2IMW30 - FOUNDATION OF DATA MINING
Assignment 1b Report

Duy Pham (0980384, d.pham.duy@student.tue.nl)
Mazen Aly(0978251, m.aly@student.tue.nl)

(Both students contributed equally in this assignment)

EXERCISE 1

In this assignment, we implement the nearest neighbor classifier using the cosine distance to recognize handwritten digits.

We try three approaches for this task. The first one is implementing everything from scratch, the second one is using Numpy library to compute the dot products and hanle the math operations, and the third technique is using Scikit Learn library only for making comparison and validating our implementation.

It worth noting that we changed the original code a little bit for running it in Python 3.5 as it was Python 2.7. The code is available in the Github repository `https://github.com/MazenAly/datamining` .

The dataset that we have contains 50,000 training data points and 10,000 testing data points.

The idea of our implementation is for every test data point, we loop on all the training points and find the one which has the smallest angle (min cosine distance) to that test point. Then we assign the label of the nearest neighbor to the test point.

We then build the confusion matrix of all the labels (digits form 0 to 9). The accuracy is measured by dividing the number of correct predictions by the total number of predictions, the precision is measured by dividing the true positives by the summation of true positives and false positives of each class, and the recall is measured by dividing the true positives by the summation of true positives and false negatives of each class. The detailed code of the nearest-neighbor module is shown in the code snippet below. The full version is accessible via the Github repository.

```
# Manual dot product
def dot_product(v1, v2):
    return sum(map(lambda x: x[0] * x[1], zip(v1, v2)))

def cosine_similarity(v1, v2):
    prod = np.dot(v1, v2)
    # len and len2 are the magnitudes of the 2 vectors
    len1 = np.sqrt(np.dot(v1, v1))
    len2 = np.sqrt(np.dot(v2, v2))
    return np.divide(prod , np.multiply(len1 , len2))

def NNClassifier():
    #opening two files for writing the predications and the true labels.
    f_predicted = open('predicted.txt', 'w')
    f_label = open('label.txt', 'w')
    #for each test entry, we will loop on every training entry and
    #check if 'ts the closest neighbor so far or not.
    for test_entry in test_data:
        test_features = test_entry[0]
        test_label = test_entry[1]
        # defining variables to save the closest neighbor
        max_sim = 0
        nearest_neighbor = 0
        for training_entry in training_data:
            training_entry_features = training_entry[0]
            #measuring the cosine similarity using scipy library
            sim = cosine_similarity(training_entry_features, test_features)
            # if this entry has the max similarity so far then
            # save this entry as nearest neighbor
            if sim > max_sim:
                max_sim = sim
                nearest_neighbor = training_entry
        # to know the predicated label we get the index of the 1 value,
        # as the label of the training data is an array of 10 values,
        # zeros for all and 1 for the label
        predicated_label = list(nearest_neighbor[1]).index(1)
        #writing the predicated and the actual labels in the files
        f_predicted.write(str(predicated_label) + '\n')
```

```
        f_label.write(str(test_label) + '\n')
```

The resulting confusion matrix is shown in table 1. The rows are the predicted labels, and the columns are the true labels. The accuracy is thus computed by summing all the diagonal values (correctly classified) and dividing by the population.

Our implementation (from scratch) has the same accuracy as the Scikit Learn library, which is 0.9708 over all 10000 test points. The error is thus 0.0292. We can see that the diagonal values are extremely bigger than the other cells. In general, nearest-neighbor classifier is very good for classifying discrete hand digits.

Class "9" has the worst precision, 0.93. According to the confusion matrix, 32 instances of class "4" are mis-classified as "9", which is the highest error among all cells. This can be explained that the shapes of "4" and "9" are more similar than other pairs of digits, especially on handwritten digits.

Class "5" has the worst recall, 0.94. According to the confusion matrix, 19 instances of class "5" are mis-classified as "3". The previous explanation might still be possible for this case.

Regarding the runtime, when we use our own implementation of cosine similarity as shown in the code, it takes 120 seconds on average to process an instance, which results in an estimated runtime of 333 hours in total. It is too slow. That is why we try using the Numpy library to handle the math. The performance is much improved, when it takes only 0.3 seconds on average to handle an instance, which results in only an hour to classify the whole test set.

Table 1: Confusion Matrix

pred-true	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**	**Precision**
0	978	0	8	0	0	1	3	1	4	9	1004	0.97
1	1	1128	0	0	3	0	3	11	2	6	1154	0.97
2	0	3	1005	1	1	0	0	5	2	1	1018	0.98
3	0	1	5	974	0	19	0	2	15	4	1020	0.95
4	0	1	0	1	937	1	2	1	2	9	954	0.98
5	0	1	0	14	0	847	3	0	4	2	871	0.97
6	0	1	1	0	6	11	947	0	5	1	972	0.97
7	1	0	10	4	2	1	0	997	5	9	1029	0.96
8	0	0	2	8	1	6	0	0	931	4	952	0.97
9	0	0	1	8	32	6	0	11	4	964	1026	0.93
Total	980	1135	1032	1010	982	892	958	1028	974	1009		
Recall	0.99	0.99	0.97	0.96	0.95	0.94	0.98	0.96	0.95	0.95		

EXERCISE 2

In this excercise, we are proving that barycenter computed in the update step minimizes the cost function.

Let's first assume that $d = 1$. Then the cost function is

$$\phi(U, b) = \Sigma_{p_i \in U} ||p_i - b||^2$$

The function is minimized when its derivative is 0. In this case, we are considering the changes of the function with respect to the way we choose the center. Thus, we can take the partial derivative with respect to b.

$$\begin{aligned}
\frac{d}{d_b} \Sigma_{p_i \in U} ||p_i - b||^2 &= \Sigma_{p_i \in U} \frac{d}{d_b} ||p_i - b||^2 \\
&= \Sigma_{p_i \in U} 2 \cdot ||p_i - b|| \cdot \frac{||p_i - b||}{p_i - b} \cdot (-1) \\
&= 2\Sigma_{p_i \in U} (b - p_i)
\end{aligned}$$

The cost function is minimized when

$$
\begin{aligned}
2\Sigma_{p_i \in U}(b - p_i) &= 0 \\
\Sigma_{p_i \in U} b &= \Sigma_{p_i \in U} p_i \\
b &= \frac{1}{n} \Sigma_{p_i in U} p_i
\end{aligned}
$$

where $n = |U|$.

The update step also assigns each point to the nearest center, which itself minimizes the total cost because the cost of each point is minimized.

When $d > 1$, the update step in each cluster is independent from the other clusters. Thus, the sum of the costs of each clusters is also minimized when each individual cost is minimized.

EXERCISE 3

In this exercise, we show an example that the cosine distance does not follow the triangle inequality.

Let p, q be 2 vectors where the angle α between them is $\frac{\pi}{2}$, and r be the bisector of that angle. Thus, the angle β between p and r and the angle γ between r and q are both $\frac{\pi}{4}$. We have:

$$
\begin{aligned}
dist(p, q) &= 1 - cosine(\alpha) = 1 \\
dist(p, r) + dist(r, q) &= 1 - cosine(\beta) + 1 - cosine(\gamma) \\
&= 2\sqrt{(2)}/2 > 1
\end{aligned}
$$

This completes the example.

EXERCISE 4

In this exercise, we prove the triangle inequality property of Jaccard distance by using the minhash property. That is, the Jaccard distance equals the probability that two sets do not minhash to the same value.

Let's A, B, C be our 3 sets. After constructing the signature matrix, it is inferred from the lecture that:

$$
\begin{aligned}
d_j(A, B) &= P(minhash(A) \neq minhash(B)) = \frac{|minhash(A) \neq minhash(B)|}{|AllPermutations|} \\
d_j(A, C) &= P(minhash(A) \neq minhash(C)) = \frac{|minhash(A) \neq minhash(C)|}{|AllPermutations|} \\
d_j(C, B) &= P(minhash(C) \neq minhash(B)) = \frac{|minhash(C) \neq minhash(B)|}{|AllPermutations|}
\end{aligned}
$$

It is obvious that $minhash(C)$ can be $minhash(A)$, $minhash(B)$, or another value. Then, when $minhash(A) \neq minhash(B)$, either $minhash(C) \neq minhash(A)$, or $minhash(C) \neq minhash(B)$, or $minhash(C) \neq minhash(A) \neq minhash(B)$ is true.

Thus, $|minhash(A) \neq minhash(B)| \leq |minhash(A) \neq minhash(C)| + |minhash(C) \neq minhash(B)|$, which gives:

$$
\frac{|minhash(A) \neq minhash(B)|}{|AllPermutations|} \leq \frac{|minhash(A) \neq minhash(C)|}{|AllPermutations|} + \frac{|minhash(C) \neq minhash(B)|}{|AllPermutations|}
$$

This is equivalent to:

$$
P(minhash(A) \neq minhash(B)) \leq P(minhash(A) \neq minhash(C)) + P(minhash(C) \neq minhash(B))
$$

Which completes the proof.