Muhammad Luqman Abdurrohman, Mohamed Elattma
Professor Jeffrey Flanigan
CSE 143 - Natural Language Processing
26 January 2020

Assignment 1: Lab Report

We are tasked with building our own language models derived from Markov's n-gram system of models in order to capture the probabilities of a language's corpus. Essentially, we are given a large dataset of text on which to train our models, from which we calculate the perplexity of our given train, dev, and test datasets to report the performance of each n-gram model - unigram, bigram, and trigram - we implemented. As a result, we have gained a more concrete understanding of n-gram language models as well as gained practical experience in implementing these models with mock real-world data. All in all, we produce nine perplexities for Part 1 of this assignment, from the cross product of the models and the datasets. In general, the higher the perplexity the lower the performance is of a language model on a specific data set, and the lower the perplexity the better the language model is. Intuitively, we expect the perplexity to significantly decrease as we progress from unigram to trigram models.

For Part 2, we want to train parameters of our model on a training set. Tune and optimize the model through our development set and apply smoothing and hyperparameters to lower the perplexity. Then observe the model's performance on new unseen data, or the test set.

To calculate perplexity, we have used the formula:

$perplexity(p; \bar{x}_{1:m}) = 2^{\frac{1}{M}\sum_{i=1}^{m} -log_2 p(\bar{x}_i)} = 2^{\frac{1}{M}\sum_{i=1}^{m}\sum_{j=1}^{n} -log_2 p(\bar{y}_{ij})}$, *where* M *is the number of words in* $\bar{x}_{1:m}$ *and* n *is the # of words in sentence* i *&* $p(\bar{y}_{ij})$ *is* $n-gram\ [i,\ j]$*'s probability.*

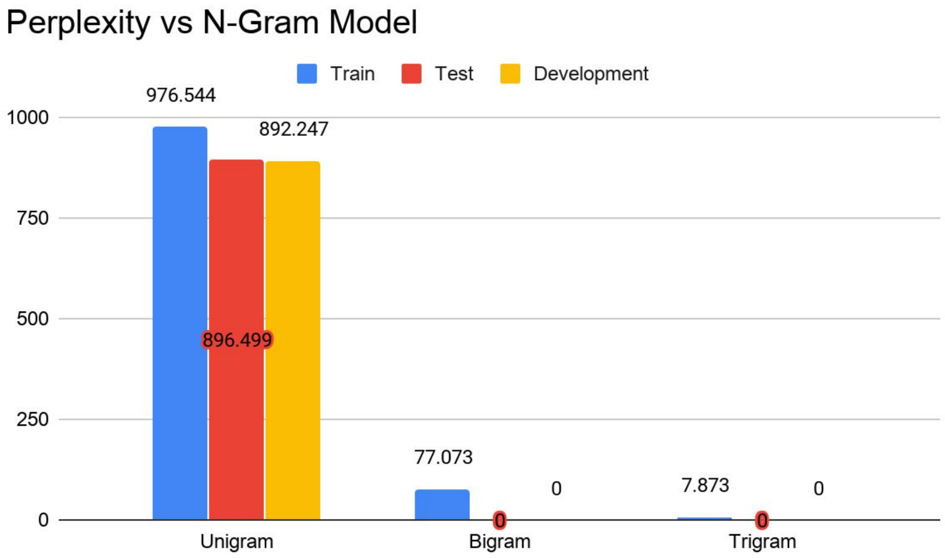
$p(\bar{x}_i)$ is the probability of each sentence on the dataset on which we are evaluating our models, and therefore will differ per model. We further broke down the perplexity calculation. We use the functions listed below to find the probability for each n-gram given the "context" of each word. Then, we use our broken down equation to find the perplexity.

$Unigram : \frac{c(v)}{N}$ *where* N *is the number of unigrams (number of tokens in the train file)*

$Bigram : \frac{c(v'v)}{\sum_{u}^{V} c(v'u)} = \frac{c(v'v)}{c(v')}$

$Trigram : \frac{c(v''v'v)}{\sum_{u}^{V} c(v''v'u)} = \frac{c(v''v'v)}{c(v''v')}$

We derive a vocabulary of tokens from our train dataset with the normalized probability of each token, replacing uncommon tokens (occurs less than 3 times) with $< UNK >$. When evaluating models on datasets, we replace any out-of-vocabulary (OOV) words with this $< UNK >$ token. This takes care of data sparseness and the high estimation variance. We also consider $< START >$ and $< STOP >$ in the context for each model to denote a sentence, but excluded the $< START >$ from the vocabulary and unigram, since they are obsolete. To contrast, the probability of the next token being $< STOP >$ is dependent on the history of tokens and trained vocab, thus $< STOP >$ has a strong expression and is kept for vocab and unigram.

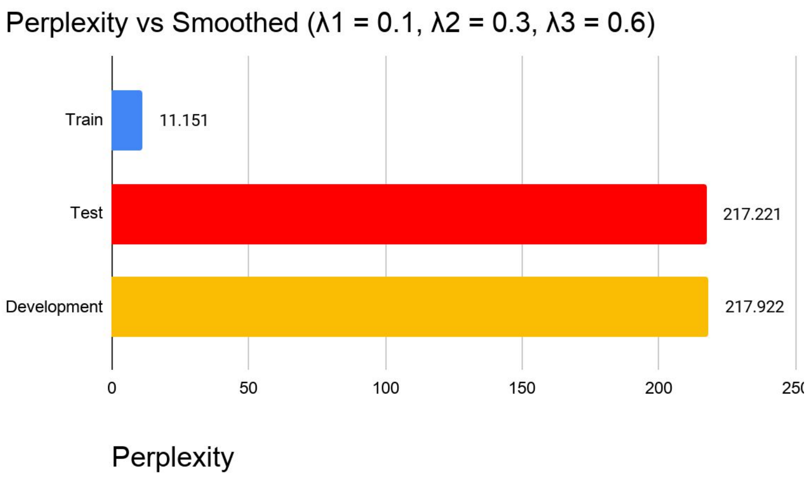


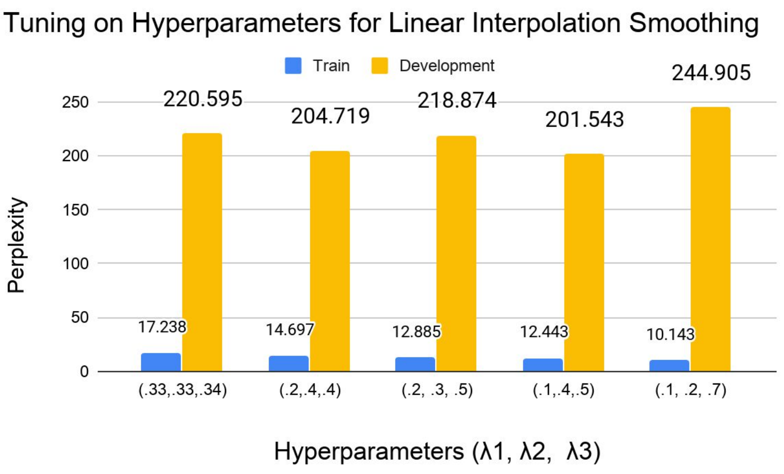
After applying each n-gram model to the data set, we notice in the graph above that the trigram model performs better than bigram, which perform better than unigram. From this, one may assume that as your n-gram length increases, the better the performance of a given language model. This can be justified by the fact that, in this situation, the occurrences of any unique n-gram will decrease, thus boiling down to data sparsity. The more sparse the dataset is, the worse you can model it. In general, bigram and trigram carries more information in their context, thus the models are more expressive. The question should then be asked: what about 4-gram or 5-gram models? It is important to note that, as the n in n-gram models increases, the higher the chance of overfitting on the train dataset, as you can see in the $976.5 \rightarrow 77.0 \rightarrow 7.8$ drop off.

The drawback of this is evaluating the model on non-train data, as evident in our models. We expected that at least one bi- or tri-gram would be in the dev or test datasets which did not occur in the train dataset, thus resulting in a perplexity of 0 for both dev and test datasets for the bigram and trigram cases. To combat this, we implement a linear smoothing technique to gain the benefits of a better perplexity from bigram and trigram models as well as prevent a 0 perplexity output.

It's also interesting to point out that our unigram model performed worst on the train dataset as compared to dev and test datasets. Although this was confusing at first since the model should be biased towards the dataset on which it was trained, we realized that the train dataset is much larger in size than the other two and unigram does not overfit its training data well, so there is a great deal of variance due to chance of the data's structure. However, one can later observe that the bigram and trigram models fit the train dataset well and produce significantly lower perplexity scores as opposed to the dev and test datasets, even in the case of the smoothed trigram model.

During the process of tuning the hyperparameters for smoothing between the n-grams, we stumbled across interesting results. To optimize the weights for each ngram model, we used the dev dataset and found that, out of the weight hyperparameters we tested, (0.1, 0.4, 0.5) produced the lowest perplexity. Furthermore, this set of weights (0.1, 0.4, 0.5) produced a perplexity score of 200.96 on the test dataset, which we only ran once so as to not unfairly overfit it.





We tested the training model on half the training data, and we noticed that the perplexity across all n-models decreased significantly. The perplexity dropped by appropximately 10% for the train data: 816.49, 63.08, 6.55. This is shocking to us because we assumed that if the model is trained on a smaller corpus, then the model will be less accurate on unseen data. The vocabulary is dependent on the train dataset, so we hypothesized that since the vocabulary is smaller, then there will be more unknown tokens in the unseen data. As a result of this, there are more unknown tokens and each unknown token has a larger probability, meaning a smaller perplexity. Also, we believe another reason we are getting lower perplexity is given the specific dataset, the vocabulary saturates coincidentally when combining both halves of the dataset.

To learn more about the model we built, we changed the threshold for $< UNK >$ tokens in the vocab to < 5 tokens, and overall the perplexities decreased, resulting in better performance. The training data perplexity scores were: 803.49, 75.63, 8.60. Again, we assume that the result is stemmed from the fact that there's less variety in tokens so the probability of seeing an unknown word from new data is higher since there are generally more $< UNK >$ tokens and higher probability for those $< UNK >$ tokens. For experimental sakes, I changed the threshold to eight to observe the trend of the perplexity and expectantly, the scores were lower: 654.96, 72.90, 9.40. The vocabulary becomes less unique as the threshold increases, which is a result of overfitting. We theorize as $\lim_{k \to \infty}$ the perplexity decreases, approaching 1. This is because the vocab as well as the dataset to be evaluated will be just $< UNK >$ tokens, giving a probability of 1 for each word and since the log of 1 is 0. Then, $perplexity = 2^0 = 1$. This is one of the reasons perplexity does not perfectly capture how good a language model is as increasing the threshold to lower the perplexity and better optimize the model results in linguistic inaccuracy. We want to be careful when optimizing the model so we don't accidentally overfit it.

The lab enabled us to grasp a strong understanding of n-gram language models. By experimenting on given data and tuning hyperparameters, we observed how the model behaved, so we can further optimize. Nuances we considered include data sparsity, variance of our vocab, special tokens, data overfitting by feature parameters. It is clear that n-gram models are simple generative language models, but this assignment allowed us to construct a strong foundation in linguistic probability modeling.