(67577) INTRODUCTION TO MACHINE LEARNING

Problem Set 3 – Classification using coordinate classifiers

Due: Monday 20.4.2015 12:00

This exercise has two goals:

- Practice learning using the ERM rule with respect to some (very naive) hypothesis class.
- Introduction to two datasets that we will use throughout the course.

Do not be intimidated by the length of this document! Its purpose is to give a detailed description of the actions you must take when you use machine learning. Most of the code was written by the course stuff, and is given to you (the school solution consists of only 16 lines on top of what you've got).

1 The datasets

- **MNIST:** This is a collection of images of digits between 0 to 9 (see figure 1). Each image is gray scale with pixel-values ranging from 0 (black) to 255 (white), and of size 28×28. The corresponding rule we want to learn is to classify images to the digit appearing in them. The dataset consists of a training set of size 60000 images and a test set of size 10000. In this exercise we will restrict ourselves to the digits 4 and 7, corresponding to the labels 1 and 0.

Figure 1: Some images from MNIST

- **SPAM:** This dataset contains SMSs, some of which are spam, and some are real messages (see "SMSSpamCollection.txt"). The goal is to learn a rule that classifies SMSs as spam or not.

2 The hypothesis class

We will use essentially the same hypothesis class for both problems (the only difference will be the number of parameters, n). The instance space will be $X = \{0,1\}^n$ and the label space will be $Y = \{0,1\}$. For every coordinate $0 \leqslant i < n$ we will have a hypothesis $h_i : X \to Y$ defined as $h_i(x) = x_i$. Finally, we will take $\mathcal{H}_n = \{h_i \mid 0 \leqslant i < n\}$.

This class is highly naive and not expressive – that is, it is very unlikely that a hypothesis from this class will have a very small error (at least on real-world problems). On the other hand, $\mathcal{H}_n$ will allow for a very simple and fast implementation of the ERM algorithm.

3 Implementing ERM

Your first task is to implement first three functions in the file "ex3.py":

- **zeroOneLoss(coordinate, data, labels).** This function computes the empirical zero-one loss of the classifier corresponding the the given coordinate.
- **allZeroOneLosses(data, labels).** This function computes the empirical zero-one losses of all classifiers in $\mathcal{H}$.
- **ERM(data, labels).** This function finds the ERM.

The exact form of the input and the output is explained in the attached file "ex3.py".

4 Learning using the MNIST dataset

Now that we have implemented the ERM rule, we would like to use it. The first task we will tackle is to learn a rule for digit recognition using the MNIST dataset. To do so, we will need to take two steps.

1. **Preparing the data.** Each image in the MNIST dataset is represented by a vector with 784 entries, where the $(x, y) \in \{0, \ldots, 27\}$

pixel corresponds to the $(x \cdot 28 + y)$'th entry in the vector representing the image. Each entry consists of a number between 0 (black) to 255 (white) that represents the color of the corresponding pixel. However, the classifiers in our class take as input a zero-one vector! Therefore, in order to use our algorithm we will have to do some preprocessing. What we will do is to turn each coordinate to 0 if it is black (concretely, has gray-value less than 10) and to 0 if it is not (concretely, has gray-value at least 10). We wrote the code that reads the data and makes this preprocessing in the function "prepareData()" in the file "MNIST.py".

2. **Find the ERM.** After the preprocessing we are ready to use our algorithm on MNIST. The function "findERM(data, labels)" in the file "MNIST.py" finds the ERM using the code in "ex3.py", and prints the pixel defining the ERM. The function "visualizeLosses(data, labels)" gives a much more detailed description of the losses of the different hypotheses. It produces a 28×28 image that describes the error of the different hypotheses - the brightness of each pixel is the error of the corresponding hypothesis.

As you can tell, we wrote all the code relating to this part. All you have to do is to execute "MNIST.py". If there is no mistake in your code from the previous part, you should see a nice image. Note the following:

- Can you recognize the digit 4 and 7 in the image?
- The hypotheses in our class are radically simple – the hypothesis corresponding to the pixel p labels an image as 4 if and only if this pixel is not black. Yet, some of these hypotheses have non-trivial performance.

5 Implementing Bag of words, and learning using the SPAM dataset

The second task we will tackle is to learn a rule for spam detection using the SPAM dataset. As with MNIST, we will do this in two steps.

1. **Preparing the data.** Each example in the SPAM dataset is a line of text. As with MNIST, we will have to make some preprocessing in order to transform each example to a zero-one vector. We will use the bag-of-words representation taught in the Tirgul (we will use the 0-1 version of it, where the coordinate corresponding to a given word is 1

if the word appears in the sentence and 0 otherwise). In the function "prepareData()" in the file "SPAM.py" we wrote a code the extracts a dictionary consisting of all the words that appear in the dataset.

Your role is to fill the missing code in "vectorizeText(text, D, num_samples)" in "ex3.py". This function takes the dataset, the dictionary and the number of samples as input, and return a bag of words representation of the dataset.

2. **Find the ERM.** The function "findTenBestClassifiers(all_words, data, labels)" in the file "SPAM.py" finds and prints the ten words defining the ten best classifiers.

After you wrote the code of "prepareData()", execute "SPAM.py", and take a look on the top ten classifiers in the class. Note again that the hypotheses in our class are radically simple – the hypothesis corresponding to the word w labels a message as spam if and only if w appears in the message. Yet, some of these hypotheses have non-trivial performance.

6 Some general guidelines

- You don't need to check the validity of the input of the functions you implement.
- You may find the following links useful:
  - `http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html`
  - `http://docs.scipy.org/doc/numpy/user/basics.indexing.html#boolean-or-mask-index-arrays`
  - `http://docs.scipy.org/doc/numpy/reference/generated/numpy.nonzero.html`
  - `http://en.wikipedia.org/wiki/Bag-of-words_model#Example_implementation`
  - `http://docs.scipy.org/doc/scipy/reference/sparse.html`
  - `http://docs.scipy.org/doc/numpy/reference/generated/numpy.zeros.html`
  - `http://docs.scipy.org/doc/numpy/reference/generated/numpy.ones.html`
  - `http://docs.scipy.org/doc/numpy/reference/generated/numpy.sum.html`

7 What to submit?

Upload to the course website a zip file named "ex3.zip" that contains the following files:

- "ex3.py"
- The image you obtained after running "MNIST.py".
- A text file named "RESULTS" with the pixel corresponding to the best classifier and the list of the 10 words corresponding to the ten best classifiers.

A small tester

The file "ex3Test.py" tests the error of the classifier that your algorithm outputs. Make sure that you pass it before submitting!