Assignment #3 - n-grams and maximum entropy

Submit Assignment

Due No due date **Points** 30 **Submitting** a website url

Introduction

You have looked at n-grams in this and previous courses. And in this course, we talked about maximum entropy classifiers in **Lecture 4**. In this assignment, you will apply a maximum entropy classifier to train an n-gram language model, although you won't apply the language model for anything but the most rudimentary performance measures.

You can/should use the off-the-shelf maximum entropy classifier tool **sklearn.linear_model.LogisticRegression (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)**. The basic goal is to be able to train a classifier that, for any sequence $w_1, w_2, ..., w_{n-1}, w_n$ for given n, the classifier will use the sequence $w_1, w_2, ..., w_{n-1}$ to predict w_n, as though w_n were just another class label. The use of softmax/logistic regression/MaxEnt allows us to develop a distribution over a large number of classes, assuming that we're willing to put in the processor power to do so. (We have barely talked about this, but this is precisely equivalent to a very simple "feed-forward" neural network with no "hidden layers".)

This assignment is officially due **2019 March 11 at 23:59.** There are **30 points** available, and 10 bonus. Everything must run on mltgpu. You can use any of the available packages on mltgpu—however, the initial data processing into one-hot vectors (see Part 1) must be done by your code, not an existing automatic vectorizer.

The data

As a source of training and test data, we have provided the following file on mltgpu:

```
/scratch/brown-rga.txt
```

These are POS-tagged sentences excerpted from the Brown corpus. The POS tags are only there for the bonus.

The data looks like this:

```
there/EX was/BEDZ about/IN that/DT song/NN something/PN incandescent/JJ ,/, for/CS this/DT brahms/NP wa
s/BEDZ milstein/NP at/IN white/JJ heat/NN ./.
not/* the/AT noblest/JJT performance/NN we/PPSS have/HV heard/VBN him/PPO play/VB ,/, or/CC the/AT mos
t/QL spacious/JJ ,/, or/CC even/RB the/AT most/QL eloquent/JJ ./.
those/DTS would/MD be/BE reserved/VBN for/IN the/AT orchestra's/NN$ great/JJ nights/NNS when/WRB the/AT
soloist/NN can/MD surpass/VB himself/PPL ./.
```

The data has been tokenized and lowercased, so there is no further processing that needs to be done, other than stripping the POS tag after the final /.

You should expect your code to be evaluated with data you haven't seen, but it is guaranteed to be in this format.

The repository

Fork and clone the following repository:

https://github.com/asayeed/lt2212-v19-a3 (https://github.com/asayeed/lt2212-v19-a3)

It contains three scripts and a README.md, all of which you will modify for this assignment.

Part 1: converting the data (8 points)

In this part of the assignment, you will convert the text into instances with feature values to be fed to the LogisticRegression classifier. Given the n-gram, $w_1, w_2, \ldots, w_{n-1}, w_n$, the first $n-1$ features will be converted to one-hot encodings. (See the full page of slide 31 of **Lecture 3** to remind yourself of what this is.)

That is, you will create binary one-hot representations for every word in the vocabulary, whose width will be (by definition) the size of the vocabulary. Then, for each given n-gram, you will concatenate $n-1$ of these representations, to form a representation of the training features of that n-gram. Only $n-1$ of these values in that representation will be 1, the rest 0.

The last column of the representation will just be the literal word w_n itself, as a string, without one-hot representation. This is the class label.

You will thus fill out the code for gendata.py, which should be run as follows:

```
python3 gendata.py [-N N] [-S S] [-E E] inputfile outputfile
```

where (the square brackets [] mean "optional", in case you didn't realize):

- inputfile contains the POS-tagged text data in the format of /scratch/brown-rga.txt.
- outputfile contains the converted features to be used in the next two scripts.
- N is the n of the n-gram, by default trigrams/3-grams. In other words, your script can generate instances of different n-gram widths. (NOTE: you will have to consider inserting a sufficient number of start symbols as an extra vocabulary item, in order that you don't lose the first word of each sentence as a potential class! Also, the minimum N should be 2.)
- S is the line of the input file, starting from default 0, to excerpt the input file/corpus for processing. This is to help generate separate training and test data, and to choose sample sizes that run in reasonable time.
- E is the line of the input file, by default the last line, that is to be converted to the output feature table.

You can add any other options, as long as you document them in the README.md file (the same goes for Part 2 and Part 3). Note that this is actually going to be a relatively inefficient way of doing things, but we're doing it for the exercise—another "getting data from point A to point B" exercise, so you should not use modules that automatically solve this problem for you, other than numpy, pandas, and the standard Python modules.

Part 2: training script (8 points)

Here, you will train a LogisticRegression model on instance data created by gendata.py and write out the model to a file that can be re-used for testing purposes. The script is to be invoked as follows:

```
python3 train.py [-N N] datafile modelfile
```

where N is just the same n-gram parameter as above. This script will train the model and write the model out (probably via Python's **"pickle"** **(https://scikit-learn.org/stable/modules/model_persistence.html)** persistence module). You should use the multi_class='multinomial' option on the LogisticRegression classifier.

Part 3: testing script (8 points)

This script will load the model and apply it to a file generated by gendata.py in order to emit a couple of test statistics to the console: accuracy and **perplexity** **(https://en.wikipedia.org/wiki/Perplexity)** over all training instances. (The LogisticRegression classifier contains the attribute classes_ and the predict_log_proba and predict_proba methods. Try to use these representations efficiently to produce the test statistics.) The script will be run like this:

```
python3 test.py [-N N] datafile modelfile
```

That is, just apply the model to the data.

Part 4: reporting (6 points)

You should fill out the README.md file with any notes, explanations of extra command-line options, and so on.

This time, there is a "creative experimentation" portion of this assignment. You should come up with your own set of tests on the scripts you have written, by varying the width of the n-grams and the size of the training samples. Give the command lines you used and the outputs of test.py. You do not need to include any of your model files in the submission. Instead explain any hypotheses you made, how you tested them, and what you observed. This need not be more than 2-4 paragraphs worth of text and will be graded on reasonable effort.

Don't forget to commit and push to the master branch (the default, normally), and then submit the URL of your repository right here.

Part Bonus: POS tags (10 points)

Add to the above scripts whatever command-line options and additional code that are necessary to model the output of POS tags rather than words as class labels as well as POS tags as input features *alongside* words, for any width of n-gram. Run the same tests and report and discuss as in Part 4, doing tests parallel to your tests *without* n-grams and comparing any trends you see in both results.