COMS 4771 HW3

Due: Thu Jun 18, 2015 (written);
Sun Jun 21, 2015 (programming)

A printed copy of the written part of the homework is due at 5:30pm (Thu) in class. You must show your work to receive full credit. Email the programming part to the TA by end of day (Sun).

Special note: To ease the burden, the programming part (Q4) can be done in groups of at most two people (please submit one tar/zip file per group which includes the names/uni of the group members in a README). The written part must be done individually.

1 **[Low-dimensional information-preserving transformations]** (*hashing the cube*) You have a collection of nonzero distinct binary vectors $x_1, \ldots, x_m \in \{0,1\}^n$. To facilitate later lookup, you decide to hash them to vectors of length $p < n$ by means of a linear mapping $x_i \mapsto Ax_i$, where A is a $p \times n$ matrix with 0-1 entries, and all computations are performed modulo 2. Suppose the entries of the matrix are picked uniformly at random (ie, each an independent coin toss)

(i) Pick any $1 \leq i \leq m$, and any $b \in \{0,1\}^p$. Show that the probability (over the choice of A) that x_i hashes to b is exactly $1/2^p$. (Hint: focus on a coordinate $1 \leq j \leq n$ for which $x_{ij} = 1$.)

(ii) Pick any $1 \leq i < j \leq m$. What is the probability that x_i and x_j hash to the same vector? This is called a *collision*.

(iii) Show that if $p \geq 2\log_2 m$, then with probability at least $1/2$, there are no collisions among the x_i. Thus: to avoid collisions, it is enough to linearly hash into $O(\log m)$ dimensions!

(question credit: Prof. Sanjoy Dasgupta)

2 **[PAC learning and VC dimension]**

(i) Compute the tightest possible VC dimension estimate of the following model classes:
- $(\mathcal{F}_1 \cup \mathcal{F}_2)$, where each $\mathcal{F}_i := \{f \in \mathcal{F}_i | f \to \{0,1\}\}$.
- $\mathcal{F} :=$ Convex polygons in $\mathbb{R}^2$.

(ii) Given a collection of models $\mathcal{F}$, suppose you were able to develop an algorithm $\mathcal{A} : (x_i, y_i)_{i=1}^n \mapsto f_n^{\mathcal{A}}$ (that is, given n labeled training samples, $\mathcal{A}$ returns a model $f_n^{\mathcal{A}} \in \mathcal{F}$) that has the following property: for all $\epsilon > 0$, with probability 0.55 (over the draw of $n = O(\frac{1}{\epsilon^2})$ samples,

$$\text{err}(f_n^{\mathcal{A}}) - \inf_{f \in \mathcal{F}} \text{err}(f) \leq \epsilon,$$

where $\text{err}(f) := P_{(x,y)}[f(x) \neq y]$.

Show that one can construct an algorithm $\mathcal{B} : (x_i, y_i)_{i=1}^{n'} \mapsto f_{n'}^{\mathcal{B}}$ with the property: for all $\epsilon > 0$ and all $\delta > 0$, with probability at least $1 - \delta$ over a draw of n' samples:

$$\text{err}(f_{n'}^{\mathcal{B}}) - \inf_{f \in \mathcal{F}} \text{err}(f) \leq \epsilon.$$

Moreover show that $n' = \text{poly}(\frac{1}{\epsilon}, \frac{1}{\delta})$ samples are enough for the algorithm $\mathcal{B}$ to return such a model $f_{n'}^{\mathcal{B}}$. Hence, the model class $\mathcal{F}$ is *efficiently* PAC-learnable.
(Hint: Algorithm $\mathcal{B}$ can make multiple calls to the algorithm $\mathcal{A}$.)

3 **[Bayesian interpretation of ridge regression]** Consider the following data generating process for linear regression problem in $\mathbb{R}^d$. Nature first selects d weight coefficients $w_1, \ldots, w_d$ as $w_i \sim N(0, \tau^2)$ i.i.d. Given n examples $x_1, \ldots, x_n \in \mathbb{R}^d$, nature generates the output variable y_i as

$$y_i = \sum_{j=1}^{d} w_j x_{1,j} + \epsilon_i,$$

where $\epsilon_i \sim N(0, \sigma^2)$ i.i.d.

Show that finding the coefficients $w_1, \ldots, w_n$ that maximizes $P[w_1, \ldots, w_d | (x_1, y_1) \ldots, (x_n, y_n)]$ is equivalent to minimizing the ridge optimization criterion.

4 **[Spam classification via Logistic Regression]** Download the datafile hw3data.tar.gz from courseworks. This datafile contains email data of around 5,000 emails divided in two folders 'ham' and 'spam' (there are about 3,500 emails in the 'ham' folder, and 1,500 emails in the 'spam' folder). Each email is a separate text file in these folders. These emails have been slightly preprocessed to remove meta-data information.

(i) (Embedding text data in Euclidean space) The first challenge you face is how to systematically embed text data in a Euclidean space. It turns out that one successful way of transforming text data into vectors is via "Bag-of-words" model. Basically, given a dictionary of all possible words in some order, each text document can be represented as a word count vector of how often each word from the dictionary occurs in that document.

Example: suppose our dictionary D with vocabulary size 10 ($|D| = 10$). The words (ordered in say alphabetical order) are:
1: also
2: football
3: games
4: john
5: likes
6: Mary
7: movies
8: to
9: too
10: watch
Then any text document created using this vocabulary can be embedding in $\mathbb{R}^|D|$ by counting how often each word appears in the text document.

Say, an example text document t is:
`John likes to watch football. Mary likes movies.`
Then the corresponding word count vector in $|D| = 10$ dimensions is:
`[ 0 1 0 1 2 1 1 1 0 1]`
(because the word "also" occurs 0 times, "football" occurs 1 time, etc. in the document.)

While such an embedding is extremely useful, a severe drawback of such an embedding is that it treats similar meaning words (e.g. watch, watches, watched, watching, etc.) independently as separate coordinates. To overcome this issue one should preprocess the entire corpus to remove the common trailing forms (such as "ing", "ed", "es", etc.) and get only the root word. This is called word-stemming.

Your first task is to embed the given email data in a Euclidean space by: first performing word stemming, and then applying the bag-of-words model.

Some useful references:

- Bag-of-words: http://en.wikipedia.org/wiki/Bag-of-words_model
- Word stemming: http://en.wikipedia.org/wiki/Stemming

(ii) Once you have a nice Euclidean representation of the email data. Your next task is to develop a logistic regression classifier to classify new emails as `spam` or `not-spam`.

Deliverables: To test your code, you must provide a function called:
Function name: `classify_new_email`
Input parameter: a string – indicating the filename containing the email.
Output: an integer – either 0 or 1 (0 indicating `not-spam`, and 1 indicating `spam`)

As you are working in a programming language of your choice, you should also provide a `README.txt` file detailing how one can invoke your classification function.

The top three best performing classifiers (on our unreleased test dataset) will get extra points!