COS 424
Homework #2
Due Tuesday, March 9th

See the course website for important information about collaboration and late policies, as well as where and when to turn in assignments.

Question 1 – Classification with rejection

Consider a classification problem with inputs x and classes y. Assuming that the conditional probability distribution of the classes given the inputs is known, the Bayes optimal decision rule is

$$f_{\text{Bayes}}(x) = \arg\max_y \mathbb{P}\{Y = y | X = x\} \ .$$

The classifier will be used in a situation where it is less expensive to reject an input than to return an erroneous class. Consider for instance a machine that reads zip code on envelopes. Asking an operator to examine the ambiguous pieces of mail can be considerably cheaper than shipping them to the wrong place.

A classifier able to reject is simply a classifier with an additional class `reject`. Although this extra class is always incorrect, it could be interesting to output it because the associated cost is smaller than the cost of an ordinary misclassification.

We assume that a correct classification costs 0, a misclassification costs 1, and a rejection costs $c < 1$. We are looking for a new optimal decision rule that tells us when to reject an input x, and, when we are not rejecting, which class to output.

Express the new optimal decision rule as a function of c and $\mathbb{P}\{Y|X\}$.

How did you reach this result?

Question 2 – Preprocessing the Reuters21578 dataset

Get the following files from `http://www.cs.princeton.edu/courses/archive/spring10/cos424/w/hw2`:

- `reuters21578.tar.gz` is an archive containing manually labeleled newswire stories. Please read carefully the included `README` file. We are interested in the five topic labels `earn`, `acq`, `crude`, `grain`, and `trade`. We want to build five classifiers indicating whether a particular story is associated with each of these five topics.
- `porter_stemmer.c` is the C source code of the *Porter stemmer*, a program that takes an english text and transforms all words into words stems by removing suffixes such as "–ing", "–ication", etc.
- `stopwords.txt` is a list of english words that are so common that they say very little about the topic of the documents that contain them.

First we have to split the data into a training set and a test set. We will use the *Modapte split* that is described in the `README` file: 9603 training stories and 3299 testing stories. Our goal in this problem is to encode each story j as a vector $\mathbf{x}_j$ whose coefficients x_{ij} measure the presence of a particular word stem in the text of a story. The vector dimension is therefore the size of the dictionary.

- We only consider the purely alphabetical words. Numbers and punctuation are eliminated.
- We eliminate all the stop words.

- We consider word stems calculated by the Porter stemmer. Therefore words `classify`, `classifies`, and `classified` are considered identical.
- We only consider word stems that appear in at least three different training stories.

We can therefore compute a matrix of counts n_{ij} indicating the number of occurrences of the word stem i in story j. Since these numbers can vary greatly, we will normalize them as follows. Let N be the number of training stories. The number M_i of training documents in which word i appears can be computed from the n_{ij}. We first define

$$\tilde{x}_{ij} = \begin{cases} (1 + \log n_{ij}) \log \frac{N}{M_i} & \text{if } n_{ij} \neq 0\,, \\ 0 & \text{if } n_{ij} = 0\,. \end{cases}$$

and then compute the vector $\mathbf{x}_j$ as

$$x_{ij} = \frac{\tilde{x}_{ij}}{\sqrt{\sum_i \tilde{x}_{ij}^2}}\,.$$

The first operation is a variant of *TF/IDF* normalization that is known to be effective for preprocessing text because it emphasizes words that appear in few documents. The second operation ensures that $\|\mathbf{x}_j\| = 1$, making the final encoding less dependent on the length of the story. Note that the vectors $\mathbf{x}_j$ are very sparse: most coefficients are null.

We want to produce a training file and a testing file for each of the five classification problems, corresponding to the five topics `earn`, `acq`, `crude`, `grain`, and `trade`. As their name suggests, the training files contain the training examples, and the testing files contain the testing examples.

Each example is represented by a sequence of space separated token on a single line. The first token is the class: $+1$ if the story is associated with this topic and -1 otherwise. The following tokens describe the nonzero coefficients x_{ij} described above. Each token has the form $< i >:< v >$ where $< i >$ is a word stem index and $< v >$ is the coefficient x_{ij}. Word stem indices should appear in increasing order. For compatibility with existing software, you should start numbering the word stems from 1 (not from 0.)

For instance a file could start like this:

```
+1 373:0.1273004158 428:0.2471911172 431:0.1934602396 579:0.1517836121 ...
+1 524:0.1687309356 593:0.1795787602 1250:0.2017339215 ...
```

Please report the following information:

- The dictionary size.
- The average number of non zero coefficients in the training and testing patterns.
- The number of training and testing examples of each class for each topic.

Data preprocessing is an error prone process. Although you are expected to go through all the steps yourself, you are *strongly encouraged* to cross-check your results by exchanging *small pieces of information* with your classmates such as counts of examples, etc. Please explain the procedure you have used to check the results. Grading will take into account the ingenuity of the procedures.

Question 3 – Text document categorization

We will now use these files for text document categorization experiments. For simplicity, we will use the testing set as a validation set. We should not do that in the real world, but that will be sufficient for the homework.

The page `http://www.cs.princeton.edu/courses/archive/spring10/cos424/w/hw2` contains a pointer to *LibLinear* which is a well maintained software package for linear classification. You should first download it and compile it.

- The LibLinear program `train` implements linear classifiers with a number of loss functions selected by the command line option `-s`. We are interested in options `-s0` which implements the log loss $\ell(z) = \log(1+e^{-z})$ and option `-s3` which implements the hinge loss $\ell(z) = \max(0, 1-z)$. The software minimizes a cost function of the form

$$C(w) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ell(y_i\, \mathbf{w}^\top \mathbf{x}_i)\,.$$

  Use option `-C` to specify the parameter C that controls the tradeoff between keeping the weight vector $\mathbf{w}$ small and minizing the empirical errors. This is essentially similar to the parameter ϵ we have been using for the Adult dataset in the second lecture.

- The LibLinear program `predict` can then be used to run the computed classifier on the testing set. This program has an option `-b` to output scores representing the estimated probability of belonging to class $+1$. These scores are derived from the dot products $\mathbf{w}^\top\mathbf{x}$ but are normalized in range $[0, 1]$.

For each of the five classification problems, and for each of the two choices of loss functions `-s0` and `-s3`, produce a plot comparing the ROC curves measured on the testing set for the classifiers obtained with $C = 0.01$, $C = 0.1$, and $C = 1$.

Each point of the ROC curve is obtaining by plotting

$$\left(\frac{\#\text{negatives recognized as positives}}{\#\text{negatives}}\,,\ \frac{\#\text{positives recognized as positives}}{\#\text{positives}} \right)$$

for a classifier that recognizes as positives all the testing examples with a score greater than a certain threshold. When you vary the threshold, you get the curve. Make sure to produce interesting curves by setting properly the bounds of the axes!

Can you choose a better value of C ?