CSCI 8360 Data Science Practicum
Department of Computer Science
University of Georgia

Project 1: Scalable Document Classification

DUE: Thursday, September 1 by 11:59:59pm

Out August 18, 2016

1 Overview

We'll kick off the semester of practicum with document classification, streamlined to help everyone warm-up with large-scale machine learning and the tools we'll be using. **Your goal is to design a large-scale classifier in Apache Spark that maximizes its classification accuracy against a testing dataset.**

For this project, we are using the Reuters Corpus, which is a set of news stories split into a hierarchy of categories. There are multiple class labels per document, but for the sake of simplicity we'll ignore all but the labels ending in `CAT`:

1. CCAT: Corporate / Industrial
2. ECAT: Economics
3. GCAT: Government / Social
4. MCAT: Markets

There are some documents with more than one `CAT` label. Treat those documents as if you observed the same document once for each `CAT` label (that is, add to the counters for all the observed `CAT` labels).

The format of the data itself is one document per line of the file, and a corresponding file with the ground-truth document labels. In effect, you have files `X.txt` and `y.txt` which

contain the same number of lines–in the former are the documents, and in the latter are the document class (label) listings on corresponding line numbers.

Specifically, here are the available files:

- `X_train_vsmall.txt`, `y_train_vsmall.txt`
- `X_test_vsmall.txt`, `y_test_vsmall.txt`
- `X_train_small.txt`, `y_train_small.txt`
- `X_test_small.txt`, `y_test_small.txt`
- `X_train_large.txt`, `y_train_large.txt`
- `X_test_large.txt`

The data have already been split into `train`ing and `test`ing subsets so you don't need to worry about implementing distributed cross-validation.

There are "very small," "small," and "large" versions of the data available. The "very small" and "small" are meant to help in initial algorithm development, as these data are small enough to fit on one machine. Only when you are confident in your core classification strategy should you move to the "large" dataset.

You'll notice there is no `y_test_large.txt` file. That file is being held on AutoLab, and will be used to evaluate your trained model and score your submission.
You can view the data on Amazon S3 through the url:

`https://s3.amazonaws.com/eds-uga-csci8360/data/project1/<file>`

and you can access the data on S3 in Spark using the url:

`s3://eds-uga-csci8360/data/project1/<file>`

2 Theory Guidelines

If you have no idea where to begin with designing a large-scale algorithm for document classification, this section is meant to provide a baseline to get you started. It should be noted, though, that to achieve grades in the **A**-range, you'll need to go above and beyond what is described here.

2.1 Naïve Bayes

Naïve Bayes is a probabilistic classifier based on Bayes' Theorem of conditional probabilities. Let's say we're working with data X, which is $n \times d$ (n instances, each with d dimensions), each of which belongs to one of K possible classes. Given some instance $\vec{x} \sim X$, we want to classify it into one of K classes y_k.

$$P(Y = y_k | X = \vec{x}) = \frac{P(Y = y_k)P(X = \vec{x}|Y = y_k)}{P(X = \vec{x})}$$

However, solving this equation in practice is often infeasible, especially when the dimensionality of the data $\vec{x}$ is extremely high. This is where the "naïve" part of Naïve Bayes comes into play: we assume that, conditioned on the class y, that the features of $\vec{x}$ are independent of each other.

$$P(Y = y_k | X = \vec{x}) \propto P(Y = y_k) \prod_{i=1}^{d} P(x_i | Y = y_k)$$

We can compute this probability for each label y, and to perform the classification, return the label with the highest probability as our prediction.

$$\hat{y} = \text{argmax}_{k \in 1 \ldots K} P(Y = y_k) \prod_{i=1}^{d} P(x_i | Y = y_k)$$

For document classification, you can consider a document as a single instance $\vec{x}$ with d features, where d is the number of words in your vocabulary (the number of *unique* words across *all* documents). Conditional independence in Naïve Bayes imposes the bag-of-words model, where word ordering does not matter, only word frequency.

Word probabilities, then, can be computed with simple term frequencies, where the probability of a particular word conditioned on a class $P(x_i|y_k)$ is the number of times that word appears in documents belonging to class y_k divided by the total number of times the word appears in *all* documents.

Once you have trained your model–essentially, counted all the words in the training set under the different class conditionals–you can then test the accuracy of your model against a testing set.

One final piece of advice in this starter approach: you will need to be able to handle the case where a word shows up in testing that you never observed in training. Without any additional tweaks, your counter for this word would be 0, which then multiplied with the rest of the conditional probabilities of your document would result in an overall probability of 0 for all categories y_k. Not good!

The easiest way to deal with this is impose *pseudocounts*, where an extra count of 1 is added to all words in both training and testing, eliminating any 0 probabilities while still keeping probabilities of infrequent words small.

2.2 Beyond Naïve Bayes

A small list of "extras" you can undertake to boost your grade into the **A**-range. This list is NOT exhaustive, merely a few suggestions. I encourage you to come up with your own improvements!

- **Features**: Word counts are nice but they suffer from intrinsic selection bias, especially common words like articles and prepositions that contribute little semantic meaning. Using TF-IDF or hashing frequencies may improve classification accuracy.
- **Smoothing**: Pseudocounting prevents the worst-case scenario of a 0-probability label if you observe a word in testing that you did not observe during training. However, you could potentially implement more sophisticated additive smoothing that scales better with the overall size of the vocabulary.
- **Regularization**: From a different angle, regularization may help focus the decision boundary on the words that make the most difference in classification while simultaneously obviating the need for smoothing.
- **Stopwords**: There are plenty of examples of words that occur frequently but are semantically useless for classification. Removing these prior to training the model can help; unfortunately, there's no universally-agreed-upon stopwords list, so you'd need to come up with your own.
- **Algorithms**: Naïve Bayes is definitely the go-to for document classification, but other classifiers can also be used. Random forests (which are embarrassingly parallel!) and logistic regression are two examples of potential alternative classification strategies that have reasonable training routines.

3 Engineering Guidelines

Everyone must use Spark. What language API you use is entirely up to you and your team. However, you are not allowed to use any packages that build *on top of* Spark. For example, the "CaffeOnSpark" package is a deep learning implementation that runs on top of PySpark; this is forbidden. Furthermore, you cannot use algorithms implemented in Spark MLlib or ML–even the featurizing utilities are forbidden for this assignment. However, you are welcome to use local packages (like NumPy arrays or Breeze vectors). Additionally, you can use Spark DataFrames if you prefer those to RDDs. Bottom line: if in doubt, ask me.

Create a GitHub repository for you and your team to work on the project. You are more than welcome to make it a private repository, but the development must take place on GitHub, as I will be using it and the activity you record on it as part of the grading process. I would highly recommend using the repository naming scheme of `team-name-project1` to help differentiate from other teams and future projects. Make sure your repository includes 1) a README giving an overview of your program and how to install and run it, 2) a MEMBERS file containing the names of you and your teammates, and 3) your code.

Compute log-probabilities so you don't run into underflow errors. When you choose the conditional independence of Naïve Bayes, you often end up multiplying lots of tiny probabilities together, which has the potential to underflow. To prevent this, compute the log of these probabilities and, instead of multiplying, add them (as per properties of logarithms). Since the log is a monotonic transformation, then if $P(x) < P(y)$, it holds that $\log P(x) < \log P(y)$.

Exercise good software design principles. This means: adopt a consistent coding style, document your code, use the GitHub ticketing system to identify milestones and flag bugs, and provide informative and frequent `git commit` comments.

Use packaged EC2 setup scripts. You have two options here. Spark comes with default EC2 startup scripts in the `spark-ec2` folder that function perfectly well. If, however, you prefer an even more streamlined interface, you can install flintrock, which provides additional niceties for setting up and tearing down EC2 clusters for Spark.

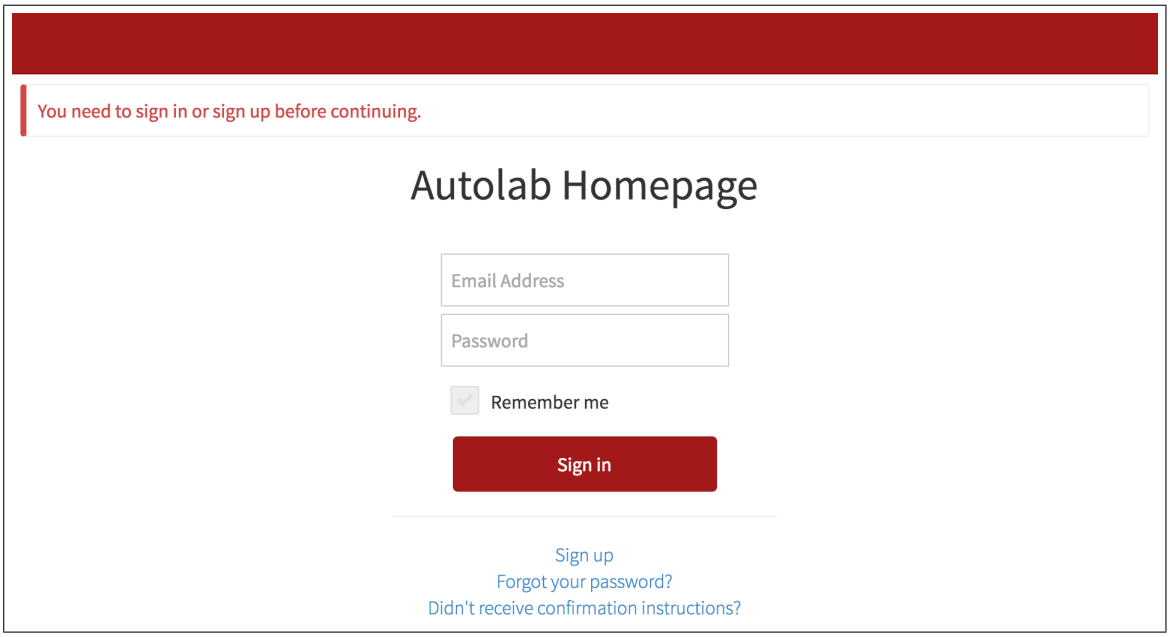
Shut down your EC2 clusters when you are finished testing. AWS EC2 instances incur costs for every hour they are turned on, even if they are not actively running any jobs. I will not hesitate to revoke AWS credentials that are racking up disproportionate expenses. Here's the EC2 dashboard you can use to double-check if you have any resources allocated: `https://quinngroup.signin.aws.amazon.com/console`

4 Submissions

All submissions will go to **AutoLab**, our brand-new autograder and leaderboard system (see Fig. 4).

You can access AutoLab at `https://autolab.cs.uga.edu`. Your team will first need to create an account on AutoLab before you can be added to the `csci8360-fa16` course and submit assignments.

To clarify: whether you create an *individual* account and submit on behalf of your team, or create a single *team* account that everyone on your team uses to submit, is entirely



up to you. I mention this because it may get confusing if some of your team members create their own individual accounts and submit separately.

Once you have been added to the course, you can make submissions on behalf of your team to the **Project 1** assignment that is open. When you do, you will submit one file: **a text file containing the predictions of your trained classifier on the `X_test_large.txt` dataset, one prediction per line for each document**. For example, if you had three documents in the test set to classify, your output might look like:

```
MCAT
CCAT
CCAT
```

Your classification accuracy will be tabulated and compared against the true labels (hidden on the server) and should appear on the leaderboard.

If you think you can do better than what shows up–particularly to beat out everyone else–make another submission! There's no penalty for additional submissions, but keep in mind that swamping the server at the last minute may result in your submission being missed; AutoLab is programmed to close submissions *promptly* at 11:59pm on Sept 1, so give yourself plenty of time!

DIRE WARNING: If you do not adhere to this output format and the autograder crashes as a result, your entire team will be forthwith banished from Athenshire! Seriously

though, please adhere to this output format.

5 Grading

Given that this is a practicum, and that you're all talented graduate students, the grading for the projects is a little different, operating on a sliding scale with some stationary points that require extra effort to move beyond (see Fig. 5).

Base Grade	Requirements	Example
A (95%)	Above and beyond, highly customized algorithm or wholly new approach. Documentation in code and README are excellent. Within 5% of leader.	Project 1: Naïve Bayes with sophisticated smoothing, stopwords, and preprocessing, or wholly different classifier. Thorough commit messages and significant team participation, use of GitHub tickets.
B (85%)	Used suggested algorithm with little or no customization. Code and README are adequately documented in GitHub. Within 5-10% of leader's score.	Project 1: Naïve Bayes with pseudocounts, minimal preprocessing, occasional commit messages, sparse in-code documentation.
C (75%)	Used suggested algorithm with no extra customization. Little documentation or use of GitHub. Within 10-20% of leader's score.	Project 1: Naïve Bayes with no pseudocounts or other regularizer. Minor bugs. Few meaningful commit messages, little or no documentation in code.
D (65%)	Partial or incomplete project submission. Little or no evidence of collaboration on GitHub. Greater than 20% discrepancy from leader's performance.	Project 1: Naïve Bayes as documented in the project handout. Major bugs present. Very little use of GitHub's team organizing tools.
F	Serious problems with the project submission that demonstrate little or no effort.	Project 1: Little effort evident in the project. Only a handful of git commits with no messages, or no use of GitHub at all. Code is nonfunctional or nonexistent.

Everyone's starting grade is a **B**. If you follow the guidelines in this handout, implementing the basic algorithm as described, and everything functions properly, and your documentation is reasonably ok, and there were no major issues with your team, then you also finish with a **B**. If you're happy with that, great! If not, read on:

Getting an **A** requires an investment of additional work, both in terms of functionality and algorithm performance, as well as on the software engineering side. You must demonstrate the extra customizations you made to your code and how those tweaks resulted in an improved score on the final testing set (getting the top score, for instance, would certainly qualify). Furthermore, you must show exemplary software engineering techniques: your code style should be clean, logical, and well-documented, and you should provide a clear README with instructions to new users on installing and deploying your document classifier. Finally, your `git commit` comments and GitHub tickets should provide a "paper trail" of documentation to show the route your team took in developing the software.

This combination of **algorithmic novelty** and **engineering best-practices** will earn you and your team that well-deserved **A**.

6 Reminders

- Your grade depends on two factors: the accuracy of the output you submit to AutoLab, and the quality of your code on GitHub. Don't neglect one!
- You only need to correctly predict the `CAT` label, and if there are multiple, you only need to get one of them for the answer to be correct.
- Be careful about text encoding; you may see a lot of "`&quot;`" constructs in the documents. Make sure you can parse these out so they don't become tokens in their own right.
- If you run into problems, work with your teammates. If you still run into problems, query the Slack channel. I'll be regularly checking Slack and interjecting where I can.
- This is the first time 8360 has been offered, and the first time AutoLab has been used, so there are bound to be hiccups. Please be patient! Thanks for your understanding.