LINGI2263 – Computational Linguistics

Text Categorization

N-gram language models

1 Introduction

In this assignment you will be asked to estimate and to use probabilistic language models, in particular N-grams, for categorizing blog messages. The task of interest is gender classification from blog messages. In other words, you must be able to predict from a message in a blog whether it has been written by a *male* or a *female*. To do so, you will implement a N-gram extension to the Naive Bayes classifier. For any blog message to classify, you can consider it as a specific sequence of word tokens $w_1, w_2, \ldots, w_n$ and assign it a probability $P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i|h)$, where h is the *history* made of the $N-1$ previous tokens (if any) on the left of w_i. Assuming you have gender specific estimates of such probabilities, respectively $\hat{P}_{Male}$ and $\hat{P}_{Female}$, the decision rule to classify a sequence $w_1, \ldots, w_n$ is simply:

$$\hat{k} = \underset{k \in \{Male, Female\}}{\operatorname{argmax}} \hat{P}_k(w_1, \ldots, w_n) = \underset{k \in \{Male, Female\}}{\operatorname{argmax}} \prod_{i=1}^{n} \hat{P}_k(w_i|h) \qquad (1)$$

In other words, one estimates a separate language model respectively for blogs written by male or female authors and ones classifies a new sequence by computing which of the two language models assigns a larger probability to this sequence. A special case is obtained when the history h is chosen to be empty. This is the 1-gram case since one ignores the left context of any word token w_i. In such a case, the resulting classifier is known as *Naive Bayes*[1]. Such a classifier offers a baseline and the objective of this assignment is to assess to which extent a non-empty history, that is a N value larger than 1, actually improves the predictive accuracy of a blog classifier.

We focus here on a real blog corpus made of 3221 messages. A local copy of this data is available from the course Website:

www.icampus.ucl.ac.be/courses/INGI2263/document/data/BlogGender/.

This corpus was partitioned in 4 text files: `Blogs_M_train.txt`, `Blogs_F_train.txt`, `Blogs_M_test.txt` and `Blogs_F_test.txt`. Each *line* of those files corresponds to a specific *blog message*. To make sure your results will be comparable with ours, you **must use** the two first files for *training* 2 language models, respectively from *male* writers or *female* writers. The 2 last files are used for *testing* your blog classification software: `Blogs_M_test.txt`, respectively `Blogs_F_test.txt`, contains additional blog messages from *male*, respectively *female*, writers. Of course, the male versus female label of each test message is supposed to be *a priori* unknown and you must use the estimated language models to predict those labels according to the decision rule (1). You will make sure that the estimated language models are conveniently smoothed (what is the problem if it is not the case?).

In practice, you are asked to turn in a report with all the elements mentioned in a frame in this document.

[1] Strictly speaking a Naive Bayes classifier also make uses of the *prior* probability of each class but the classes considered here (*Male* versus *Female*) can be assumed *a priori* equally likely.

You are free to use or to adapt existing softwares for this assignment, **provided** that you cite explicitly any software you use. In this case, you should describe the necessary adaptations you had to implement (if any) or any parameter tuning you had to go through. In other words, we would like to know the necessary steps you went through to reuse existing softwares.

The second option is to program some parts or everything yourself. This may be a very good choice as it may be significantly faster than learning and adapting existing softwares. It would also let you go through the bits and pieces of N-gram modeling. Efficient scripting languages (such as PERL, PYTHON, RUBY, *etc*) are particularly convenient for text processing. As a demonstrative example, consider a very short PERL program to construct and to print a vocabulary (or lexicon) of the distinct word types found in a text corpus:

```
while (<>) {
    @tokens = split;
    $lexicon{$_}++ foreach (@tokens);
}
foreach $word (keys %lexicon) { print "$word\n"; }
```

As a good programmer, you surely noticed that the above program has *several implicit preconditions* such as the possible separator(s) used when tokenizing words in a sentence, or the fact that if special tags are present in the input text, they are likely to appear as such in the constructed lexicon. In any case, it should be quite easy (as long as you roughly speak PERL...) to adapt this illustrative example to fit your needs. It should also be quite straightforward to extend the above script (or its equivalent in your favorite scripting language) in order to output the words sorted according to their frequency of occurrence in the text, or to build a lexicon of consecutive word pairs, or even more generally consecutive N-grams. If you follow these suggestions, many of the basic steps of this assignment will already be done.

In any case, you are free to reuse software or to program yourself as long as you tell us what you did. You are not asked to turn in the program sources but a description of your implementation choice(s).

2 Preprocessing

Get the training data (`Blogs_M_train.txt`, `Blogs_F_train.txt`) and glance at it. This first analysis should let you realize that the morphology, syntax, style, *etc* is not highly rigorous. From a statistical language modeling point of view, this is not a critical issue (as a matter of fact, it should illustrate the flexibility of those models). However, this corpus is small (how many word tokens, word types, sentences do you count?) and highly diverse according to many aspects (which ones can you think of?).

Statistical models induce recurrent patterns from some data. For those models to be relevant, the patterns captured should be informative for the task at hand. This observation has some direct consequence on the corpus **tokenization** that you have to perform. The general question is: *how to define the word types on which N-gram models are going to be estimated?* If tokenization is very important here, some traditional form of **lemmatization** (mapping different inflected word types to their unique corresponding lemma) is probably less crucial because blog messages do not tend to use many different inflected forms of the same lemma (it is recommended to check this statement by browsing through the corpus). As a first approximation, you can ignore the problem of lemmatization of inflected forms. Another kind of standardization may however be *useful*, such as mapping `2010-01-15` or `1991` to DATE or mapping all smileys to a single type. Here is a non-exhaustive list of related questions that you should consider.

- How to deal with punctuation marks? Should we distinguish between variants of number of consecutive dots, question or any punctuation marks?
- Do we need to distinguish between lowercase words, uppercase words or partially capitalized words?
- How to deal with numbers, smileys, *etc*?

- *What else?*

Since the corpus is **small** and you want the most relevant part out of it, it is good to make design choices that tend to reduce the lexicon size by mapping several distinct observations to the same type. There is obviously a limit to this reasoning as mapping everything to a single type is always possible but would not result in informative models.

The test sets are assumed to be representative of new data you should categorize by gender, once models are built from the training data only. The construction of the lexicon should thus be based only on the training data. How are you going to deal with the issue that a word type may appear in a test while it was never observed in the training?

Preprocess the training and test data. Build a lexicon from the observed types present in the **training corpus** (consider the 2 training sets, one per gender, as a single corpus for that matter), after some adequate **tokenization** and **standardization**. Report your lexicon size and a table with the 20 most frequent types in this lexicon.
This lexicon is supposed to be fixed and common for all models that you will consider throughout the rest of this assignment.
Report the respective frequency of occurrence of those 20 types in the training corpus. The word `the` is often the most frequent word in an English text. Is it the case here? If not, how many times does it occur in the training and what is its rank in a lexicon sorted by decreasing frequencies (or unigram counts)?

3 Word and N-grams counts

Extract the unigram, bigram and trigram counts from the training corpus. Consider the 2 training sets, one per gender, as a single corpus for that matter. Report a plot of the histogram of those counts. Use a different color for each distinct model order. Both axes in your plot should be in log scale. Do you observe the Zipf's law for unigrams? Is it the same for bigrams or trigrams? Add any comment you would consider relevant from this analysis.

4 N-gram estimation

In this part of the assignment, you should estimate N-gram models from the 2 training sets. Here you should estimate specific models for each gender (and hence rely your estimation on the corresponding training set for each gender but with the common lexicon you designed earlier). Depending on the flexibility of your implementation, you will be able to consider, more or less easily, different model orders, i.e. different values for N. Your implementations (or the software you use) should at least deal with $N \leq 3$. Maximum likelihood (ML) estimation should be straightforward from the counts you already computed before. You should however *smooth* the ML estimates (do you remember why?). You are expected to use *Laplace smoothing* as a baseline (it is extremely simple but offers a poor smoothing). You are expected to use another more advanced smoothing technique (pick the one you prefer) and report comparative results. When implementing a smoothed N-gram model it is **very strongly recommended** to check that the model consistency equation **always** applies.

$$\sum_{w} P(w|h) = 1, \forall h$$

where the sum runs over all[2] word types w in the lexicon and h denotes a history or N-gram context $w_{i-1}w_{i-2}\ldots$ Since running this check on all possible histories may take some prohibitively large computing time (how many such histories would you have to consider as a function of N?), you are expected to run this check at least on a significant fraction of the

[2]Some care need to be taken with the sentence boundary markers `<s>` and `</s>`. Indeed `<s>` should never be predicted, hence the sum over w does not include `<s>` as one possible w. However, `<s>` may very well be part of a useful history h. In contrast, `</s>` is never included in some history since we do not predict after the end of sentence but `</s>` is a possible word w to be predicted, since N-gram models normally predict the end of sentence as well.

observed histories in the training **and** test data[3]. This check should also be repeated when you change the model order.

If the above equation is not satisfied, wrong perplexity results will be reported in your evaluation below (Can you see why? How to artificially increase or decrease the perplexity if the above equation is not fulfilled?).

Estimate N-grams models from the training data with Laplace smoothing. This time, estimate a specific model for each gender. For each model order you consider and for each gender, report the training perplexity and the test perplexities obtained with your models. For each model and a given gender, *e.g.* *Male*, report **separately** the test perplexity computed on the test messages written by the same gender (*e.g.* `Blogs_M_test.txt`) and the other gender (*e.g.* `Blogs_F_test.txt`). Which test perplexity is expected to be larger: when training and testing on the same gender or on opposite genders? **Why?** Argue whether the concrete results confirm your expectations.
Repeat the previous experiments and report the training and test perplexities for various model orders: at least $1 \leq N \leq 3$, ideally $1 \leq N \leq 5$.

Repeat the same experiments while smoothing N-grams either by linear interpolation or using the Kneser-Ney model. Report updated perplexity results.

In each case, always report as well the OOV rate, that is the number of out-of-vocabulary events divided by the number of predictions used to compute the perplexities.
Add any comment you would consider relevant about those comparative results.

5 Categorization of blog messages per gender

Your final task is to use the various models defined in section 4 to actually categorize the test messages contained in `Blogs_M_test.txt` and `Blogs_F_test.txt`. Your classification method should be based on the decision rule (1) but, in order to avoid numerical precision issues, you are invited to sum logarithms of probabilities rather than computing actual products.

Report categorization results in the form of 2 by 2 confusion matrices between predicted gender (*Male* or *Female*) and actual gender (*Male* or *Female*). Each cell of such a matrix should be a percentage and the sum of the 4 cells should be 100 %. Report results for various model orders and for the 2 smoothing techniques you have considered.
What is your conclusion about the optimal model order and the best smoothing technique to be used? Discuss to which extent the perplexity results reported in section 4 are indicative of the categorization results. In other words, are the optimal models in terms of test perplexity also optimal in terms of classification accuracy? Why?
Turn in your assignment as a report in PDF format in due time.

[3] An even more rigorous protocol would not use the test data at this point. It is however important to run this check on some data which was not used to estimate the model (can you figure out why?). One possible way to do so consists in partitioning the training sets into two folds (typically 90%, 10%), estimate a model on the first fold and check it on both folds. Once the consistency checked is performed, a new model is built on the whole training data for a given gender.