TI2736-C Datamining
Assignment 2: Classification

Delft University of Technology, February–February 2016

Thomas Abeel, Marcel Reinders
Zekeriya Erkin, Julian Kooij
Chantal Olieman, Gijs Weterings, Alex Salazar

Pattern Recognition and Bioinformatics Group

TU Delft

2 Classification

In the next few exercises you will be working on some classification algorithms; the Perceptron algorithm and the Nearest Neighbour algorithm. These algorithms are used to classify data to belong to some class, based on a training set.

Exercise 2.1. Perceptron

For this first assignment you will work on the Perceptron algorithm, with varying threshold value. Let $\mathbf{x}$ be d-dimensional a column vector with d features of an object,

$$\mathbf{x} = [x_1, x_2, ..., x_d]^\top \tag{1}$$

and $\mathbf{w}$ a corresponding d-dimensional column vector of feature weights in the Perceptron,

$$\mathbf{w} = [w_1, w_2, ..., w_d]^\top. \tag{2}$$

The Perceptron tests if a feature vector $\mathbf{x}$ belongs to the positive class by evaluating:

$$\mathbf{w}^\top \mathbf{x} \geq \theta \tag{3}$$

where θ is the threshold value. In order to treat θ similar to the feature weights w_i, we can replace the d-dimensional feature vectors $\mathbf{x}$ by $d+1$-dimensional vectors

$$\mathbf{x}' = [x_1, x_2, ..., x_d, -1]^\top, \tag{4}$$

and the weights vector $\mathbf{w}$ of the Perceptron by

$$\mathbf{w}' = [w_1, w_2, ..., w_d, \theta]^\top. \tag{5}$$

Question 0.1. Using $\mathbf{w}'$ and some object with feature values $\mathbf{x}'$, what is now the formula that needs to be evaluated to determine if the object belongs to the positive class?

Question 0.2. How can you determine on which side of the decision boundary, that is represented by the weights $\mathbf{w}'$, a certain feature vector is?

An implementation for the Perceptron algorithm is given in `Perceptron.java`. `PerceptronPlotter.java` can be used to visualize the data and the Perceptron decision boundary. The `Dataset` class can be used to read in datasets that are used in training the Perceptron classifier.

Step 1. Complete the `FeatureVector.product` and `FeatureVector.distance` methods. They should calculate the inner product and the Euclidean distance of two feature vectors respectively.

Step 2. Complete the `Perceptron.train` method. This method evaluates a feature vector and updates its weights if it is found to be misclassified. The weights are updated according to:

$$\mathbf{w}' \leftarrow \mathbf{w}' + \eta y \mathbf{x}' \tag{6}$$

Where η is the learning rate of the Perceptron algorithm, $\mathbf{w}'$ are the weights, $\mathbf{x}'$ is the feature vector that was misclassified and y its label.

Question 2.1. What could happen if the learning rate is too low? What could happen if the learning rate is too high?

Step 3. In `main.java` there is a `perceptron` method. Initialize in this method a `Perceptron`, `Dataset` and `PerceptronPlotter` object. Let the `Dataset` object read the data with filename `"data/gaussian.txt"` and set `addThresh` to `true`. Next, initialize a Perceptron object with a learning rate of 1. Update the weights of the Perceptron and visualize it using the `PerceptronPlotter` object. There is no need to iterate the updating, the algorithm already achieves a steady state solution after one round.

Question 3.1. The yellow line indicates the vector represented by $\mathbf{w}$. The black line represents the decision boundary. How are the two related?

Question 3.2. What equation belongs to the decision boundary? Write it down.

Step 4. In `main.java` there is a `perceptronDigits` method. Initialize a `Dataset` object, let it read the data with filename `"data/train_digits.txt"` and set `addThresh` to `true`. This dataset contains handwritten digits as 8x8 grayscale images (pixel values). Use `DigitFrame` to display some images from the dataset.

Question 4.1. What is contained in the dataset? How many images of each class are there?

Step 5. Similar to before, initialize a `Perceptron` object. Update the weights twice this time, the algorithm does not get a steady state solution after just one iteration.

Step 6. The weights can also be visualized, just like the images themselves. Display the weights using `DigitFrame`.

Question 6.1. Explain the shape of the image representation of the weights. How does this distinguish the different classes from one another? Why are some parts dark and other parts light?

Step 7. Create another `Dataset` object and let it read the data with filename `"data/test_digits.txt"`. This dataset is intended to test the previously trained Perceptron classifier. Predict the label of each test object, count the number of misclassified objects and divide that by the total number of test subjects to compute the error rate.

Question 7.1. Why do we need a separate dataset to test the classifier? What would happen if we tested our classifier on our training set?

Step 8. Visualize the digits which the Perceptron classifier wrongly classifies.

Question 8.1. Why do you think these were misclassified?

Exercise 2.2. Nearest Neighbour

Next we will work on a nearest neighbour classifier. When classifying an object, this classifier selects k of its nearest neighbours and calculates the majority vote using some kernel function. The simplest of this type of classifiers just assigns the label of the object to the label of its nearest neighbour.

Step 1. In `NearestNeighbour.java`, implement the `predict` method. The steps of this method are as follows:

1. Calculate the distance of the object to each element in the dataset.

2. Sort them by distance, smallest distances first.
3. Select k of the nearest objects.
4. Calculate which label has the highest majority vote.
5. Assign the label to that class.

Hint: For measuring and sorting the measurements, use a list of `Measurement` objects. This list can then easily be sorted using `Collections.sort`.

Note: There are quicker ways of getting k nearest neighbours, but for the purpose of this exercise this is not important.

Question 1.1. Imagine we have one point of class a at the origin and two points of class b near infinity. What would happen if we want to predict a point near the origin if we set k to 3? How could we fix this?

Step 2. In `main.java`, implement the `nearestNeighbour` method. This method should create a `NearestNeighbour` and `NearestNeighbourPlotter` object with k set to 3. Let the `NearestNeighbour` object read in the data from the file `"data/banana.txt"`. Let the `NearestNeighbourPlotter` plot the data of the `NearestNeighbour` classifier.

Note: You can click on the scatterplot to add points. These points will then be classified and shown on screen. Added points will not contribute to further classifications.

Step 3. In `main.java`, implement the `nearestNeighbourDigits` method. This method trains a nearest neighbour classifier on the same dataset of digits as before with the perceptron. Again, calculate the error rate on the *test* set and compare the results.

Question 3.1. What are the advantages/disadvantages of the nearest neighbour classifier?