# 2IMW30 - FOUNDATION OF DATA MINING
## Assignment 1a Report

Duy Pham (0980384)
Mazen Aly(——)

EXERCISE 1

In this assignment we implement the nearest neighbor classifier using the cosine distance to recognize handwritten digits. We use three techniques for this task, the first one is implementing everything from scratch, the second one is using scipy library just to measure the cosine similarity, and the third technique was using scikit learn Python library to for fitting, predicting and measuring the performance of the nearest neighbor classifier.

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 mnist 784 dataset that we have contains 50,000 training data points while the test data contains 10,000 data points.

The idea of nearest neighbor for our implementation is for every test data point that we have, we loop on all the training dataset and measure the cosine similarity which is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. And get the label of the closest neighbor to be the prediction of this test point.

We then build the confusion matrix of all the labels (digits form 0 to 9) and measures the accuracy as and the precision for each label as and the recall

```
# function that takes two vectors and return the dot product value of it by
# summing the multiplication of each two elements
def dot_product(v1, v2):
    return sum(map(lambda x: x[0] * x[1], zip(v1, v2)))

def cosine_similarity(v1, v2):
    prod = dot_product(v1, v2)
    # len and len2 are the magnitudes of the 2 vectors
    len1 = math.sqrt(dot_product(v1, v1))
    len2 = math.sqrt(dot_product(v2, v2))
    return prod / (len1 * len2)

#module for digits classification using nearest neighbor classifier
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
                        #we can replace this one by our function cosine_similarity(v1,v2)
                        sim = 1 - spatial.distance.cosine(training_entry_features, test_fea
                        # 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 are 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');

    # closing the files
    f_predicted.close()
    f_label.close()

# defining the confusion matrix
mtx = np.zeros((11, 11))
#printing options of numpy
np.set_printoptions(suppress=True)
#function to build the confusion matrix, rows are the predicted labels
#and columns are true labels
def build_matrix():
    f_predicted = open('predicted.txt', 'r')
    f_label = open('label.txt', 'r')
    for predicted_label in f_predicted:
        true_label = f_label.readline()
        mtx[int(predicted_label)][int(true_label)] += 1
    for i in range(0,10):
        mtx[i][10] = sum(mtx[i,:10])
        mtx[10][i] = sum(mtx[:10, i])
    mtx[10][10] =   sum(mtx[10,:10]) + sum(mtx[:10,10])
    return mtx

# function that takes the confusion matrix as input and measure the
# accuracy of the classifier, as well as the precision and recall for
# each class
def print_analytics(mtx):
    # the accuracy is measured by dividing the number of correct predictions by the total
    print("Accuracy of the NN classifier is : ", sum(mtx.diagonal()[:10]) * 100/len(test_d
)
    for label in range(0,10): # for each label we measure the precision and recall
        print("Class " , label , ":")
        # the precision is measured by dividing the true positives by the summation of tru
        print("Precision:" , mtx[label][label] / mtx[label][10])
        # the recall is measured by dividing the true positives by the summation of true p
        print("Recall:" , mtx[label][label] / mtx[10][label] )

training_data, validation_data, test_data  = load_data()
training_data, validation_data, test_data = list(training_data), list(validation_data), li
start_time = time.time()
# defining Scikit learn K neighbors classifier with metric cosine, and number of neighbors
#instead of the default 'auto' as this option works with the cosine metric
knn = KNeighborsClassifier(n_neighbors = 1 , metric = 'cosine' , algorithm='brute' )
# fit function takes the training feature as the first argument and training labels as seco
knn.fit(training_data[0] , training_data[1])
# getting the predicted labels for the test data.
Y_pred = knn.predict(test_data[0])
#measuring the accuracy
print(metrics.accuracy_score(test_data[1] ,Y_pred))
```

### EXERCISE 2

In this excercise, we are proving that if the Jaccard similarity of two sets is 0, then minhashing always gives a correct estimate of the Jaccard similarity.

If the Jaccard similarity of two sets is 0, then the intersection of the two set is empty. This means there are no rows which the elements are both "1" in the characteristic matrix. Hence, when performing minhashing,

there cannot be any row with the same values in 2 cells. Thus $P(h(S_i) = h(S_j)) = 0$, which is the same as the Jaccard similarity.

### EXERCISE 3

In this exercise, we show an example that the cyclic permutations are not enough to estimate the Jaccard similarity correctly.

Let's take a look at the example in table **??**. We will perform a cyclic permutation to estimate the Jaccard similarity.

Table 1: Example

| | S1 | S2 |
|---|---|---|
| a | 1 | 0 |
| b | 1 | 1 |
| c | 0 | 0 |
| d | 0 | 1 |
| e | 1 | 1 |

In this example, we can see that the 2 sets have 2 common words, "b" and "e". In general, $S1 \cap S2 = \{b, e\}$ and $S1 \cup S2 = \{a, b, d, e\}$. Thus the (correct) Jaccard similarity is $|S1 \cap S2|/|S1 \cup S2| = 1/2$.

Now we perform the cyclic permutation.

- a-b-c-d-e: The signature is 1 - 2.
- b-c-d-e-a: The signature is 1 - 1.
- c-d-e-a-b: The signature is 3 - 2.
- d-e-a-b-c: The signature is 2 - 1.
- e-a-b-c-d: The signature is 1 - 1.

We have 2 rows which contain the same values on 2 cells, and 5 rows in total. So the estimated Jaccard similarity is $2/5$, which is not correct. The reason is the $0 - 0$ row, which makes the next rows to be counted twice in the intersection (while every row is counted only once in the total number of permutations).

### 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:

$$d_j(A, B) = P(minhash(A) \neq minhash(B)) = \frac{|minhash(A) \neq minhash(B)|}{|AllPermutations|}$$

$$d_j(A, C) = P(minhash(A) \neq minhash(C)) = \frac{|minhash(A) \neq minhash(C)|}{|AllPermutations|}$$

$$d_j(C, B) = P(minhash(C) \neq minhash(B)) = \frac{|minhash(C) \neq minhash(B)|}{|AllPermutations|}$$

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.