Project 1: Image Classification Using KNN

Release date: 01/31
Due date: 02/09

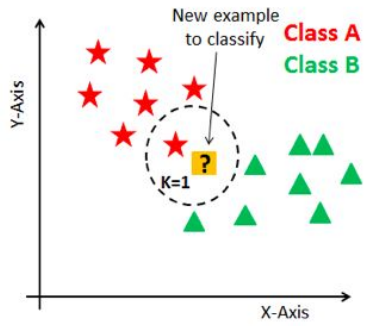
K-Nearest-Neighbors (KNN) Introduction

K-nearest neighbor algorithm is a majority vote between the K most similar (close) instances to a given "unseen" observation. For example, if we have two classes: Class A is images of cats and class B is images of dogs. Assume each image can be represented by two values (x,y). We can plot the images on a 2D plan as shown below.

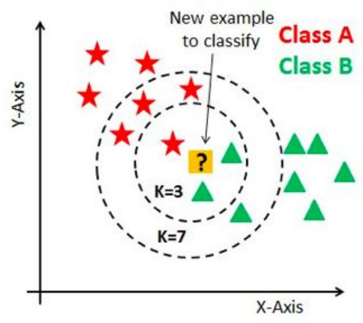
Now let's assume we have a new image (represented also by two values x,y) and we'd like to know which class this new images belongs to, is it an image of a cat or a dog?

One simple idea is to plot the new image on the same 2D plan with all the other training images we have and see which of the training images is closer to it. If it's closer to cat's images then it's a cat and if it closer to dog's images then it's a dog. This is exactly what KNN does. It calculates the distance between the new image with all training images, and take the most K close images and perform a majority voting over them. For example, if K=3 then we consider the top 3 closest images to our new image and classify it according to the majority class of these top 3 images. (i.e. if 2 or more images are dogs then we classify as a dog)

In the figure below, at K=1 then the new example is classified as Class A.



at K=3 then the new example is classified as Class B.
at K=7 then the new example is classified as Class A.



The Euclidean distance (d) between two image (vectors) p and q is:

$$p = (p_1, p_2, ...p_n)$$
$$q = (q_1, q_2, ...q_n)$$
$$d = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$$

Multiclass Evaluation and Confusion Matrix

In this project, you will implement the classification algorithm KNN on handwritten digits dataset. I.e. for a given image of a handwritten digit from 0 to 9. It's required to determine which digit this image represents. In other words, it's required to classify the image between 10 classes (0,1,2 … 9).

Let's assume we have a function that performs this classification, we need to assess how good this classification function. One way to measure how good our classification is by examining the *confusion matrix.* The confusion matrix is the generalization of the tp, fp, tn, fn of HW2 to many classes.

Below is a confusion matrix for the handwritten digits classification, for simplification we only show 3 classes. Each cell holds and integer represents the number of test images classified as row class while the true class is the column.
For example:
The test set have 100 digit-0 images, 50 digit-1 images and 40 digit-2 images.

- The number of digit-0 images in the test set that are classified as digit-0 is 75
- The number of digit-0 images in the test set that are classified as digit-1 is 20
- The number of digit-0 images in the test set that are classified as digit-2 is 5

- The number of digit-1 images in the test set that are classified as digit-0 is 7
- The number of digit-1 images in the test set that are classified as digit-1 is 30
- The number of digit-1 images in the test set that are classified as digit-2 is 13

- The number of digit-2 images in the test set that are classified as digit-0 is 3
- The number of digit-2 images in the test set that are classified as digit-1 is 2
- The number of digit-2 images in the test set that are classified as digit-2 is 35

		True Class		
		Digit 0	Digit 1	Digit 2
Predicted	Digit 0	75	7	3
	Digit 1	20	30	2
	Digit 2	5	13	35

Using the confusion matrix we can calculate a metric called "F1score" for each class which shows how good is our predictions for that class. Once we have an F1score per class we can calculate the mean of these scores to get a single global F1score that describes our predictions.

The F1score for a certain class is:

$$F1score = \frac{2tp}{2tp + fp + fn}$$

For class digit-0:

- tp = 75 (shown in green in the table below)
- fn = 20+5 (shown in red in the table below)
- fp = 7+3 (shown in grey in the table below)
- tn = 30+2+13+35 (shown in blue in the table below)

		True Class		
		Digit 0	Digit 1	Digit 2
Predicted	Digit 0	75	7	3
	Digit 1	20	30	2
	Digit 2	5	13	35

For class digit-1:

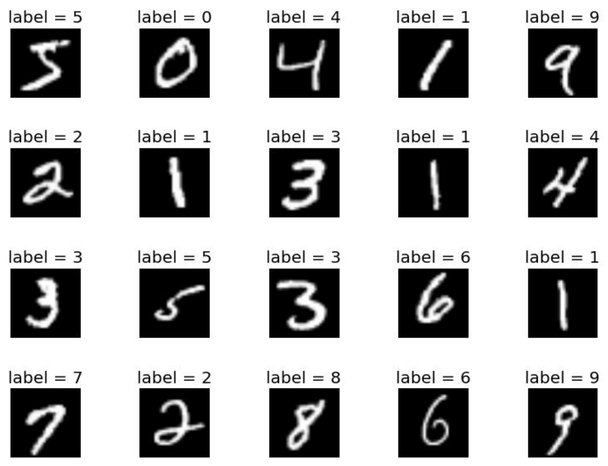
- tp = 30 (shown in green in the table below)
- fn = 7+13 (shown in red in the table below)
- fp = 20+2 (shown in grey in the table below)
- tn = 75+3+5+35 (shown in blue in the table below)

		True Class		
		Digit 0	Digit 1	Digit 2
Predicted	Digit 0	75	7	3
	Digit 1	20	30	2
	Digit 2	5	13	35

And so on for class digit-2.

Project Description

In this project, you will implement the classification algorithm KNN on hand written digits dataset called MNIST. Sample images is shown below.
The MNIST data consists of 20,000 examples of 28 × 28 images of digits (i.e., numbers from 0-9).



Since each image is a 28 × 28 matrix of pixels, we used T-SNE algorithm to reduce the dimensionality of each image to just 2 features (x, y). In this project you will be using the reduced version: **digits-embedding.csv**:

- **digits-embedding.csv**: contains a precomputed 2 features for each image
  (first column: image id, second column: class label, third and fourth columns: image features).

Design Tasks

You are required to implement KNN classification algorithm on the MNIST dataset. You are free to design your project the way you see fit in terms of functions, parameters…etc. However, your project should meet certain design constraints. You may ***NOT*** use ready implementation for KNN.

You must implement a Class called **MyKNN**. ***All*** your needed functions and attributes should be included in this class, such that a user using your class should be able to use it as follows:

```
knn = MyKNN() #creating an object

parsedData = knn.readData('digits-raw.csv') #reading, parsing the data.
predictions = knn.classify(5, parsedData, trainingIds, testIds) #classify
f1 = knn.evaluate(parsedData, testIds, predictions) #calculating the f1 score for the classification
```

As shown in the previous example, your class should implement those functions:

1. ***readData (filename)***
   Inputs:
   - Which takes a file name (digits-raw.csv or digits-embedding.csv) as an input then parse it.

   Returns:
   - The parsed data. It's up to you to decide what data structure should hold the data.

2. ***classify (k, parsedData, trainingIds, testIds)***
   Inputs:
   - ***k*** (int): the number of neighbors.
   - ***parsedData***: The dataset parsed using the "readData" function.
   - ***trainingIds*** (list<int>): A list containing the ids of images used as training samples.
   - ***testIds*** (list<int>): A list containing the ids of images used as testing samples.

   Returns:
   - ***predictions:*** The list of class labels (list<int>). Each entry in the list is the class label corresponds to each test sample.

3. ***evaluate (parsedData, testIds, predictions)***
   Inputs:
   - ***parsedData***: The dataset parsed using the "readData" function.
   - ***testData*** (list<int>): A list containing the ids of images used as testing samples.
   - ***predictions*** (list<int>): A list containing the class labels of the test samples calculated from the "classify" function.

   Returns:
   - ***confusionMat*** (2D numpy array): The confusion matrix of the predictions.
   - ***avgF1*** (float): returns the average F1score of predictions.

Reporting Tasks

Consider three versions of the data for each of the tasks below:

(i) Use the full dataset digits-embedding.csv,

(ii) Use only the subset of the data (digits-embedding.csv) consisting of the digits 1, 9, 8 and 5.

(iii) Use only the subset of the data (digits-embedding.csv) consisting of the digits 1 and 9.

For each dataset configuration (i, ii, iii), it's required

1. Visualize the images using their 2D features from (**digits-embedding.csv**), coloring the points to show their corresponding class labels. I.e. all images of the digit 1 colored in black, images of digit 2 colored in blue … and so on.

2. Classify the test data with different values of k $\in$ [1, 5, 15, 31,q]. Where q is all the samples of the training data.

3. For each k report the confusion matrix and the average F1score.

4. Construct a plot showing the average F1score on y-axis as a function of k on the x-axis.

Turning in Your Work

- Submit your completed file (yourEmail_project1.py) on Blackboard.
- You are required to submit a document (yourEmail_report.pdf) to report the previous plots, results and visualization.

Rubric

Project 1	
Task 1: Reading Data	**10%**
Task 2: Implementation of KNN	**30%**
Task 3: Evaluation	**25%**
Task 4: Successfully running experiments	**20%**
Task 6: Report	**15%**
TOTAL	**100%**