CS6140: Machine Learning **Spring 2017**

Problem Set 2

Official Due Date: March 1, 2017
Handed Out: February 18, 2017 *Extended Due Date: March 17, 2017*

- Please submit your solutions via your CCIS `github` account.
- Materials associated with this problem set are available at `https://github.ccs.neu.edu/cs6140-03-spring2017/materials`.
- I encourage you to discuss the homework with other members of the class. The goal of the homework is for you to learn the course material. However, you should write your own solution.
- Please keep your solution brief, clear, and legible. If you are feeling generous, I would *really* appreciate typed solutions (and if you plan on publishing CS/Math/Engineering research, this is actually a good exercise) – see the source material if you would like to use LaTeX to do this.
- I encourage you to ask questions before class, after class, via email, or the Piazza QA section. However, please do not start e-mailing me questions the night before the homework is due. ☺

1. [**Naïve Bayes I – 20 points**] A function we have discussed (briefly) in class is the m-of-n function, where $\mathbf{x} \in \{0,1\}^n$ is an n-dimensional Boolean vector and $f(\mathbf{x}) = 1$ if and only if at least m values of $\mathbf{x} = 1$.

   (a) Write down a linear threshold function of the form $f(\mathbf{x}) = sgn(\mathbf{w} \cdot \mathbf{x} - \theta)$. Specifically, what are the values of $\mathbf{w}$ and θ.

   (b) Now assume that your training data is the complete enumeration of the 8-dimensional Boolean hypercube (*i.e.,* there are 2^8 examples) and the data is labeled by a 3-of-8 function.

   Knowing from class that the 2-class Naïve Bayes hypothesis is given by

$$\log \frac{p(y=1)}{p(y=0)} + \sum_{i=1}^{n} \log \frac{1-\mu_i}{1-\chi_i} + \sum_{i=1}^{n} \left[\log \frac{\mu_i}{1-\mu_i} - \log \frac{\chi_i}{1-\chi_i} \right] x_i > 0$$

   where $\mu_i = p(x_i = 1|y = 1)$ and $\chi_i = p(x_i = 1|y = 0)$, write down the resulting learned Naïve Bayes hypothesis. [Hint: This is basically a counting problem; for example, $p(y = 0)$ is the number of binary vectors of length 8 with {0,1,2} positive bits divided by the total space of 8 dimensional Boolean vectors.]

   (c) Does the Naïve Bayes learning algorithm learn the target function? Interpret this result in terms of the Naïve Bayes hypothesis space as compared to the hypothesis space of a general linear function (*i.e.,* $\mathbf{w} \in \mathbb{R}^d, \theta \in \mathbb{R}$).

2. **[Naïve Bayes II – 40 points]**

For this problem, you will be implementing the multinomial version of the Naïve Bayes algorithm for text classification.[1] Specifically, we will be considering the email folder classification problem based on the Enron email dataset available in its entirety at `http://www.cs.cmu.edu/~enron/`. Even more specifically, I have selected a subset of the data processed by Ron Bekkerman for this specific task. If you are interested in exploring this dataset further, information information including a technical report is available at `http://management.haifa.ac.il/images/info_people/ron_bekkerman_files/email.pdf`. However, we will only be considering the email of Michelle Lokay[2] and attempting to automatically classify her emails into one of eight folders.

In the course github repository, you should first observe the `lokay-m.zip` file which contains the original emails as processed by Ron Bekkerman. In the `processed` folder, I have further processed the data, all of which was done with `mnb/ProcessEmail.java` (also in the github repository). Based on this process, you will find eight training files (one from each directory) following the naming convention `[directory].train.txt`. The format of each of these files (the example below is from `corporate.train.txt`) is

```
brave 2
logos 3
child/children=20 3
time. 5
time: 10
logon 6
```

where the first column is the word and the second column is the number of times the word occurs with respect to the given class label. Furthermore, `vocabulary.txt` contains the counts for the entire vocabulary, `train-files.txt` is a list of all the training email files, and `test-files.txt` is a list of all the testing email files. The basic preprocessing algorithm was to compile the entire `Lokay` corpus, remove end of sentence periods and oxford-style commas, lowercase the corpus, and split on spaces (see `process_line` if you want to change this for some reason). Furthermore, I removed all tokens which did not occur at least three times and the 100 most frequent tokens (in a modest effort to remove determiners and the such). Note that you have to essentially compute $p(y = y')$, $\forall y'$ and $p(x_i = w|y = y')$, $\forall w_i, y'$, which can be done entirely from the `[directory].train.txt` files and `train-files.txt`.[3]

[1] If you are looking for a reference beyond what was presented in class, `http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html` is a pretty good description.

[2] Yes, I am sort of uncomfortable with these emails being made public for the more innocent parties – however, I have never actually read them and I doubt you will either.

[3] I used `vocabulary.txt` to generate a global word map, but there are other ways to do this.

Secondly, you have been provided with a `test.txt` file which contains the testing data such that the first column is the email folder label (and should not be a feature). Training data consists of all emails before 2002 and testing data consists of all emails from 2002. The distribution of emails per folder is given below.

```
articles training:237 testing:6
corporate training:362 testing:45
enron_t_s training:173 testing:6
enron_travel_club training:19 testing:2
hea_nesa training:79 testing:12
personal training:159 testing:31
systems training:109 testing:17
tw_commercial_group training:1008 testing:151
```

As previously stated, for this problem, you will be generating a Naïve Bayes classifier. The specific form of Naïve Bayes for text classification I would like you to consider is given by Algorithm 1.

Algorithm 1 (Multinomial) Naïve Bayes Learning

Input: Labeled training corpus $\mathcal{S}$ which is label-stratified as denoted by $\mathcal{S}_y$ where $y \in \mathcal{Y}$ is the set of output labels.

$\mathcal{V} \leftarrow$ set of all tokens occurring in $\mathcal{S}$ {Use `train-files.txt`}
for $y \in \mathcal{Y}$ **do**
  $p(y) \leftarrow \frac{|\mathcal{S}_y|}{|\mathcal{S}|}$
  $\mathcal{C}_y \leftarrow$ corpus generated by concatenating $\mathcal{S}_y$ {These are `[directory].train.txt`}
  for $w \in \mathcal{V}$ **do**
    $n_w \leftarrow$ number of times w occurs in $\mathcal{C}_y$
    $p(x_i = w|y) \leftarrow \frac{n_w+1}{|\mathcal{C}_y|+|\mathcal{V}|}$
  end for
end for

Output: Learned parameters $p(y)$ and $p(x|y)$ for all $y \in \mathcal{Y}, w \in \mathcal{V}$

Once you have learned the parameters, the decision rule is given by

$$\hat{y} \leftarrow \arg\max_{y \in \mathcal{Y}} p(y) \prod_{i=1}^{d} p(x_i = w|y)$$

where d is the dimensionality of the test document. Note that this product will end up being a very small number and therefore you will want to change this to a sum of logs to prevent underflow.

(a) Create a program that can be run with the command

`./nb-run`

which should produce a predictions file `predictions.nb` such that the labels should be mapped as stated int Table 2.

Text Label	Numerical Value
articles	1.0
corporate	2.0
enron_t_s	3.0
enron_travel_club	4.0
hea_nesa	5.0
personal	6.0
systems	7.0
tw_commercial_group	8.0

Table 1: Label to Numerical Value Mapping

While this labeling might seem a bit strange, it will allow us to maintain consistency *if* I decide to use this data in future problem sets. Look at `output/labels.txt` to see the "gold" labels in the decided format. Additionally, in the `output` directory, you can run `evaluate.pl labels.txt predictions.nb` can be used to generate a confusion matrix (and `predictions.nb` is the file generated above).[4]

(b) Describe anything you did differently in regards to processing the files or anything else we may find interesting. Note that you are allowed to use the files *as-is* and receive full credit. However, I am always impressed by interesting results.

(c) Write down the confusion matrix for the Naïve Bayes output.

(d) Interpret these results.

3. **[Logistic Regression – 40 points]**

For this problem, you will be implementing stochastic gradient descent for text classification with logistic regression. Specifically, we will again be considering the email folder classification problem based on the Enron email dataset. However, to simplify the problem, we will be attempting to automatically classify her emails into one of two folders, `personal` or `corporate`.

For this problem, in the `libsvm` folder, I have processed the data into `libsvm format`,[5] all of which was done with `logistic/ProcessEmail.java` (also in the github repository) to generate sparse feature vectors. Basically, each feature has an id followed by a superfluous `1.0` to indicate a *strength* of 1. If you would like to see what the ids correspond to, look at `features.lexicon`.[6]

[4] You are welcome to generate your own method for deriving a confusion matrix.

[5] I have adopted this standard as it is reasonably widely used – see `https://www.csie.ntu.edu.tw/~cjlin/libsvm/` for more information

[6] If you do look at this, it is easy to see that there is room for better preprocessing.

Based on this process, you will find one training file and one test file. As previously, the basic preprocessing algorithm was to compile the entire `Lokay` corpus, remove end of sentence periods and oxford-style commas, lowercase the corpus, and split on spaces (see `process_line` if you want to change this for some reason). Furthermore, I removed all tokens which did not occur at least three times and the 100 most frequent tokens (in a modest effort to remove determiners and the such). Training data again consists of all emails before 2002 and testing data consists of all emails in 2002. The distribution of emails per folder is given below.

```
corporate training:362 testing:45
personal training:159 testing:31
```

As previously stated, for this problem, you will be generating a Logistic Regression classifier with parameters estimated via Stochastic Gradient Descent (SGD) as given by Algorithm 2.

Algorithm 2 (Binary) Logistic Regression with Stochastic Gradient Descent

Input: Labeled training corpus $\mathcal{S} \subset \mathcal{X} \times \mathcal{Y}$ (where $\mathcal{X} \subset \mathbb{R}^D$ is the feature space and $\mathcal{Y} \in \{0, 1\}$ is the label space); learning rate α; number of rounds T

$\mathbf{w} \leftarrow \mathbf{0}, w_0 \leftarrow 0$ $\{w_0$ is a standard convention for learning a *bias*$\}$
{Should pick your own halting criteria, but hard-coding rounds will work}
for $t = 1, \ldots, T$ **do**
  SHUFFLE($\mathcal{S}$)
  for $(\mathbf{x}, y) \in \mathcal{S}$ **do**
    $\sigma(\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x}-w_0)}$ {Sigmoid function}
    $\delta \leftarrow y - \sigma(\mathbf{x})$
    {D is dimensionality}
    for $d \leftarrow 1, \ldots, D$ **do**
      $w_d \leftarrow w_d + \delta x_d$
    end for
    $w_0 \leftarrow w_0 + \delta$
  end for
end for

Output: Learned parameters $\mathbf{w}$ and w_0

Once you have learned the parameters, the decision rule is given by

$$\hat{y} \leftarrow [\![\sigma(\mathbf{x}) \geq 0.5]\!]$$

where $[\![p]\!] = 1$ iff p is true. Since you are using SGD, you should randomize the order for every round of training, set the learning rate to a reasonable value, and possibly use a hold-out set to determine convergence.

(a) Create a program that can be run with the command

`./lr-run`

which should produce a predictions file `predictions.lr` such that the labels should be mapped as stated int Table 2.

Text Label	Numerical Value
corporate	2.0
personal	6.0

Table 2: Label to Numerical Value Mapping

Again, the odd labeling will allow us to maintain consistency *if* I decide to use this data in future problem sets. Look at `output/labels26.txt` to see the "gold" labels in the specified format. Additionally, in the `output` directory, you can run `evaluate.pl labels26.txt predictions.lr` can be used to generate a confusion matrix (and `predictions.lr` is the file generated above).

(b) Describe anything you did differently in regards to processing the files or anything else we may find interesting. I am particularly interested in how you set the learning rate and number of rounds. Note that you are allowed to use the files *as-is* and receive full credit. However, I am always impressed by interesting results.

(c) Write down the confusion matrix for the logistic regression output.

(d) Interpret these results.

- Use your CCIS github repository to submit all relevant files. You are free to use the programming language of your choice, but please attempt to conform to the instructions above. To be safe, try submitting something **before** the assignment deadline.

- The code you submit must be your own. If you find/use information about specific algorithms from the Web, etc., be sure to cite the source(s) clearly in your source code. You are not allowed to submit existing naïve Bayes or logistic regression implementations or code downloaded from the internet (obviously).