Big Data Analytics Programming

Assignment 1:

Blocked Matrix Multiplication and Spam Filters

Jessa Bekker
jessa.bekker@cs.kuleuven.be

Toon Van Craenendonck
toon.vancraenendonck@cs.kuleuven.be

Collaboration policy:
Projects are independent: no working together! You must come up with how to solve the problem independently. Do not discuss specifics of how you structure solution, etc. You cannot share solution ideas, pseudocode, code, reports, etc. You cannot use code that is available online. You cannot look up answers to the problems online. If you are unsure about the policy, ask the professor in charge or the TAs.

1 Blocked Matrix Multipication [5 points]

In this part of the assignment you will implement blocked matrix multiplication in C. Look at matrix_multiplication.c and complete the parts marked `TODO`. For simplicity, we will work with square matrices. You can use gcc to compile the code:

```
$> gcc matrix_multiplication.c -o matrix_multiplication
```

Make a script "mulmatrix.py". The first argument to the script is a file name F. The remaining arguments are possible block sizes. Each line in F will be of the form

`<N><whitespace><filename_matrixA><whitespace><filename_matrixB>`.

In this format, `<N>` is size of the matrix (N x N) and `<filename_matrixA>` and `<filename_matrixB>` are the matrices to be multiplied. The file "input_matrix.txt" contains an example of the file format.

The script should call your matrix multiplication code to multiple matrixA and matrixB using the naive strategy and the blocked strategy for each given block size.

Then the script should create a graph 'graph.png' that shows the results with matrix size on the x-axis and time on the y-axis. Create one line for each block size and one line for the naive strategy. This plot should be included in your report.

2 Spam Filters [20pts]

The goal of this part of the assignment is to build and evaluate some online spam filter algorithms that can deal with an infinite features space. The spam filters are supposed to operate in the following setting: Whenever the user receives an e-mail, the filter should classify it as spam or ham (= legitimate e-mail). Then, the user labels the e-mail as spam or ham and the filter should be updated with this information. The (base) features of the spam filter are the n-grams (groups of words) that appear in the e-mail. It is not known beforehand which words will appear, how many words and which ones will prove to be important. The words can even change over time, for example, when the spammers have learned that all their e-mails with "FREE TRY NOW" get blocked, they might switch to "WINNER WINNER WINNER".

You will implement two techniques for spam filtering in `Java`: Naive Bayes and the Perceptron classifier. To deal with infinite and unknown features, you will also use two techniques: Feature hashing and Count-Min Sketch.

2.1 Classifiers

2.1.1 Naive Bayes

Naive Bayes is a simple, standard, method to train a classifier. It was one of the original methods for spam filtering [SDHH98].

In the traditional formulation of Naive Bayes with binary features, the feature being 'true' or 'false' are considered equally informative. In our case, the features are the words that (do not) appear in the e-mail. Naively, the presence of a word in an e-mail is thus considered as informative as the absence of that word:

$$\Pr(S|\text{text}) = \frac{\Pr(S)\Pi_{w\in\text{text}}\Pr(w|S)\Pi_{w\notin\text{text}}(1-\Pr(w|S))}{\Pr(\text{text})}$$

However with a large vocabulary, the absence of a specific word is not informative. Intuitively, an e-mail only contains a subset of the words that could have been used to deliver the message. Therefore, a word that does not occur in an e-mail is considered unobserved, i.e.: it may or may not be in the e-mail. Under this assumption, the Naive Bayes formula simplifies to:

$$\Pr(S|\text{text}) = \frac{\Pr(S)\Pi_{w\in\text{text}}\Pr(w|S)}{\Pr(\text{text})}$$

The calculations for Naive Bayes should be done in the logarithmic domain, otherwise they might suffer from underflows. Hint: sums should be calculated as follows:

$$\log(a+b) = a + \log(1+\exp(b-a))$$

The probabilities in the formula are estimated from counts in the data. However, we must avoid zero-counts. Naively, a zero-count corresponds to a zero probability. If a spam e-mail comes in with the words "free", "viagra" and "roses" in it and "roses" never occured in spam before, then this mail would be falsely classified as ham. To prevent such situations, methods such as Laplace estimates can be used where '1' is added to every count. This is equivalent to initializing the classifier with a spam e-mail that has all possible words and a ham e-mail that has all possible words.

2.1.2 Perceptron

The perceptron classifier dates from 1958 and is one the first artificial neural networks, laying the foundations of the current deep neural networks. For this assignment, you will implement learning with the delta rule and stochastic gradient descent.

Attention! The perceptron expects the classes to be 1 and -1, while the implementation provides them as 1 and 0. Keep this in mind when writing the percepron code.

2.2 Dealing with Infinite features

In this assignment you will use two ways to deal with the infinite and beforehand-unknown vocabulary used in the e-mails: Feature hashing and count-min sketch.

2.2.1 Feature Hashing

E-mails can be represented as word vectors, where every entry of the array represents a word which has value '1' if this word appears in the e-mail and '0' otherwise. With an infinite stream of e-mails, the size of this array is unknown and potentially infinite. To deal with this, feature hashing maps the word vector to an array of finite size, using a hash function. The mapped array of finite size is then used as the feature vector in the learning algorithm.

Feature hashing can be interpreted as a sketching method for classifiers with infinite features: For Naive Bayes, it approximates the count of each word with an overestimate, namely the count of all the words that hash to the same value. For the perceptron classifier, it approximates the weights for each word by the weight for all the words that hash to the same value.

In your implementation you can use Murmur hash for hashing words. However, the provided implementation only provides hash functions with ranges of size 2^{32} or 2^{64}. You should adjust the range to the desired range which is given as an input parameter ($2^{\texttt{logNbOfBuckets}}$) in the method `hash(String str)`.

2.2.2 Count-Min Sketch

Count-min sketch is a sketching algorithm for keeping approximate counts, or numbers in general. It is related to feature hashing, but reduces the estimation error by using multiple hashing functions. The name "count-min" comes from the non-negative case, where the number, like a count, can only be increased. In this case the count of the feature value can only be an overestimate. By taking the minimal count accross the counts of the different hashing values for this feature, the estimation error is minimized. The general case, where the numbers can be reduced, like the weights of the perceptron, uses the median instead of the minimum. The count-min sketch paper [CM05] can be found in the folder `assignment1/SpamFilter/literature/`.

Like for feature hashing, you can use Murmur hash, but you should adjust the hash range.

2.3 Classifier Evaluation

Because the spam filters are online learners, the algorithm should be evaluated periodically. In contrast to using a fixed test set, a classifier is evaluated against examples that will subsequently be used to update the

Table 1: Data Characteristics

Dataset	Ham	Spam	Total
Enron	19.088	32.988	52.076
SpamAssasin	6.954	3.797	10.751
Trec2005	39.399	52.790	92.189
Trec2006	12.910	24.914	37.822
Trec2007	25.220	50.199	75.419
Total	103.571	164.686	268.257

model. The provided skeleton implementation uses a reporting period r: every r examples the classifier is evaluated on the next r examples, before adding them to the model and repeating the process.

The provided code evaluates the classifiers using accuracy (# correctly classified/ #total classified). However this is not the only way to evaluate the performance of a classifier. Implement at least two additional evaluation metrics based on the values of the contingency table (#true positives, #false positives, #true negatives, and #false negatives) and discuss why this evaluation metrics are fit for evaluating a spam filter.

2.4 Data

The data used for this assignment is the concatenation of five real world datasets in the given order: Enron[1], SpamAssasin[2], Trec2005, Trec2006, and Trec2007[3]. The datasets are summarized in table 1. All the e-mails are in .eml format and are not preprocessed.

You can access the data directly on the departemental machines in the directory `/cw/bdap/assignment1/data/` and should not copy it to your machine.

The order that the e-mails come in, and their label are dictated by index files. The root index is `/cw/bdap/assignment1/data/index` which points to the index files of the different datasets: `/cw/bdap/assignment1/data/<dataset>/index`. Each line of the index file of a specific dataset has this format: `<label><whitespace><path/to/mail>`. The label is 'spam' or 'ham' and the path points to the e-mail.

Code is provided that simulates a stream of pre-processed e-mails. The pre-processing extracts n-grams (groups of n subsequent words) from the e-mails' subject and body. To get clean n-grams, html tags, punctuation and stop words[4] are removed from the e-mail. Furthermore, all the words are stemmed, which maps similar words to the same stem, e.g.: eat, eats, and eating all map to the stem eat.

2.5 Bonus: Further Improvements

If you want, you can improve your spam filter further. You are free on how to do this, as long as the required spam filters still work as expected with the default parsed e-mails as input. You can for example extract different features from the e-mails. modify the classifiers or implement new classifiers. Discuss the further improvements you made in the report.

[1] `http://csmining.org/index.php/enron-spam-datasets.html`
[2] `http://csmining.org/index.php/spam-assassin-datasets.html`
[3] `https://plg.uwaterloo.ca/~gvcormac/spam/`
[4] We use the stopwords from `https://github.com/stanfordnlp/CoreNLP`

Table 2: Summary of the java classes that should be completed, subclassed or not altered

Complete	Subclass	Do not alter	
`NaiveBayesFeatureHashing`	`EvaluationMetric` (minimum 2)	`OnlineTextClassifier`	`ParsedText`
`NaiveBayesCountMinSketch`	`EmlParser` (optional)	`EvaluationMetric`	`LabeledText`
`PerceptronFeatureHashing`	`ParsedText` (optional)	`MurmurHash`	`EmlParser`
`PerceptronCountMinSketch`	`OnlineTextClassifier` (optional)	`Accuracy`	`PorterStemmer`
		`MailStream`	

2.6 Implementation

All the class stubs and other code are available in `SpamFilter/code/`. Read them and understand them carefully. You should be careful to not only implement everything that is expected, but also, not to alter anything that is not suposed to change. Table 2 gives an overview of which classes should be completed, subclassed or not altered. Below we give some more details about the implementation.

Spam Filters The spam filters that you will implement are subclasses of `OnlineTextClassifier.java`. The superclass is not to be changed, however make sure you understand it. The spam filters are: `NaiveBayesFeatureHashing.java`, `NaiveBayesCountMinSketch.java`, `PerceptronFeatureHashing.java`, and `PerceptronCountMinSketch.java`. These classes are incomplete. In order to make a functional algorithm, you must at least complete the methods that are tagged with "THIS METHOD IS REQUIRED". The headers of all the methods, including the instructor should not be changed. The rest of the code may be changed as long as all the required methods work as expected.

Evaluation Metrics You should add at least two more evaluation metrics. Each metric should be implemented as a subclass of `EvaluationMetric.java`. To make learning curves for your evaluation metrics, add them to the array `evaluationMetrics` in the main method of the spam filters.

Further Improvements You can get different information from the e-mails by subclassing the `EmlParser`. If this information is not represented by Strings, then you can make a subclass of `ParsedText` to hold this information. The required text classifiers should work with the provided `ParsedText`. To allow them to work with a subclass, you can use `text instanceof YourSubClass` and casting where necessary. If you wish to implement new or improved classifiers, you can make subclasses of `OnlineTextClassifier`.

2.7 Compiling and Running code

A Makefile is provided that can be used for compiling, testing and running the code. Make sure that you read and understand this file. You can adjust it for running experiments if you wish.

You can compile the code for all spam filters with `make all` or for each one separately with
`make NaiveBayesFeatureHashing.class`,
`make NaiveBayesCountMinSketch.class`,
`make PerceptronFeatureHashing.class`, and
`make PerceptronCountMinSketch.class`.

You can test the code locally on a subset of the data with `make nbfh_small`, `make nbcms_small`, `make pfh_small`, and `make pcms_small`. From the departemental machines, you can run the code with default parameters on all the data with `make nbfh`, `make nbcms`, `make pfh`, and `make pcms`

3 Report [5 pts]

Part of this course's goal is to gain insights into the interplay between large amounts of data and algorithms. To this end, you should conduct several experiments to explore issues such as efficiency, scalability, effects of parameter values, etc, to help gain insight into how the algorithms you implemented work. Please focus on posing an experimental question (e.g., What is the effect of varying parameter X), designing an experiment to test that question, and then providing a summary of the finding (e.g., increasing the value of parameter X causes...). Any valuable insight on your work will be graded accordingly. You should write a small report of at most three pages (excluding Tables or Figures) explaining your experiments and findings.

First, remember to include the graph for the matrix multiplication experiment in the report.

Second, you should report on the spam task. One experiment that you must perform and include in your report is generating a *learning curve*, which visualizes how a classifier's accuracy (or other evaluation metric) varies as a function of the number of training examples. Figure 1 gives an example of a learning curvre for a logistic regression classifier on the UCI `zoo` dataset. The learning curve in your report should show the performance of the Naive Bayes with feature hashing, Naive Bayes with count-min sktech, perceptron with feature hashing, and perceptron with count-min sketch. FAILING TO USE THE FULL DATASET TO GENERATE THE LEARNING CURVE WILL INCUR A SUBSTANTIAL PENALTY.

The report should also include a discussion of the different evaluation metric that you used and the different insights that they provide.

If you added further improvements to your spam filter, you should include one extra page describing the changes and how they affect the performance.

In addition, the report can include a maximum of one extra page describing any problems (that is, bugs) in your code and what may have caused them as well as any special points about your implementation.

References

[CM05] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[SDHH98] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop*, volume 62, pages 98–105, 1998.

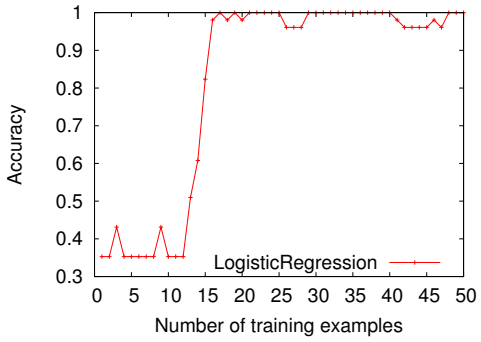

Figure 1: Example of a learning curve on `zoo`.

4 Important remarks about grading and submitting the assignment

Deadline The assignment should be handed in on Toledo before November 19th 11:59 pm. There will be a 10% penalty per day, starting from the due date.

Deliverables You should upload a .zip or .tar.gz file with the following structure and files:

- report_<FirstnameLastname>.pdf
- MatrixMultiplication/
  - matrix_multiplication.c
  - mulmatrix.py
- SpamFilter/
  - NaiveBayesFeatureHashing.java
  - NaiveBayesCountMinSketch.java
  - PerceptronFeatureHashing.java
  - PerceptronCountMinSketch.java
  - <EvaluationMetric1>.java
  - <EvaluationMetric2>.java
  - <EmlParserSubClass>.java (optional)
  - <ParsedText>.java (optional)
  - <OnlineTextClassifier>.java (optional)

The c file should contain the complete source code for the matric multiplication. The java files should contain complete source code for the spam filters. The report with the learning curve must be submitted in `.pdf`.

Evaluation Your solution will be evaluated based on the correctness of your results, the speed of your code, and the style of code.

Automated grading Automated tests are used for grading. Therefore you must follow these instructions:

- Use the exact filenames as specified in the assignment.
- Do not change the provided interface/skeletons, this includes the constructor.
- Your code must run on the departmental machines, this can be done over ssh.
- Never use absolute paths.

If you fail to follow these instructions, you will lose points.

Running experiments This is a class about big data, so running experiments could take hours or even multiple days. You can do this easily on the departmental machines: it takes several minutes to the start the experiments and then you check back later to see the results. Failure to run and report results on the full data set will result in a substantial point reduction.