Summer Research Paper

Will Povell
Baltimore Polytechnic Institute
willipovell@gmail.com

Matt Post
Johns Hopkins University
post@cs.jhu.edu

Frank Ferraro
Johns Hopkins University
ferraro@cs.jhu.edu

Abstract

A review of basic knowledge about the field of computation linguistics and a description of what I've accomplished over the summer.

1 Introduction

The field of computational linguistics concerns using computer science, machine learning, statistics and other related fields to analyze and model language. This can be used to solve many problems, from detecting spam to improving human-computer interaction.

1.1 Natural Language Processing

1.1.1 N-grams

One type of model I studied in depth this summer was the N-gram. N-grams work by assigning a sequence of tokens (words, punctuation, etc.) of length N a probability of occurring based on the number of times that sequence occurs in training data. The probability is equal to

$$P\left(w_i|w_1, w_2, \cdots, w_{i-1}\right) = \frac{c\left(w_1, w_2, \cdots, w_i\right)}{c\left(w_1, w_2, \cdots, w_{i-1}\right)}$$

where $c\left(w_1, w_2, \cdots, w_i\right)$ returns the number of times the sequence $w_1, w_2, \cdots, w_i$ occurred in the training data. The order of an N-gram model is related to how large i is. If i is 1, you have a 1-gram (a unigram). If i is 2 you have a 2-gram (a bigram), and so on. So, for example, a bigram probability would be calculated as follows

$$P\left(w|v\right) = \frac{c\left(v, w\right)}{c\left(v\right)}$$

An example use of an N-gram model would be when you want to predict the next word in a sequence. If I have the sequence "How are", you could use an N-gram trained on other data to try and predict the next word by iterating over your vocabulary and choosing the highest probability $\arg\max_{w \in V} P\left(w \mid \text{How, are}\right)$. This could, for example, be "you" where $P\left(\text{you} \mid \text{How, are}\right) = 0.7$. In this example, the N-gram is a trigram (2-gram) as it uses 2 tokens of context in calculating probability.

1.1.2 Smoothing

Simple N-gram models work as described above, but they can be greatly improved. Higher-order models especially suffer from over-fitting where they become too specialized to the dataset they were trained on, generalizing poorly. Smoothing the N-gram probability distribution can help alleviate this problem by moving probability mass to lessen extremes and prevent counts that are equal to 0.

Additive One method of smoothing is called additive smoothing. This method works by adding a certain amount α to every count. Using this type of smoothing, the probability function is calculated as follows

$$P^*\left(w_i|w_1, w_2, \cdots, w_{i-1}\right) = \frac{c\left(w_1, w_2, \cdots, w_i\right) + \alpha}{c\left(w_1, w_2, \cdots, w_{i-1}\right) + V\alpha}$$

where V is the length of the vocabulary.

Other Interpolations An interesting thing to note about additive smoothing is that it is also just a linear interpolation between a bigram probability distribution and uniform probability distribution and can also be represented (for a bigram model) as follows

$$P^*\left(w|v\right) = \lambda \frac{c\left(v, w\right)}{c\left(v\right)} + \left(1 - \lambda\right) \frac{1}{V}$$

where $0 < \lambda\left(v, \alpha\right) < 1$. For the full derivation of this, see Appendix A.

Other interpolations can also be used. For example, you can interpolate bigram, unigram, and uniform models as follows

$$P(w|v) = \lambda \frac{c(v,w)}{c(v)} + \mu \frac{c(v)}{N} + \theta \frac{1}{V}$$

$$\text{where}$$

$$\lambda + \mu + \theta = 1$$

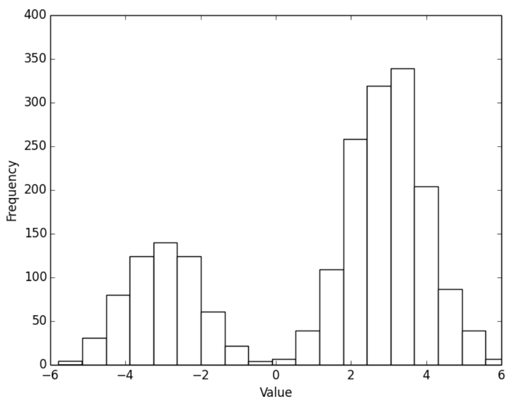
$$0 \leq \lambda, \mu, \theta \leq 1$$

The values for λ, μ, and θ can be varied to maximize the generalization of the model.

The reason that interpolation is useful is it averages multiple models that are all able to partially explain the generative story of the text.

For example, say there are two probability distributions, $\mathcal{F} = N(-3,1)$ and $\mathcal{G} = N(3,1)$, both normal distributions but with different means. There is also a weighted coin that has a 0.3 chance of landing heads up (Bernoulli(.7)). Let X be equal to 2000 data points drawn from the hierarchal model $\mathcal{H}$. For each point, flip the coin and if it's heads, draw a point from $\mathcal{F}$, otherwise draw a point from $\mathcal{G}$

This produces the following histogram



The normalized log-likelihood of X scored with model $\mathcal{F}$ and $\mathcal{G}$ equals

$$\mathcal{F}_{score}(X) = \left(\sum_{x \in X} \log \text{normPDF}(x, -3, 1) \right) / 2000 \approx -14$$

$$\mathcal{G}_{score}(X) = \left(\sum_{x \in X} \log \text{normPDF}(x, 3, 1) \right) / 2000 \approx -7$$

However, when you score the data with the interpolated model you get

$$\sum_{x \in X} \log (0.3 \cdot \text{normPDF}(x, -3, 1) + 0.7 \cdot \text{normPDF}(x, 3, 1))$$

$$\approx -2$$

which is significantly lower. Interpolating multiple N-gram models works in a similar manner to lower normalized log-likelihoods by better describing the generative story of the text.

Maximum Likelihood Estimator When a model is unsmoothed, it is the maximum likelihood estimator (MLE) as it will give the highest probability when run on the data the model was trained. For the full proof of this see Appendix B.

1.1.3 Evaluation

Normalized Log-Likelihood Minuscule probabilities are difficult to deal with due to limits on the number of significant digits in floating point point arithmetic. A way to solve this is to get the log of the probabilities. From there, a multiplication of probabilities becomes an addition of the log-probabilities.

This can be used by N-gram models as a way to gauge similarity between text. If you train an N-gram model on a dataset and then "score" another dataset by adding the log-probabilities of every N-gram in the scoring dataset, you get the log-likelihood. The more negative the likelihood is, the more "surprised" your model is by the text. The log-likelihood can be calculated as follows for the bigram

$$\mathcal{L}(\mathcal{H}) = \sum_{v,w \in V} N(v,w) \log P(w|v)$$

This value can be normalized to make comparisons easier, as follows

$$\mathcal{L}_{\mathcal{N}}(\mathcal{H}) = \frac{\mathcal{L}(\mathcal{H})}{T}$$

where T is the number of tokens in the held-out data.

An example usage of this would be to train a model on email spam, and then calculate the log-likelihood of another email. The closer the email's log-likelihood is to zero, the more likely that it is spam.

Maximization When tuning the amount of smoothing for a model, we often want to maximize the log-likelihood on a held-out dataset. This can be done using the derivative of the log-likelihood and gradient descent.

The derivative of an additively smoothed bigram log-likelihood can be derived as follows

$$P(\mathcal{H}) = \sum_{v,w \in V} N(v,w) \log P^*(w|v) \quad (1)$$

$$= \sum_{v,w \in V} N(v,w) \log \frac{c(v,w)+\alpha}{c(v)+V\alpha} \quad (2)$$

$$= \sum_{v,w \in V} N(v,w) \log (c(v,w)+\alpha) - \log (c(v)+V\alpha) \quad (3)$$

$$P'(\mathcal{H}) = \sum_{v,w \in V} N(v,w) \left(\frac{1}{c(v,w)+\alpha} - \frac{V}{c(v)+V\alpha} \right)$$

From there, log-likelihood can be maximized using gradient descent, as described as follows

$$d = \text{precision}$$
$$\epsilon = \text{step size}$$
$$\mathcal{G}(x,f) = \begin{cases} \mathcal{G}(x+\epsilon \cdot f'(x), f) & \text{if } |x-\epsilon \cdot f'(x)| > d \\ x & \text{else} \end{cases}$$

1.1.4 Math

NLP Distributions and Properties Language models have several constraints. The largest of these is that language must be valid probability distributions, with all probabilities between 0 and 1 and summing to 1.

$$\forall w \in V : 0 \leq P(w) \leq 1 \quad (1)$$
$$\sum_{w \in V} P(w) = 1 \quad (2)$$

Additionally, the marginal count will always equal the sum of the individual counts

$$c(v) = \sum_{w \in V} c(v,w)$$

Lagrange Multipliers The method of Lagrange multipliers is a way to find extrema of a function bounded by equality constraints. In formal terms the optimization problem is stated as

maximize $f(x)$
subject to $g(x) = c$

Lagrange multipliers are central to the MLE proof in Appendix B.

$$\Lambda(x,\lambda) = f(x) + \lambda(g(x) - c)$$

Entropy and Cross-Entropy Went over these and I have notes, will probably just need to pull up the wiki page to refresh myself

Perplexity Same as above

KL Similarity Same as above

XEnt I don't believe we went over this. Related to Cross-Entropy

2 Methods & Results

2.1 Python Implementation

TwitterKov details

2.2 Performance Comparisons

list vs dict vs trie

2.3 ACL 2014

Summary of papers, new things I learned about

3 Summary

4 Acknowledgements

A Additive Smoothing as a Linear Interpolation

$$\begin{aligned}
P^*(w|v) &= \frac{c^*(v,w)}{\sum_{w`\in V} c^*(v,w)} && (3)\\
&= \frac{c(v,w)+\theta}{\sum_{w`\in V}(c(v,w)+\alpha)} && (4)\\
&= \frac{c(v,w)+\alpha}{c(v)+V\alpha} && (5)\\
&= \frac{c(v,w)}{c(v)+V\alpha}+\frac{\alpha}{c(v)+V\alpha} && (6)\\
&= \frac{c(v)}{c(v)+V\alpha}\frac{c(v,w)}{c(v)}+\frac{1}{V}\frac{V\alpha}{c(v)+V\alpha} && (7)\\
&= \frac{c(v)}{c(v)+V\alpha}\frac{c(v,w)}{c(v)}+\left(1-\frac{c(v)}{c(v)+V\alpha}\right)\frac{1}{V} && (8)\\
&= \lambda\frac{c(v,w)}{c(v)}+(1-\lambda)\frac{1}{V} && (9)
\end{aligned}$$

B Derivation of Maximum Likelihood Estimator

Below I have shown the derivations for the Maximum Likelihood Estimators (MLEs) of both unigram and bigram models.

From the derivations below, it can be extrapolated that the MLE of an n-gram model would be

$$\phi_{t_1,t_2,\cdots,t_n} = \frac{c(t_1,t_2,\cdots,t_n)}{c(t_1,t_2,\cdots,t_{n-1})}$$

B.1 Unigram

Given n tokens

$$y_0, y_1, y_2, \cdots, y_n$$

With a vocabulary set

$$V = \{v_0, v_1, v_2, \cdots v_m\}$$

Whose generative story is:
From probability vector θ draw

$$y_i \mid \theta \overset{i.i.d.}{\sim} \text{cat}(\theta)$$

The maximum likelihood estimator of θ is the probability distribution ϕ that maximizes the log-likelihood of observed data, calculated with

$$L(\phi) = \sum_{v\,\in\,V} c(v)\log\phi_v$$

To be derived is that the maximum likelihood estimator is

$$\phi_i = \frac{c(i)}{n}$$

To do this, you maximize the log-likelihood function, varying ϕ (restricted to valid probability distributions). This can be done with the method of Lagrange multipliers.

$$\max_{\phi}\ L(\phi)\ \text{ s.t. } \sum_{v\in V}\phi_v = 1$$

$$\Lambda(\phi,\lambda) = L(\phi) - \lambda\left(\sum_{v\in V}\phi_v - 1\right)$$

The partial derivative of Λ with respect to ϕ_i for any type i is

$$\begin{aligned}
\frac{\partial}{\partial\phi_i}\Lambda &= L'(\phi) - \frac{\partial}{\partial\phi_i}\lambda\left(\sum_{v\in V}\phi_v - 1\right)\\
&= L'(\phi) - \lambda\\
&= \frac{\partial}{\partial\phi_i}\left(\sum_{v\in V} c(v)\log\phi_v\right) - \lambda\\
&= \frac{c(i)}{\phi_i} - \lambda
\end{aligned}$$

This can be set to 0 and be solved for ϕ_i

$$\begin{aligned}
\frac{c(i)}{\phi_i} - \lambda &= 0\\
\frac{c(i)}{\phi_i} &= \lambda\\
\phi_i &= \frac{c(i)}{\lambda}
\end{aligned}$$

The partial derivative of Λ with respect to λ is

$$\frac{\partial}{\partial\lambda}\Lambda = 1 - \sum_{v\in V}\phi_v$$

This can be set to 0 and solved for λ, substituting ϕ_v for $\frac{c(v)}{\lambda}$.

$$\begin{aligned}1-\sum_{v\in V}\phi_v &= 0\\ 1-\sum_{v\in V}\frac{c(v)}{\lambda} &= 0\\ \sum_{v\in V}\frac{c(v)}{\lambda} &= 1\\ \lambda = \sum_{v\in V} c(v) &= n\end{aligned}$$

It then becomes clear that the maximum likelihood estimator ϕ is

$$\phi_i = \frac{c(i)}{n}$$

B.2 Bigram

Given n tokens

$$y_0, y_1, y_2, \cdots, y_n$$

With a vocabulary set

$$V = \{v_0, v_1, v_2, \cdots, v_m\}$$

Whose generative story is:
From probability vector θ draw

$$y_i \mid y_{i-1}, \theta \overset{i.i.d.}{\sim} \text{cat}(\theta_{i-1})$$

The maximum likelihood estimator of θ is the probability distribution ϕ that maximizes the log-likelihood of observed data, calculated with

$$L(\phi) = \sum_{v,w\in V} c(v,w)\log\phi_{v,w}$$

To be derived is that the maximum likelihood estimator is

$$\phi_{i,j} = \frac{c(i,j)}{c(i)}$$

To do this, you maximize the log-likelihood function, varying ϕ (restricted to valid probability distributions). This can be done with the method of Lagrange multipliers.

$$\begin{aligned}&\max_{\phi}\ L(\phi)\ \text{ s.t. } \forall\, v\in V : \sum_{w\in V}\phi_{v,w} = 1\\ &\vec{\lambda} = (\lambda_1, \lambda_2, \cdots, \lambda_m)\\ &\vec{G} = \left(\sum_{w\in V}\phi_{v_1,w} - 1, \sum_{w\in V}\phi_{v_2,w} - 1, \cdots, \sum_{w\in V}\phi_{v_m,w} - 1\right)\\ &\Lambda(\phi,\lambda) = L(\phi) - \vec{\lambda}\cdot\vec{G}\end{aligned}$$

The partial derivative of Λ with respect to $\phi_{i,j}$ for any bigram (i,j) is

$$\begin{aligned}\frac{\partial}{\partial\phi_{i,j}}\Lambda &= L'(\phi) - \frac{\partial}{\partial\phi_{i,j}}\vec{\lambda}\cdot\vec{G}\\ &= L'(\phi) - \lambda_i\\ &= \frac{\partial}{\partial\phi_{i,j}}\left(\sum_{v,w\in V} c(v,w)\log\phi_{v,w}\right) - \lambda_i\\ &= \frac{c(i,j)}{\phi_{i,j}} - \lambda_i\end{aligned}$$

This can be set to 0 and be solved for $\phi_{i,j}$

$$\begin{aligned}\frac{c(i,j)}{\phi_{i,j}} - \lambda_i &= 0\\ \frac{c(i,j)}{\phi_{i,j}} &= \lambda_i\\ \phi_{i,j} &= \frac{c(i,j)}{\lambda_i}\end{aligned}$$

The partial derivative of Λ with respect to λ_i is

$$\frac{\partial}{\partial\lambda_i}\Lambda = 1 - \sum_{w\in V}\phi_{v_i,w}$$

This can be set to 0 and solved for λ, substituting $\phi_{v_i,w}$ for $\frac{c(v_i,w)}{\lambda_i}$.

$$\begin{aligned}1-\sum_{w\in V}\phi_{v_i,w} &= 0\\ 1-\sum_{w\in V}\frac{c(v_i,w)}{\lambda_i} &= 0\\ \sum_{w\in V}\frac{c(v_i,w)}{\lambda_i} &= 1\\ \lambda_i = \sum_{w\in V} c(v_i,w) &= c(v_i)\end{aligned}$$

It then becomes clear that the maximum likelihood estimator ϕ is

$$\phi_{i,j} = \frac{c(i,j)}{c(i)}$$