

Natural Language Processing Homework 3

Katie Chang, Willis Wang

March 8, 2016

README

1 Perplexity per Word

First, we get that the \log_2 -probability of each of the sample files is as follows:

-12111.3 ../../speech/sample1

-7388.84 ../../speech/sample2

-7468.29 ../../speech/sample3

Word count for each sample file (found with `wc -w`)

sample1: 1686

sample2: 978

sample3: 985

Cross-Entropy for each sample:

sample1: 7.18345

sample2: 7.55505

sample3: 7.58202

Calculating the perplexity per word for each of the sample files:

$2^{-\frac{1}{N} * \log_2 p(x)}$

sample1: $2^{-\frac{1}{1686} * (-12111.3)} = 2^{7.18345} = 145.3565$

sample2: $2^{-\frac{1}{978} * (-7388.84)} = 2^{7.55505} = 188.0602$

sample3: $2^{-\frac{1}{985} * (-7468.29)} = 2^{7.58202} = 191.6088$

\log_2 probabilities on the bigger switchboard corpus...

-12561.5 ../../speech/sample1

-7538.27 ../../speech/sample2

-7938.95 ../../speech/sample3

A larger corpus results in a more positive value of the log2 probability. This shows that the larger corpus results in a better fitting model as it has a higher probability of predicting the future. Furthermore, this results in the perplexities per word decreases. This is due to the larger training corpus gives more training data . The perplexity going down means that the model is less confused on the sample text, and is able to choose from a smaller number of possibilities for each word.

2 textcat.py

textcat.py included in submission.

3 Categorizing with spam detection problem

3.1

Lowest error is 0.09259259..., or an accuracy of 0.90740740... 195 of the files were categorized as gen and 75 were categorized as spam.

textcat was modified so that it automatically calculates the accuracy of categorization when dealing with two training files.

3.2

The value of 0.00035 was used for lambda.

3.3

An error rate of .13148... was observed when our value of lambda was used on the test data.

3.4 Graphs

Graphs are included in the submission. One graph shows a comparison between the length of the test file and its accuracy rate. The second graph shows a comparison between the size of the training corpus with its resulting accuracy rate.

3.5 Increasing training size

Graph is included.

4 Questions

V the size of the vocabulary including *OOV*

4.1

UNIFORM estimate $\hat{p}(z|xy) = 1/V$

ADDL estimate $\hat{p}(z|xy) = \frac{c(xyz)+\lambda}{c(xy)+\lambda V}$

If we mistakenly take V to equal 19,999, then the UNIFORM estimate would be larger than it is suppose to be, since V is the denominator. Having a lower-than-expected denominator would give more weight to smaller, more novel events.

For the ADDL estimate, having V equal 19,999 would result in a similar situation where the estimation would be larger than expected because V is in the denominator and give more weight to novel events. However, we would see less of an impact with ADDL estimation depending on what λ is.

For both, by having V equal a smaller number than it is supposed to be means that the sum of all probabilities would not equal 1.

4.2

Setting $\lambda = 0$ would give the naive historical estimate. This would mean that no smoothing is occurring at all.

Beyond that, if it happens that $c(xy) = 0$, (in other words, we didn't see xy in training) then we get that $\hat{p}(z|xy)$ has no value at all / is undefined.

4.3

If $c(xyz) = c(xyz') = 0$, then:

$$\hat{p}(z|xy) = \frac{\lambda V \hat{p}(z|y)}{c(xy) + \lambda V}$$

$$\hat{p}(z'|xy) = \frac{\lambda V \hat{p}(z'|y)}{c(xy) + \lambda V}$$

If $c(xyz) = c(xyz') = 1$, then:

$$\hat{p}(z|xy) = \frac{1 + \lambda V \hat{p}(z|y)}{c(xy) + \lambda V}$$

$$\hat{p}(z'|xy) = \frac{1+\lambda V \hat{p}(z'|y)}{c(xy)+\lambda V}$$

4.4

BACKOFF ADDL estimate $\hat{p}(z|xy) = \frac{c(xyz)+\lambda V * \hat{p}(z|y)}{c(xy)+\lambda V}$

Increasing lambda will make it so that the trigram probabilities will be more like the corresponding bigram's probability because we're putting less weight on the trigram's count.

5 Other Smoothing

5.1 add-l smoothing

Implemented

5.2 ADDL vs BACKOFF_ADDL

With the same lambda value as in 3.c (0.00035), switching to BACKOFF_ADDL improved the performance. This resulted in an error rate of 0.072222, smaller than our error rate of 0.13148 from 3.c.

Basically running fileprob.py as in Q1, but using backoff_add0.01 as the smoothing method.

-9898.16 ../../speech/sample1 -6048.05 ../../speech/sample2 -6105.56 ../../speech/sample3

Cross Entropy:

sample1: 5.87079

sample2: 6.1841

sample3: 6.1985

Calculating the perplexity per word for each of the sample files:

sample1: $2^{\frac{-1}{1686} * (-9898.16)} = 2^{5.87079} = 58.5174$

sample2: $2^{\frac{-1}{978} * (-6048.05)} = 2^{6.1841} = 72.7109$

sample3: $2^{\frac{-1}{985} * (-6105.56)} = 2^{6.1985} = 73.4422$

As can be seen here, there is also a marked decrease in the perplexity and cross entropy per word for the speech sample files.

For

6 The Long Question

6.1

See code

6.2

See code

6.3

Using the code:

```
python fileprob.py loglinear1 chars-10.txt en.1K
```

Output:

```
Training from corpus en.1K
```

```
.
```

```
Vocabulary size is 30 types including OOV and EOS
```

```
Start optimizing.
```

```
finished one epoch: -2998.76590608
```

```
finished one epoch: -2923.93611832
```

```
finished one epoch: -2884.8817791
```

```
finished one epoch: -2861.08771739
```

```
finished one epoch: -2845.19134998
```

```
finished one epoch: -2833.85038073
```

```
finished one epoch: -2825.36942636
```

```
finished one epoch: -2818.80193777
```

```
finished one epoch: -2813.57736719
```

```
finished one epoch: -2809.33075972
```

```
Finished training on 992 tokens
```

6.4

Running the program with a value of $C = 1$ and maintaining the same lambda of 0.01 as before, we used the following input command:

```
python textcat.py loglinear1 chars-10.txt sp.1K en.1K [included all the dev spanish and english data]
```

It should be noted that [included all the dev spanish and english data] in all future uses refers to the fact that when we had initially coded the program, we had done it in a windows environment.

As a result, a directories to include all files had to be more explicitly written. Therefore, to add the files one has to go directly to the subfolder and add a star in the place of the file. This was done for all file lengths for both english and spanish.

The output resulted in a 75.342 percent accuracy of categorization with 113 categorized as sp.1K and 106 categorized as en.1K. Additionally, and average cross entropy was measured at 4.827. Compared to a AddL smoothing with $\lambda = 1$ which resulted in a cross entropy amount of 5.27 and an accuracy of 89.50

Using a $C = 0.05$, an accuracy of 73.5 percent and an average cross entropy level of 4.847 was measured.

Using a $C = .5$, an accuracy of 74 percent and an average cross entropy level of 4.837 was measured.

Using a $C = 2$, an accuracy of 75.3 percent and an average cross entropy level of 4.812 was measured.

Using a $C = 5$, an accuracy of 75.799 percent and an average cross entropy level of 4.783 was measured.

Using a $C = 8$, an accuracy of 78.54 percent and an average cross entropy of 4.768 was measured.

Using a $C = 10$, an accuracy of 78.082 percent and an average cross entropy of 4.76

Using a $C = 12$, an accuracy of 77.62 percent and an average cross entropy of 4.757 was measured.

Therefore, we will set $C^* = 8$.

Using a learning rate of 0.05 and larger lexicon dimensions....

Using chars-10.txt with the new learning rate: 78.08 percent accuracy with cross entropy of 4.8

Using chars-20.txt with the new learning rate: 81.735 percent accuracy with cross entropy of 4.5448

Using chars-40.txt with the new learning rate: 81.278 percent accuracy with cross entropy 4.383

The higher dimension lexicon yields a higher accuracy percentage.

6.5

Using a learning rate of 0.01 and $C = 8$ with the command: `python textcat.py loglinear_f8 chars-10.txt sp.1K en.1K` [all english and spanish files from dev]

An accuracy of 80.40 percent is measured with an average cross entropy of 4.54. This is an improvement over 78.54 as seen with $C = 8$ with the same hyperparameters and files. As can be seen, cross-entropy decreases and accuracy increases. Additionally, on epoch 10 we have a value of 1.1295 and 1.371 for beta in sp.1K and en.1K, respectively.

6.6

We improved the model by adding both an "ensemble" model with a log-trigram implementation and a recurring word model (vii and vi, respectively).

Testing was only performed on the ensemble model, but both can be performed depending on the training task. To run the ensemble, the command the smoothing command is "loglinear_f_vii". To run the recurring word model, the smoothing command is "loglinear_f_vi".

Additionally, in some instances, to run the ensemble model the learning rate had to be radically reduced to .0001 to prevent overfitting and a gradient descent behavior as opposed to ascent.

Using the command `python textcat.py loglinear_f_vii1 chars-10.txt sp.1K en.1K` [include all english and spanish test files from dev]. For this particular run, $C = 1$ and the learning remained at 0.01

The result was an accuracy of 87.86 percent. A very marked improvement over the previous seen numbers. The Cross entropy was found to be 4.5783. It should be noted that some of the largest (closest to 0) values of F were seen while running this ensemble system.

7 a-priori

When adding the prior probability that $P(\text{spam}) = 1/3$, this obviously means that $P(\text{gen}) = 2/3$. Text categorizing is looking at the probability of the model (whether an email is *gen* or *spam*) given the data (a test email text file), and looking for the data that has the greatest resulting probability.

Using Bayes Theorem, this means that determining the maximum probability out of all of the models given the data =

$$\frac{P(\text{data}|\text{model}) * P(\text{model})}{P(\text{data})}$$

$$= P(\text{data} \mid \text{model}) * P(\text{model})$$

Since $P(\text{data})$ doesn't help us to find the maximum probability. Rather, we use the a-priori that was given to us.

The number is used only at test time, where it's used to help predict the test file.

8 Speech Recognition

8.1

We are trying to maximize the probability of the utterance U given the intentional statement someone was trying to say \vec{w} . In other words, we want to maximize:

$$\begin{aligned} & P(U \mid \vec{w}) \\ &= \frac{P(\vec{w} \mid U)P(U)}{P(\vec{w})}, \text{ by Bayes'.} \end{aligned}$$

We can get $P(\vec{w} \mid U)P(U)$ from the sample file, where the second column provides the value of $\log_2(P(U \mid \vec{w}) = P(\vec{w} \mid U)P(U)$.

We can get $P(U)$ from the trigram model.

8.2

speechrec.py is included in the submission.

8.3 Error Rates

Chosen smoothing method, and why: We tested all of the smoothings (just on one n-gram model) and selected the one that gave us the best error rates. This experiment is described in Table 1. We used the words-10 lexicon.

Table 2 shows the overall error rate on easy and unrestricted for each of the three N-gram models. We are using the smoothing *backoff_add0.1*.

Used overleaf.com to generate LaTeX document.

Smoothing	errorRate [Easy]	errorRate [unrestricted]
Uniform	0.202	0.360
add10	0.187	0.362
add1	0.183	0.355
add0.1	0.166	0.360
add0.01	0.160	0.353
backoff_add10	0.184	0.358
backoff_add1	0.168	0.356
backoff_add0.1	0.156	0.363
backoff_add0.01	0.157	0.356

Table 1: Error Results from speechrec.py, used to decide which smoothing method to pick.

N-Gram	Overall errorRate [Easy]	Overall errorRate [unrestricted]
1-Gram	0.203	0.390
2-Gram	0.163	0.373
3-Gram	0.150	0.362

Table 2: Overall error results for speechrec.py