CS 224N Final Project: Adaptive^2 Language Models

Hyun Goo Kang (hyungoo@stanford.edu)

March 10, 2011

1 Introduction

To learn a language, one should be exposed to the right environment. A person who used English for his/her entire life would never understand a single Japanese word. Even within the same language, people use the same language very differently from person to person and from occassion to occasion. For instance, you would find English written on a NLP research paper (i.e. this one) very different from Justin Bieber's tweets (i.e. "@alfredoflores why would you ruin it gosh"). Even on the same media, Twitter for example, the language Barack Obama uses is different from what Justin Bieber uses. This gives us an important idea for statistical NLP model: our models would perform better if they're trained on the right domain of data.

In this paper, I used domain-adaptive methods to improve n-gram language models, one of the most commonly used statistical NLP models. In the first part of this project, I trained language models with texts from different domains and verified that domains indeed affect language model performances. It showed that training with the right domain data improves language model performance. Then, I improved the model by building an adaptive hybrid model that combine language models trained on different language domains. This model allows us to use all training data and yet put more weight on the right domain to beat trained-on-whole-data or trained-on-single-domain models. Lastly, in case we do not have well-defined domain information of our training data, I propose using clustering to define domains, build "cluster"-specific models and use our adaptive hybrid model on the clusters to improve language model performance.

2 Prior Works

In [1], Song and Croft showed that a language model trained on domain-specific data (WSJ articles, in their example) performs better than the one trained on general, heterogeneous collections of data (TREC4). The paper suggests using a combination of models, but it did not explore how to combine the models together and how to optimize it. There also has been a numerous works done on language models, and many of the techniques (i.e. n-gram models, smoothing, interpolation, etc.) was adopted from [2].

There has been some prior works on identifying language domains, too. In [3], Dunning suggested a model that identifies languages (i.e. English and French). This is similar to our work of identifying language domains. In [4], Smeaton et al. has clustered research papers from SIGIR conference, clustering them into sets of documents on different topics. Blei, et al. models a document as a mixture of topics, using latent Dirichlet distribution [5], and it could be applied to our model - it will be discussed further in the "Future Works" section.

3 Approach

3.1 Skeleton Language Model

To compare performances of language models trained on different data sets, bigram language model was used as our skeleton model. The role of language model is to predict the likelihood, or naturalness, of language usage [2]. Given a English sentence, for example, it should tell us how "English-like" the sentence

is. Bigram language model is trained on a training data (i.e. many documents or sentences), counts all 2-word-long subsequences - or bigrams - that appear on the data, and builds a probability distribution of bigrams. Given that language usage follows a set of patterns, we extract those patterns in the form of bigrams and use the patterns to predict how "natural" a test text is. N-gram model is one of the most commonly used and effective language models in statistical NLP. To overcome handle unseen words properly, Good-Turing smoothing was added to the model. Its basic framework has been adopted from CS224N's programming assignment framework [7].

3.2 Adaptive[1] Language Model - Hybrid Language Model

Even though domain-related data improves language models more effectively than other data do, we would still like to include all the data into our model; not-too-related data would not represent a language domain most accurately but it could still capture the patterns that our domain-related training data has missed. This can also be seen as optimizing between the following two properties: "more data is better" and "more relevent data is better". Basically, we would like to take all training data into account, but weigh them differently.

To support the "mixing" functionality, I built a hybrid language model that builds multiple bigram models and use them together to predict the likelihood of a given text. It will build separate language models for each of the domain-specific training data and interpolate the models. In the interpoloation process, we can specify weights of each domain-specific models and weigh them differently.

One problem with our hybrid model is that we do not exactly know how much weight to put on each data. To find the optimal weight values, by using validation data. A language model used by Justin Bieber's gmail account, for example, should use a subset of Bieber's e-mails as validataion set and find the optimal weights that performs the best on the validation e-mails. From now on, we will use the optimized weights and "adapt" to Bieber's language usage patterns when we process his future e-mails! Note that we can run cross-validation on his data, too. We can also use hill-climbing methods to find the optimal values more effectively. (These works are left for future work.)

3.3 Adaptive[2] Language Model - Domain Clustering

Unfortunately, we have too many "flavors" of language data out there and we cannot simply train one sub-model for each of them. Then how do we apply our model in the real world? To solve this curse of dimensionality problem, I used k-means clustering, one of the unsupervixed learning covered in the class, to form "clusters" of text documents. My assumption is documents in the same cluster would have similar characteristics and follow similar patterns. As a result, our hybrid language model with higher weight on similar clusters could perform better. In some cases, a well-tuned document clustering could detect language characteristics human cannot easily find and cluster documents better than we do. It could improve our adaptive model even further.

To implement an efficient clustering algorithm, I adopted the techiques used in [4]. First we preprocess our documents using stemming and stopword removal. Then we represent word usage (unigram) with a vector and weight them properly (i.e. BM25 term weighting). To capture the characteristics of a language usage better, we also add other features (i.e. average length of sentences, usage of punctuations, etc.) to our document vector. Then, we run clustering using libraries (i.e. WEKA) to form "domains" of language. This way, we can "adapt" to any kind of language pool, form domains and use our previous Adaptive Language Model on it. We call it Adaptive[2] Language Model.

4 Results

4.1 Language Data and Experiment Setups

To test and evaluate our language models, we collected text data the followings corpora: European Parliament Proceedings Parallel Corpus (EuroParl) [6], Enron e-mail corpus (Enron) [6] and scripts from the

TV show "Friends" (Friends) [7]. The first two data were already cleaned up and processed down to sentence-level by the courtesy of Stanford NLP group. The last data was collected from [7] and divided in into 6 documents, each containing one of its 6 main characters' dialogues. It was then cleaned up (i.e. remove scene descriptions) and processed to the same format as the previous data (i.e. to-lowercase, remove punctuations, etc.). The EuroParl corpus contain fomral, written English, while the Friends corpus contain very casual, spoken English. The Enron corpus contains e-mails which are used for both formal and conversatial uses; we expect the corpus to lie in the middle of the other two corpora. In this project, language models will be trained, validated and tested on the texts from one of, or mix of, these corpora.

To test our language models, I first used perplexity (power of cross-entropy) of our models on test data. Unfortunately, it was not the best metric I. If a language model always returns the same wrong answer, for instance, it's perplexity value would be low while its actual performance is indeed very poor. To overcome this, I ran two different tasks using language model and used the scores we got from the tasks to measure our language model's performance (we could call it task-specific performance).

The first task creates a list of candidates by suffling words within a sentence and asks a language model to find the orignal sentence among the candidates (JumbleTask). If our language model captures a corpus's langauge pattern well, it should give higher probability to the original sentence and be able to find it. JumbleTask's performance was measured by word error rate (WER) between our prediction pick and the original sentence. CS224N's programming assignment framework was used to evaluate this measure.

The second task creates a list of candidates by replacing a word in a sentence with commonly-used Enghlish words (i.e. the, is, that, ..) and lets a language to find the original sentence out of them (GuessTask). The word to be replaced was hand-picked. To measure perforamce, we first let language model make a guess and get the ratio of the original sentence's probablility to the best-guess sentence's probability; higher the ratio, more probability we give to the original sentence and thus better the language model. We ran this task on multiple sentences and calculated an "average" by using geometric mean.

$$(GuessTask\ Score) = (Prob\ Ratio) = \sqrt[N]{\prod_{i=1}^{N} \frac{(likelihood\ of\ the\ correct\ sentence)}{(likelihood\ of\ the\ best\ guess\ sentence)}}$$

The second task is similar to how our language models are used in speech recognition: we need to guess what our user just said. Let's say our speech recognition program just heard a word "half" in the middle of a sentence, but it is not very sure if the word is "half" or "have". If the ratio of probabilities for those words is high (we're more confident on "half"), our language model would vote more strongly for "half" and our recognition software would recognize the sentence more accurately. This task also overcome's one of JumbleTask's limitation: JumbleTask's candidates contains the same set of words, which fails to make use of word frequency information, but this taks's candidates do contain different sets of words. In the following sub-sections, we will evaluate our adaptive language model by running these two tasks.

4.2 Text Domain vs. Language Model Performance

Before building the adaptive models, a set of tests were performed to show that language model performance indeed depends on text domains. I tested the bigram language model on test data selected from Ross's (a character from "Friends" [7]) lines. We measured 1) model's perplexity on 300 sentences, 2) 300 Jumble problems (each containing one correct and 100 randomly-permutated candidates) and 3) 160 Guess problems (each containing ~20 candidates). We've trained five models on five different train data sets from each of the followings: Ross's/Phoebe's/Chandler's lines, Enron e-mails and Europarl documents. The result is on Figure 1. As expected, the model trained on Ross's sentences performed the best in general. Phoebe's and Chandler's, which should have similar characteristics as they're conversational language of people from very similar backgrounds (i.e. age, city and interest), performed nearly as well as Ross's. Enron and Europarl performed poorly compared to the previous three as they should have quite different usage patterns of English

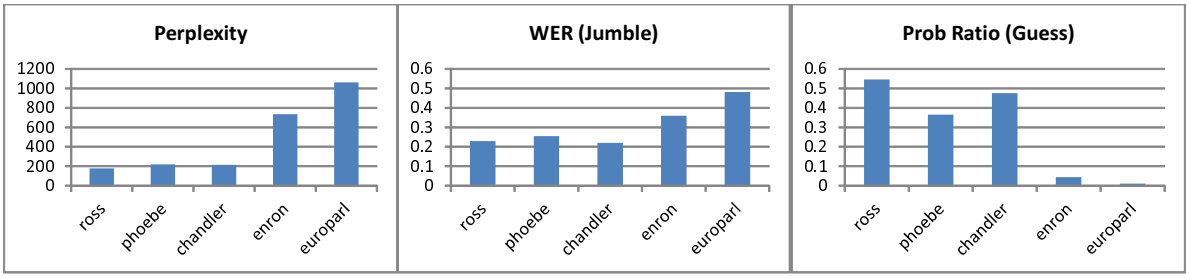

Figure 1: Comaprison of language models trained on different domains of data. Perplexity and WER: lower the better. Ratio: higher the better.

- just looking at a couple of examples, you can tell the difference very easily. As predicted earlier, the Enron model performed better than the Europarl model.

4.3 Evaluation of Adaptive Language Models

To ease the evaluation, we simplified the experiment setting: training data contains sentences from two domains only. This is a reasonable simplification, as we can classify all the documents out there into "related" and "not-related" documents. When constructing a training set, we selected a small number of senteneces from "related" domain and a much larger number of sentences from "not-related" domain. This is also very reasonable, since there are so many different domains of documents out there and they all tend to be very different from each other; only a small subset of language data would be similar to one domain. We first constructed four sets of training data:

```
1) 1K_ENRON : 1000 sentences from Ross's (related) and 80000 from Enron (not-related)
2) 2K_ENRON : 2000 sentences from Ross's (related) and 80000 from Enron (not-related)
3) 1K_EURO : 1000 sentences from Ross's (related) and 80000 from EuroParl (not-related)
4) 1K_EURO : 2000 sentences from Ross's (related) and 80000 from EuroParl (not-related)
```

while Ross's lines were used to test our models. On each of the datasets, we trained our models the following ways:

```
1) Baseline:  trained on the whole data (i.e.  ROSS + ENRON all together)
2) Domain:  trained on the domain-specific data only (ROSS only)
3) Hybrid:  combination of two models trained on the domain-specific / whole data:
  3a) Optimal  :  omnipotent - always chooses the optimal weights at document-level
  3b) Adaptive :  optimize weights on validation data at document level (optimize different
tasks
```

Setting its domain/all weights to 0.0/1.0 or 1.0/0.0, a hybrid model run exactly the same as the Baseline or Domain model. Thus, Optimal should always perform at least as well as both Baseline and Domain. We expect Adaptive model to perform nearly as well as Optimal, beating both Baseline and Domain. There were a couple of design considerations in building the model. First of all, I have tried two different types of hybrid models: combination of DOMAIN & NON-DOMAIN models vs. DOMAIN & ALL models. Both performed similarly, but the latter was slightly better and we chose the Domain & ALL hybrid model. Secondly, I tried optimizing weights just once for all tasks (Once) vs. once for each task (PerTask). The former would be more effective, if the function of optimal weights is task-independent. In most cases, very similar optimal weights were found in JumbleTask and GuessTask. In some cases, they differed a bit (i.e. in 2K_EURO, JumbleTask performed best at 0.0001/0.9999 while GuessTask performed best at 0.001/0.99). To get better results, PerTask optimization was used in the evaulations.

Figure 2 shows our evaluation results. In both JumbleTask and GuessTask, Adaptive model beat Baseline and Domain models. In many cases - especially in GuessTask -, Adaptive models performed almost as well

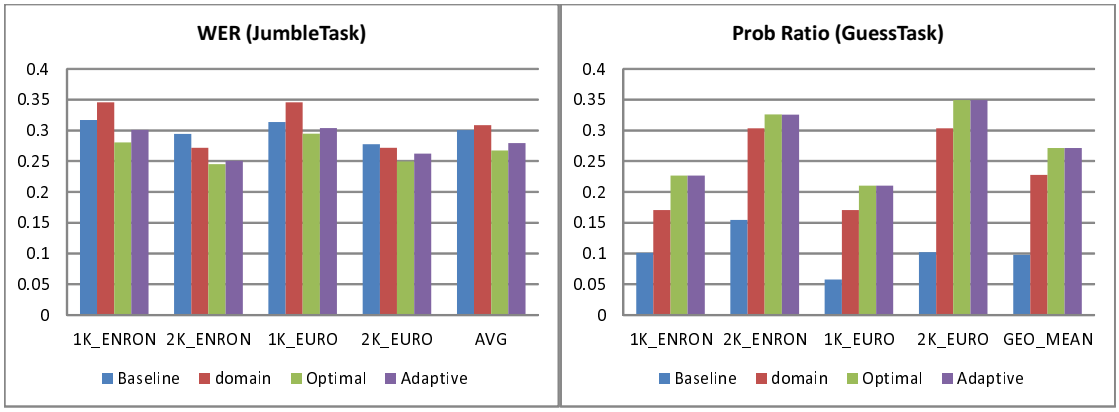

Figure 2: Evaluation of Language Models

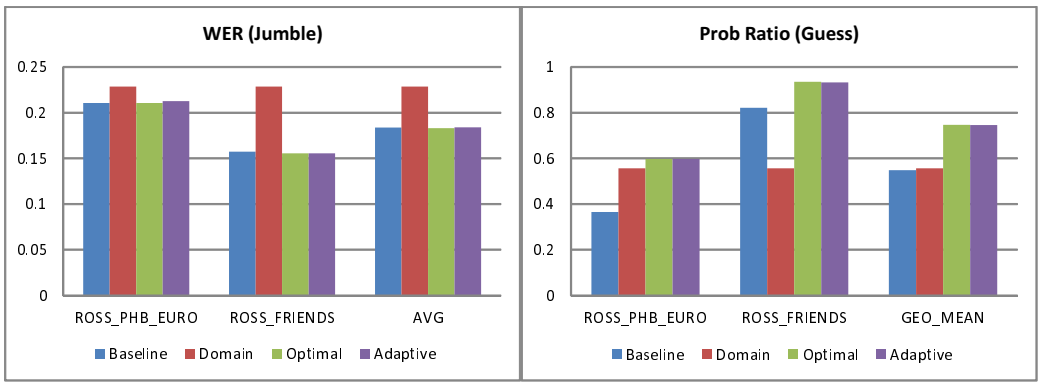

Figure 3: Evaluation of models on mixed data

as Optimal models.

To make things more interesting, I created more interesting data sets. Instead of mixing two of very dissimilar data sets, I mixed more correlated data together:

`5) ROSS_PHB_EURO : 7.5K sentences from Ross's and (7.5K from Phoebe's + 80K from EuroParl)`
`6) ROSS_FRIENDS : 7.5K sentences from Ross's and (37.5K from the rest of the characters from Friends - 7.5K from each of them)`

Surprisingly, our Adaptive models performed better than (in GuessTask) or as nearly as well as (in JumbleTask) Baseline and Domain models. In JumbleTask, hybrid model did not perform any better than Baseline models; yet, our Adaptive model still achieve nearly the same performance level. In GuessTask, things got more interesting. In ROSS_PHB_EURO, Domain performed better than Baseline while Baseline was superior in ROSS_FRIENDS. In both cases, our adaptive model found optimal points (0.3/0.7 and 0.5/0.5) that beat both Baseline and Domain very easily.

4.4 Evaluation of Adaptive^2 Language Models

Unforunately, I ran out of time for this evaluation section. A simple clustering (k-means) code has been implemented, but I had no time to evaluate this model. Considering the previous successes of document clusterings [3,4] and our hybrid models' impressive performance on even very similar data sets, we expect our Adaptive^2 Language Model to perform pretty well. I leave this work for future works.

5 Discussion

5.1 Applications

Our Adaptive^2 Language Model framework can be extended to a various NLP applications. First of all, our domain-adaptive approach can be applied to other statistical NLP models, such as POS tagger named-entity recognizer. This model can also be applied at a different number of levels. For instance, we can use it for general usage (i.e. word processor); we detect a document's domain and optimize our models. It can be applied at task-specific level (i.e. speech recognition on a smartphone), too; cluster language usages of different people (i.e. elder vs. youngters, Caucasian vs. African-American, Texans vs. non-Texans) and tune NLP models based on the smartphone user's past history. Limited evaluation done in this paper suggests that there is a huge potential in our Adaptive^2 Language Model framework.

5.2 Future Works

First of all, we would have to complete the evaluation of the second adaptive model. Once it is done, we should test our adaptive model with more advanced skeleton language models. We may gain a lot with Bigram models, but we may do worse or even better in more sophisticated language models (i.e. that counts POS, named entities separately). Also, we should build more "tasks" to evaluate our models more accurately.

Another improvement we can make is to handle documents that belong to multiple domains. For example, a document could be a mix of English and Spanish or talk about multiple topics (i.e. bioinformatics). Our model would decently work for those documents, but it would perform better if we could extract those multiple domains separately. There was a related work on multiple-topic detection using latent Dirichlet allocation [5].

Note

This project was done for CS221 and CS224N final projects. The 224N report was written from more of NLP perspectives while this report focuses more on general A.I. techniques (i.e. validation, clustering, interpolation, etc.).

References

[1] Song, F., Croft,. W. B., "A General Language Model for Information Retrieval", Research and Development in Information Retrieval, 1999.
[2] Manning, C.D., Schütze, H., Foundations of Statistical Natural Language Processing, 1999.
[3] Dunning, T., "Statistical Identification of Language", 1994.
[4] Smeaton, A. F., et al., "Analysis of papers from twenty-five years of SIGIR conferences: what have we been doing for the last quarter of a century?", 2002
[5] Blei, D. M., et al., "Latent Dirichlet Allocation', 2003.
[6] Stanford NLP Group's Linguistic Data, /afs/ir/data/linguistic-data.
[7] Television Show "Friends" Scripts, www.livesinabox.com/friends.
[8] Klein, D., et al., "CS224N Programming Assignment1", http://cs224n.stanford.edu.