News: We have put all the code we wrote for the assignments of this course on Github https://github.com/Xiang-Gu/CS245_Principle_Data_Science
Since we do not show any code in the final report we turn in, feel free to check out the Github repo for any code-related issues. Furthermore, the code contains very useful comments that are worth reading if you want to reproduce our results!

Xiang Gu 5130309729
Tao Zhu 515020910243
Weihong Lin 515030910643
Chacha Chen 515021910302

k-Nearest Neighbor with Different Distance Metrics

0. Assignment Overview

In this assignment, we will practice another importance technique learned in class for classification task -- k-Nearest Neighbors (kNN). There are many variants of kNN out there but we will use the simplest one:
Assume we have a bunch of data points of different classes. Then for a new data point that we want to predict its class, we simply compute the k-nearest data points according to some distance metric and predict its class to be whatever class the majority of classes of these k nearest data points.

Like in assignment 1, we will use the AwA2 data set with the provided pre-extracted features. The main goal of this assignment is to play with different distance measures and see what is the optimal one. Without further due, let's get started.

1.Experiment Setup

We used the Animals with Attributes (AwA2) dataset from https://cvml.ist.ac.at/AwA2/, which consists of 37322 images of 50 animal classes with pre-extracted features for each image. We split the data set into a training set (60%) and a test set (40%). We then use the pre-extracted features in the training set to build a kNN classifier and test its performance (i.e. classification accuracy) on the test set. In the following sections, we will try using Manhattan distance, Euclidean distance, Chebyshev distance, Cosine distance, and finally a method to automatically learn the optimal distance metric for kNN given k.

2.Euclidean Distance

We start with the Euclidean distance. Namely, for any two points p1, p2, their Euclidean distance is given by

$$d(p_1, p_2) = (\sum_{i=1} |p_{1i} - p_{2i}|^2)^{\frac{1}{2}}$$

We use the kNeighborsClassifier from sklearn.neighbors to build the kNN. There is one hyperparameter k, the number of nearest neighbors, that we need to pick. Therefore, we use the Euclidean distance to perform 5-fold cross-validation within the training set to determine the best k. Due to computation limitations, we only try 10 possible values of k [1, 2, …, 10]. The results are shown below:

K	1	2	3	4	5	6	7	8	9	10
Mean Accuracy on 5-fold cross-validation	0.874 686 (+/- 0.009 534)	0.860 709 (+/- 0.010 099)	0.883 756 (+/- 0.006 013)	0.882 278 (+/- 0.006 234)	0.886 962 (+/- 0.007 648)	0.885 088 (+/- 0.008 137)	0.887 903 (+/- 0.007 351)	0.885 492 (+/- 0.007 048)	0.887 012 (+/- 0.008 589)	0.885 358 (+/- 0.006 760)

Table 1: Results of hyperparameter search. The number after +/- represents the 95% confidence interval of the mean accuracy.

From the result, we can see that k = 7 gives the highest accuracy, although it wins by only a very small margin. We will then stick to k = 7 for all other distance metrics in the following experiments.

Finally, we build a kNN (k=7) from the entire training set and test it on the test set. Here is the accuracy to report:

Accuracy = 0.894501

2.Manhattan Distance

The second distance metric we try is the Manhattan distance. For any two points p1, p2, their Mahanttan distance is

$$d(p_1, p_2) = \sum_{i=1} |p_{1i} - p_{2i}|$$

Same as before, we build a kNN classifier with k being 7 and test it on the test set. Here is the result we report:

Accuracy = 0.8859937035300423

3.Chebyshev Distance

The third distance metric we try is the Chebyshev distance. For any two points p1, p2, their Chebyshev distance is computed as

$$d(p_1, p_2) = \max_i |p_{1i} - p_{2i}|$$

Again, after building a kNN classifier with k being 7, the accuracy evaluated on the test set is

Accuracy = 0.7830397213477125

4.Cosine Similarity Metric

It is a common technique to use cosine to measure the similarity between two vectors. Concretely, the cosine similarity between two points p1 and p2 is given by

$$Similarity(p_1, p_2) = \frac{{p_1}^T p_2}{\|p_1\| \, \|p_2\|}$$

However, this formula above is just a similarity measure rather than a valid distance metric because it does not satisfy the four properties a valid distance metric must satisfy (e.g. this similarity outputs results between [-1, 1] which violates the non-negativity property of a distance metric). Thus, we do some transformation on the output of this similarity function. Namely,

$$d(p_1, p_2) = 1 - \frac{1 + similarity(p_1, p_2)}{2}$$

It is not too hard to show that this new function is not a valid distance metric either, because it does not satisfy the 'triangle inequality' property (see Appendix for more detail). It does satisfy the remaining three though, so no matter what, we still plug it in and see how well a kNN performs with this distance measure empirically.

We build a kNN classifier (k=7) using this distance measure on the training set. The classification accuracy on the test set is

Accuracy = 0.90635675530846

Amazing! This 'semi-metric' outperforms all the previous three distance metrics.

5.Metric Learning -- Large Margin Nearest Neighbor (LMNN)

So far we have tried several different distance measure metrics, but there is simply more possibilities out there. We can, of course, go ahead and try all them one by one, but wouldn't it be nice if we can learn an optimal distance metric from the training set that maximizes the resulting kNN classifier accuracy?

We did some research on the internet and found one paper -- Weinberger, K. Q., Saul, L. K. *Distance Metric Learning for Large Margin Nearest Neighbor Classification. JMLR 2009* -- that learns a distance metric that maximizes the resulting kNN classifier performance given k. They proposed a framework called Large Margin Nearest Neighbor (LMNN) to accomplish this task.

Fortunately, someone has already written this algorithm in python and encapsulated it to pylmnn package. We installed and used this package to implement our experiment. For more information, check out https://pypi.org/project/PyLMNN/.

We ran the algorithm for several different numbers of iterations to learn a transformation/distance metric. Then we build a kNN (k=7) from the training set and tested the classification accuracy on the test set. The pylmnn package involves setting up a maximal number of iterations this algorithm is allowed to train on, so we tried different

numbers of maximal iteration and recorded the performance of the kNN (k=7) built with the learned distance metric when tested on the test set:

# of iterations	5	50	100	200	400
Accuracy	0.90823	0.92109	0.91827	0.93114	0.93147

Table 2: Accuracy of kNN built with learned distance metric using LMNN. # of iterations means the number of iterations allowed for the LMNN algorithm to learn a distance metric.

As we can see from the result above, using LMNN to learn a distance metric indeed helps the performance of kNN with significant performance improvement from around 90% (using cosine similarity) to 93%.

6.Conclusion

In this report, we evaluate several distance metrics for kNN, a classic non-parametric method for classification. We test four common distance metrics -- Mahanttan distance, Euclidean distance, Chebyshev distance, and (modified) cosine distance -- as well as one distance metric learning method -- LMNN. Among the four common metrics, cosine distance gives the best performance. LMNN does an even better job than cosine distance by improving the accuracy by more than 3%. Our experiments serve as a piece of empirical evidence of supporting metric learning method over common choices of distance metrics, albeit learning a good distance metric can also be very time-consuming.

7.Reference

Weinberger, K. Q., & Saul, L. K. (2009). Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, *10*(Feb), 207-244.

8.Appendix

Properties of $d(p_1, p_2) = 1 - \frac{1+similarity(p_1,p_2)}{2} = 1 - \frac{1+\frac{{p_1}^T p_2}{||p_1||\,||p_2||}}{2}$

- Non-negativity -- $d(x,y) \geq 0, \quad \forall x,y$: Since $\frac{x^T y}{||x||\,||y||}$ is between -1 and 1 (Cauchy-Schwarz Inequality), $d(x,y) = 1 - \frac{1+\frac{x^T y}{||x||\,||y||}}{2}$ is between 0 and 1.
- Symmetry -- $d(x,y) = d(y,x)$: it is obviously true since $x^T y = y^T x$
- Identity of indiscernibles -- $d(x,y) = 0 \leftrightarrow x = y$: From left to right: if $d(x,y) = 0$, then we have $\frac{x^T y}{||x||\,||y||} = 1$. Then again, according to Cauchy-Schwarz inequality, $\frac{x^T y}{||x||\,||y||} = 1$ if and only if $x = y$. From right to left: it is trivial to show this direction holds.
- Triangle Inequality -- $d(x,y) + d(y,z) \geq d(x,z)$: Unfortunately, we have found a counterexample to this property (acknowledgment to Tao Yu from Zhiyuan Math): Consider a case where x and y are both 2-dimensional vectors, and the angle between x and y and the angle between y and z are both $\frac{\pi}{3}$. Then $d(x,y) = d(y,z) = 1 - \frac{1+\frac{1}{2}}{2} = \frac{1}{4}$, so $left = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$ .But $right = d(x,z) = 1 - \frac{1-\frac{1}{2}}{2} = \frac{3}{4}$. So this property does not hold for this function.