CSE 417T: Homework 5

Due: 8th of Nov 2016 – 10am

Notes:

- There are 3 problems on 6 pages in this homework.
- This homework consists of two parts.
- **Implementation part:** Problem 1(c) and 3 (you may work in groups of 2)
  - You may work in pairs on the **implementation part**. Each group should turn in the source code only once. Please do not collaborate with anyone other than your partner. Indicate group work by adding both wustlkeys to the `partners.txt` file and commit this file to both repositories (partnerships must me mutually accepted!).
  - Stubs for your implementations are in your SVN repository. Your implementations will be autograded. **Follow the submission instructions exactly.** Do **not** change the names or signatures of the functions as this will lead the autograder to fail in which case you cannot get credit!
  - Comment your code properly.
- **Written part:** Problem 1(a-b) and 2 (**no** group work)
  - Submit all written answers by committing a single pdf file named **`YOUR_WUSTL_KEY_hw5.pdf`** to the hw5 folder in your SVN repository.

Autograder: The autograder for this class is used for grading only. It is not meant to be used for debugging. We will run it after the submission deadline. Since this is the first time that your work will be autograded, we offer an *autograder deadline* prior to the submission deadline, where we grade your work and give you the feedback of the autograder. This deadline is optional, but highly recommended!

- **THU 3rd of Nov 10am: deadline for optional autograder feedback**
- **TUE 8th of Nov 10am: submission deadline**
- THU 10th of Nov 10am: deadline after automatic extension (**no late days** after this deadline!)

If you have any questions with respect to the autograder, ask on Piazza using the **autograder tag**.

Problems:

1. **(30 points) L2-distances through Matrix Operations**
   Many machine learning algorithms access their input data primarily through pairwise distances. It is therefore important that we have a fast function that computes pairwise (Euclidean) distances of input vectors. Assume we have n data vectors $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and m vectors $\vec{z}_1, \ldots, z_m \in \mathbb{R}^d$. And let us define two matrices $X = [\vec{x}_1, \ldots, \vec{x}_n] \in \mathbb{R}^{d \times n}$, where the i^{th} column is a vector $\vec{x}_i$ and similarly $Z = [\vec{z}_1, \ldots, \vec{z}_m]$. Our distance function takes as input the two matrices X and Z and outputs a matrix $D \in \mathbb{R}^{n \times m}$, where

$$D_{ij} = \sqrt{(\vec{x}_i - \vec{z}_j)^\top (\vec{x}_i - \vec{z}_j)}.$$

   The key to efficient programming in Matlab and machine learning in general is to think about it in terms of mathematics, and not in terms of loops.

   (a) Show that the Gram matrix G (aka inner-product matrix) with

$$G_{ij} = \vec{x}_i^\top \vec{z}_j$$

   can be expressed in terms of a pure matrix multiplication.

   (b) Let us define two new matrices $S, R \in \mathbb{R}^{n \times m}$

$$S_{ij} = \vec{x}_i^\top \vec{x}_i, \;\; R_{ij} = \vec{z}_j^\top \vec{z}_j.$$

   Show that the squared-euclidean distance matrix $D^2 \in \mathbb{R}^{n \times m}$, defined as

$$D^2_{ij} = (\vec{x}_i - \vec{z}_j)^2,$$

   can be expressed as a linear combination of the matrix S, G, R. (Hint: It might help to first express D^2_{ij} in terms of inner-products.) Further, think about and answer the following questions:

   - What mathematical property do all entries of D satisfy?
   - What are the diagonal entries of D assuming that $X = Z$ (we want the distance among all data points in X)?
   - What do you need to do to obtain the true Euclidean distance matrix D?

   Remember the answers to all of these questions and ensure that your implementation in the next part satisfies all of them.

   (c) Remember the slow function `l2distanceSlow.m` using nested for loops to compute the L2-distance. You find it in the `hw5` folder in your SVN repository. Read through the code carefully and make sure you understand it. It is perfectly correct and will produce the correct result ... eventually. To see what is wrong, run the following program in the MATLAB console:

```
>> X=rand(100,10000);
>> Z=rand(100,2000);
>> tic;D=l2distanceSlow(X,Z);toc
```

   This code defines some random data in X and Z and computes the corresponding distance matrix D. The `tic, toc` statements time how long this takes. On my laptop it

takes over a minute. This is way too slow! If I were to compute the distances between larger training and testing sets or higher dimensional data points (imagine for instance images with millions of pixels), it would take days! The problem is that the distance function uses a programming style that is well suited for C++ or Java, but not MATLAB!

Implement the function `l2distance.m` (a stub file is in your SVN repository), which computes the Euclidean distance matrix D without a single loop. Remember all your answers to the previous parts of this problem and test your implementation to make sure ot incorporates all of those. Once you are sure that your implementation is correct, time the distance function again:

```
>> X=rand(100,10000);
>> Z=rand(100,2000);
>> tic;D=l2distance(X,Z);toc
```

How much faster is your code now? With your new implementation you should easily be able to compute the distances between many more vectors. Call the function `l2dist_tictoc` to see how many distance computations you can perform within one second. With the loopy implementations, the same computation might have taken you several days.

Submit your implementation by committing your function **`l2distance.m`**, and the **`partners.txt`** file to the hw5 folder in your SVN repository.

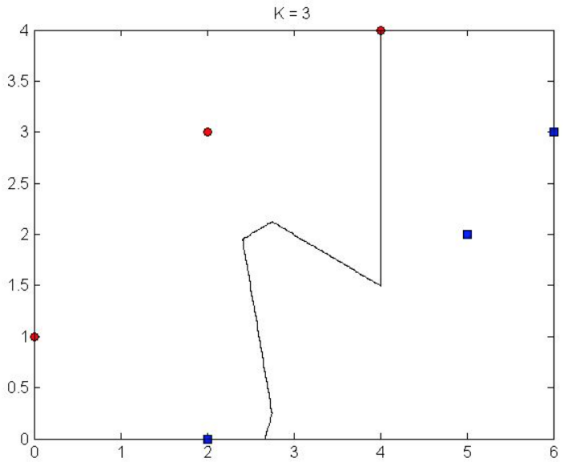
2. **(20 points)** k**-Nearest-Neighbor**
In this problem, you are going to look at a small dataset to understand various properties of k-NN better. Suppose there is a set of points on a two-dimensional plane from two different classes. Below are the coordinates of all points.

Points in class **red:** $(0, 1)$, $(2, 3)$, $(4, 4)$
Points in class **blue:** $(2, 0)$, $(5, 2)$, $(6, 3)$

(a) Draw the k-nearest-neighbor decision boundary for $k = 1$. Remember that the decision boundary is defined as the line where the classification of a test point changes. Use the standard Euclidean distance between points to determine the nearest neighbors. Start by plotting the points as a two-dimensional graph. Please use the corresponding colors for points of each class (e.g, **blue and red**).

(b) If the y-coordinate of each point was multiplied by 5, what would happen to the $k = 1$ boundary (draw another picture)? Explain in at most two sentences how this effect might cause problems when working with real data.

(c) The k-NN decision boundary for k = 3 is shown as below. Suppose now we have a test point at $(1, 2)$. How would it be classified under 3-NN? Given that you can modify the 3-NN decision boundary by adding points to the training set in the diagram, what is the minimum number of points that you need to add to change the classification at $(1, 2)$? Show also where you need to add these points.



(d) What is the testing complexity for one instance, e.g., how long does it take for k-NN to classify one point? Assume your data has dimensionality d, you have n training examples and use Euclidean distance. Assume also that you use a quick select implementation which gives you the k smallest elements of a list of length m in $O(m)$.

Suggest two strategies to decrease the testing complexities. Provide the *average* and *worst case* testing complexities for both strategies. At what cost are you gaining this increased efficiency?

3. **(50 points) Build a k-Nearest Neighbor Classifier**
In this project, you will build a k-NN classifier for handwritten digits classification and face recognition. The data for the experiments resides in the files `faces.mat` and `digits.mat` (download them from Piazza resources; do **NOT** add those to your SVN repositories).

Data description:
`xTr` are the training vectors with labels `yTr`.
`xTe` are the testing vectors with labels `yTe`.

As a reminder, to predict the label or class of an image in `xTe`, we will look for the k-nearest neighbors in `xTr` and predict a label based on their labels in `yTr`. For evaluation, we will compare these labels against the true labels provided in `yTe`.

Data visualization:
You can visualize one of the faces by running

```
>> load faces
>> figure(1);
>> clf;
>> imagesc(reshape(xTr(:,1),38,31));
>> colormap gray;
```

Note that the shape 38×31 is the size of the image. We convert it from a flat vector form in the dataset to a matrix which can be visualized. Note: If your images appear upside down,

use `imagesc(flipud(reshape(xTr(:,1),38,31)))`.

Implementation:
The following parts will walk you through the project. Your development will work out best, if you finish these functions in this order. **Note that your project will also be graded based on efficiency. So, avoid tight loops at all cost.** Stub files for all functions are in the `hw5` folder in your SVN repository.

(a) Implement the function `findknn.m`, which should find the k-nearest neighbors of a set of vectors within a given training data set based on Euclidean distance (Hint: use your `l2distance` function here). The call of

```
>> [I,D]=findknn(xTr,xTe,k);
```

should result in two matrices I and D, both of dimensions $k \times n$, where n is the number of input vectors in `xTe`. The matrix $I(i,j)$ is the index of the i^{th} nearest neighbor of the vector $xTe(:,j)$. So, for example, if we set `i=I(1,3)`, then `xTr(:,i)` is the first nearest neighbor of vector `xTe(:,3)`. The second matrix D returns the corresponding Euclidean distances. So, $D(i,j)$ is the distance of $xTe(:,j)$ to its i^{th} nearest neighbor.

(b) The function `analyze.m` should compute various evaluation metrics. The call of

```
>> result=analyze(kind,truth,preds);
```

should output the **accuracy** or **absolute loss** in variable `result`. The type of output required can be specified in the input argument `kind` as strings `"abs"` or `"acc"`. The input variables `truth` and `pred` should contain vectors of true and predicted labels respectively. For example, the call

```
>> analyze("acc",[1 2 1 2],[1 2 1 1])
```

should return an accuracy of 0.75. Here, the true labels are $1,2,1,2$ and the predicted labels are $1,2,1,1$. So the first three examples are classified correctly, and the last one is wrong $\Rightarrow 75\%$ accuracy.

Take a look at `knneval.m`. This script runs the (not yet implemented) k-nearest neighbor classifier over the faces and digits data set and uses `analyze` to compute the classification accuracy. In its current implementation the results are picked at random. Run it to test your `analyze.m` implementation. The faces data set has 40 classes, the digits data set 10. What classification accuracy would you expect from a random classifier?

(c) Implement the function `knnclassifier.m`, which should perform k-nearest neighbor classification on a given test data set. The call

```
>> preds=knnclassifier(xTr,yTr,xTe,k);
```

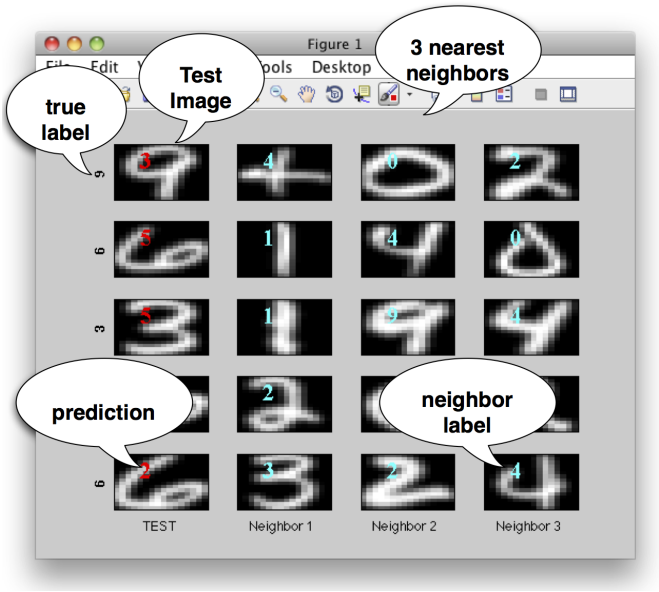
should result in the predictions for the data in `xTe`. I.e., $preds(i)$ will contain the prediction for $xTe(:,i)$. You can compute the actual classification accuracy (or error) on the test set by calling

```
>> analyze("acc",yTe,knnclassifier(xTr,yTr,xTe,3))
```

Test your function on both data sets with `knneval.m`. To visualize your classifications, you can also run

```
>> knnvisual faces
>> knnvisual digits
```

Hopefully you will get better results than the ones in the figure below (using the random classifier):



Shuffle the dimensions of the images in random order and verify that this does not change your classification error. You can use the command `randperm` and modify `knnvisual.m`.

(d) Sometimes a k-NN classifier can result in a draw, when the majority vote is not clearly defined. Can you improve your accuracy by falling back onto k-NN with lower k (say $k-1$) to break ties? Edit the file `knnclassifier.m` accordingly.

(e) Edit the function `competition.m`, which reads in a training and testing set and makes predictions. Inside this function you are free to use any combination or variation of the k-nearest neighbor classifier. Can you beat my implementation on our secret training and testing sets in terms of **accuracy**? (Hint: My accuracy is around 92% for `faces` and 95% for `digits`.)

Submit your implementations by committing your functions **`findknn.m`**, **`analyze.m`**, **`knnclassifier.m`**, **`competition.m`**, and **`partners.txt`** to the `hw5` folder in your SVN repository.