Big Assignment
Building a Simple Nearest-Neighbor Classifier for Image Datasets

Motivation

Here is a commonly encountered **classification task** - given an image, answer which class does it belong to. We are trying to apply our knowledge of data structures to solve it. Here is the formal definition of the problem:

Consider a dataset S with many objects -- say, the *MNIST*[1] *dataset which has many images corresponding to each of the handwritten digits (0-9).*

For each object s \in S, there is a label from a label-set L. (|L| << |S|)

In MNIST there are |L|*=10 classes in total, with labels "0","1",....,"9" respectively. Each image in the MNIST dataset is associated with exactly one of these ten class labels.*

Now, if a new object, let's call it **q**, is given to the classifier, the goal of the classifier is to assign the most likely class label to **q** based on the information available to it from S.

If a new image of a handwritten digit is given to the classifier on MNIST, it assigns it the most likely class label from "0" -- "9", using the information in the MNIST dataset.

k-NN classifier[2] [3]

There are very many classifiers for the above described task. We are interested in building a very simple yet very powerful classifier -- viz., **a k-Nearest Neighbor classifier (k-NN)**.
How does k-NN classifier work? It has two ingredients:

1) **a distance function, d,** which takes any two images and computes the "distance" between them using some metric function (more details below); and
2) an efficient way to compute the **k objects** in **S** that are **closest** (based on d) to **q**.

---

[1] http://yann.lecun.com/exdb/mnist/
[2] http://courses.cs.tamu.edu/rgutier/cs790_w02/l8.pdf
[3] https://www.cs.toronto.edu/~urtasun/courses/CSC411_Fall16/05_nn.pdf

A trivial solution is to first compute d(s, q) for each s \in S, sort the objects based on their distance from q, and select the k objects that have the smallest d(s,q). The complexity of this method is **$O(|S|.w_d)$**, where w_d is the work that needs to be done for each computation of d.

Once you have these **k** objects, the classifier simply takes the majority of the class labels of these objects (i.e., class labels that are most frequent, breaking ties arbitrarily), and returns that as the class label of the new object q.

We assume k = 1 in our assignment.

Efficient k-NN

Naturally, we are not satisfied with the trivial solution of having to compute d with all objects in our dataset for every q. It will not work as the size of the set S becomes large, and also when we receive a large number of objects that we need to classify (i.e., large number of q). Let us make use of our penchant for data structures, and *preprocess* the set S into a data structure so that we can reach our favorite O(log |S|) complexity for each q. At the heart of this is building a tree data structure over objects in S, for a given distance function d, so that the nearest neighbor to any q can be efficiently found.

> **Note***: if we had only objects of 1-dimension (i.e., just numbers), and a simple distance function (i.e., difference between two numbers), we can simply use our Binary search tree. Alas, in most uses of k-NN classifiers the objects are large dimensional objects, and the distance function is also complex.*

A standard way to compute a nearest neighbor efficiently is through KD-Trees[4]. However, there are limitations of KD-Trees due to their axis parallel style of separating an n-dimensional space. (*There are other limitations also such as their implementation code being readily available which precludes them from the purview of this assignment*) Hence, in this assignment we would like to ask you to implement a different data structure for the same job, called VP-Trees. You may find the details here: http://web.cs.iastate.edu/~honavar/nndatastructures.pdf. As part of the assignment we would like you to **implement vp-s trees as described in Section 3.2 of the paper.** Your implementation of the tree must be able to find the nearest neighbor of a given image from MNIST data. **We will use Euclidean distance as the distance between two images**. Note that an image can be represented as a 2D-array of scalars in the range [0,255].

Datasets

As we hinted earlier, we will use image datasets in our work. Specifically, we will use MNIST dataset. Each image is represented using a (m x n) byte values matrix. Each image also has an

[4] https://courses.cs.washington.edu/courses/cse373/02au/lectures/lecture22l.pdf

associated “class label” -- a categorical label which we assume is simply represented as an integer from a small domain. In MNIST, m = n = 28 pixels. (In CIFAR-10 it is 32 x 32 size).

File Format for MNIST

The data is stored in a very simple file format designed for storing vectors and multidimensional matrices. All the integers in the files are stored in the MSB first (high endian) format used by most non-Intel processors. Users of Intel processors and other low-endian machines must flip the bytes of the header.

There are 2 files:

train_images: training set images

train_labels: training set labels

Assignment

You are given a 10% sample of MNIST dataset. Full dataset is also available freely from (http://yann.lecun.com/exdb/mnist/), although not required.

You have to implement a **Java generic class** called VPTree that takes as arguments

(a) a collection of objects of any given type, T,

(b) a metric class implementing the interface `DistanceFunction <T> { double distance (T one, T two);}`. You will use Euclidean (L2) distance in this assignment.

```
class VPTree <T> {
    VPTree (Collection <T> collection, DistanceFunction d);
    T findOneNN (T q); //should print root-leaf pivot?
    void printTree (); //print the root, pivot, left subtree and right
subtree
}
class OneNNClassifier <T> {
      VPTree <T> index;
      void train (Collection <T> trainingSet);
      void test (Collection <T> testSet);
      void printAccuracy ();
}
```

And constructs a vp-tree index over it, using the images included in the train set of MNIST dataset: http://yann.lecun.com/exdb/mnist/

It should have a kNN function **T kNN (T q)** that takes as input an object and returns the k-nearest neighbor from the collection.

For evaluation we will be taking random images from the test set of the MNIST and check if your algorithm returns the correct image as the nearest neighbor. In addition, we will also verify the construction of VPTree.

Bonus (10%)

Implement other distance metrics:

1. L1 distance (https://en.wikipedia.org/wiki/Taxicab_geometry)
2. L∞ distance (https://en.wikipedia.org/wiki/Uniform_norm)