for k-NN. That would help with recognising rotations and different sizes of the same letter.

5 References

[1]. Pg 22. Ch 2.5 Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning, Second Edition: Data Mining, Inference, and Prediction.* New York: Springer, 2009.