Natural Language Processing

Statistical Inference: n-grams

Updated 4/2018
Statistical Inference

- Statistical Inference consists of taking some data (generated in accordance with some unknown probability distribution) and then making some inferences about this distribution.

- We will look at the task of **language modeling** where we predict the next word given the previous words.

- Language modeling is a well studied Statistical inference problem.
Applications of LM

- Speech recognition
- Optical character recognition
- Spelling correction
- Handwriting recognition
- Statistical Machine Translation
- “Shannon Game”
“Shannon Game” (Shannon, 1951)

- Predict the next word given $n-1$ previous words.
- Past behaviour is a good guide to what will happen in the future as there is regularity in language.
- Determine the probability of different sequences from a training corpus.
Bins: Forming Equivalence Classes

- Reliability vs. Discrimination
  - We try to predict the target feature on the basis of various classificatory features for the equivalence classes.
  - More bins give greater *discrimination*.
  - Too much discrimination may not leave sufficient training data in many bins. Thus, a statistically *reliable* estimate cannot be obtained.
Reliability vs discrimination - example

Inferring $p(x,y)$ using 20 samples from $p$

- Original distribution
- Sampling into 1600 bins
- Sampling into 64 bins
Predicting the next word is attempting to estimate the probability function: $P(w_n|w_1,\ldots,w_{n-1})$.

**Markov Assumption:** only the $n$-1 previous words affect the next word. (n-1)th Markov Model or n-gram.

- n=2,3,4 gram models are called bigram, trigram and four-gram models.
- We construct a model where all histories that have the same last $n$-1 words are placed in the same equivalence class (bin).
Problems with n-grams

- Sue swallowed the large green ______ . Pill, frog, car, mountain, tree?
  - Knowing that Sue swallowed helps narrow down possibilities. How far back do we look?
- For a Vocabulary of 20,000, the number of parameters that should be estimated
  - bigram model \((2 \times 10^4)^2 = 400\) million
  - trigram model \((2 \times 10^4)^3 = 8\) trillion
  - four-gram model \((2 \times 10^4)^4 = 1.6 \times 10^{17}!!!\)
- But other methods of forming equivalence classes are more complicated.
  - Stemming, grouping into semantic classes…
Building n-gram Models

- Data preparation: Decide training corpus, remove punctuations, sentence breaks keep them or throw them?
- Create equivalence classes and get counts on training data falling into each class.
- Find statistical estimators for each class.
Statistical Estimators

- Derive a good probability estimate for the target feature based on training data. From n-gram data $P(w_1,..,w_n)$ we will predict $P(w_n|w_1,..,w_{n-1})$

$$P(w_n|w_1,..,w_{n-1})=P(w_1,..,w_{n-1}w_n)/ P(w_1,..,w_{n-1})$$

- It is sufficient to get good estimates for probability distributions of n-grams.
Statistical Estimation Methods

- Maximum Likelihood Estimation (MLE)
- Smoothing:
  - Laplace’s
  - Lidstone’s and Jeffreys-Perks’ Laws
- Validation:
  - Held Out Estimation
  - Cross Validation
- Good-Turing Estimation
Maximum Likelihood Estimation (MLE)

- It is the choice of parameter values which gives the highest probability to the training corpus.
  
  \[ P_{\text{MLE}}(w_1, \ldots, w_n) = \frac{C(w_1, \ldots, w_n)}{N}, \]

  where \( C(w_1, \ldots, w_n) \) is the frequency of n-gram \( w_1, \ldots, w_n \) in the text.

- \( N \) is the total number of n-gram counts.

- \( P_{\text{MLE}}(w_n | w_1, \ldots, w_{n-1}) = \frac{C(w_1, \ldots, w_n)}{C(w_1, \ldots, w_{n-1})} \)
MLE: Problems

- Problem of Sparseness of data. Vast majority of the words are very uncommon (Zipf’s Law). Some bins may remain empty or contain too little data.
- MLE assigns 0 probability to unseen events.
- We need to allow for non-zero probability for events not seen in training.
Discounting or Smoothing

- Decrease the probability of previously seen methods to leave a little bit of probability for previously unseen events.
Laplace’s Law (1814; 1995)

- **Adding One** Process:
  \[ P_{\text{LAP}}(w_1,\ldots,w_n) = \frac{C(w_1,\ldots,w_n) + 1}{N + B} \]
  where \( C(w_1,\ldots,w_n) \) is the frequency of n-gram \( w_1,\ldots,w_n \) and \( B \) is the number of bins training instances are divided into.

- Gives a little bit of the probability space to unseen events.

- This is the Bayesian estimator assuming a uniform prior on events.
Laplace’s Law: Problems

- For sparse sets of data over large vocabularies, it assigns too much of the probability space to unseen events.
Lidstone’s Law and The Jeffreys-Perks Law

- \( P_{\text{LID}}(w_1, \ldots, w_n) = (C(w_1, \ldots, w_n) + \lambda)/(N + B\lambda) \), where \( C(w_1, \ldots, w_n) \) is the frequency of n-gram \( w_1, \ldots, w_n \) and \( B \) is the number of bins training instances are divided into, and \( \lambda > 0 \).

**Lidstone’s Law**

- If \( \lambda = 1/2 \), Lidstone’s Law corresponds to the maximization by MLE and is called the **Expected Likelihood Estimation** (ELE) or the **Jeffreys-Perks Law**.
Lidstone’s Law: Problems

- We need a good way to guess $\lambda$ in advance.
- Predicts all unknown events to be equally likely.
- Discounting using Lidstone’s law always gives probability estimates linear in the MLE frequency.
- This is not a good match to the empirical distribution at low frequencies.
Validation

- How do we know how much of the probability space to “hold out” for unseen events? (Choosing a $\lambda$)

- **Validation**: Take further text (from the same source) and see how often the bigrams that appeared $r$ times in the training text tend to turn up in the further text.
Held out Estimation

- Divide the training data into two parts.
- Build initial estimates by doing counts on one part, and then use the other pool of held out data to refine those estimates.
Held Out Estimation (Jelinek and Mercer, 1985)

- Let $C_1(w_1,..,w_n)$ and $C_2(w_1,..,w_n)$ be the frequencies of the $n$-grams $w_1,..,w_n$ in training and held out data, respectively.

- Let $r$ be the frequency of an $n$-gram, let $N_r$ be the number of bins that have $r$ training instances in them.

- Let $T_r$ be the total count of $n$-grams of frequency $r$ in further data, then their average frequency is $T_r / N_r$.

- An estimate for the probability of one of these $n$-grams is: $P_{ho}(w_1,..,w_n) = T_r / (N_r T)$, where $T$ is the number of $n$-gram instances in the held out data.
Pots of Data for Developing and Testing Models

- Training data (80% of total data)
- Held Out data (10% of total data).
- Test Data (5-10% of total data).
- Write an algorithm, train it, test it, note things it does wrong, revise it and repeat many times.
- Keep development test data and final test data as development data is “seen” by the system during repeated testing.
- Give final results by testing on n smaller samples of the test data and averaging.
Cross-Validation (Deleted Estimation) (Jelinek and Mercer, 1995)

- If total amount of data is small then use each part of data for both training and validation - cross-validation.

- Divide data into approximately equal parts 1 and 2. In one model use 1 as the training data and 2 as the held out data. In another model use 2 as training and 1 as held out data.
Cross-Validation (Deleted Estimation) (Jelinek and Mercer, 1995)

- Do a weighted average of the two:
  \[ P_{\text{del}}(w_1,..,w_n) = \frac{(T_{r12} + T_{r21})}{T(N_{r1} + N_{r2})} \]
  where \( C(w_1,..,w_n) = r \), \( N_{r1} \) is the number of n-grams in part a with \( C_a(w_1,..,w_n) = r \) and \( T_{r1b} \) is the total occurrences of these ngrams from part a in the b’th part.

- Dividing the corpus in half at the middle is risky (why?) Better to do something like splitting into even and odd sentences.
Deleted Estimation: Problems

- It overestimates the expected frequency of unseen objects, while underestimating the expected frequency of objects that were seen once in the training data.

- *Leaving-one-out* (Ney et. Al. 1997) Data is divided into $N$ sets and the hold out method is repeated $N$ times.
Good-Turing Estimation

- Determines probability estimates of items based on the assumption that their distribution is binomial.

- Works well in practice even though distribution is not binomial.

- \( P_{GT} = \frac{r^*}{N} \) where, \( r^* \) can be thought of as an adjusted frequency given by \( r^* = \frac{(r+1)E(N_{r+1})}{E(N_r)}. \)
Good-Turing cont.

- In practice, one approximates $E(N_r)$ by $N_r$, so $P_{GT} = (r+1) N_{r+1}/(N_r N)$.

- The Good-Turing estimate is very good for small values of $r$, but fails for large values of $r$. Solution – re-estimation!
Good-Turing - rejustified

- Take each of the \( N \) training words out in turn, to produce \( N \) training sets of size \( N-1 \), held-out of size 1. (AKA LOOCV).
- What fraction of held-out word (tokens) are unseen in training?
  - \( \frac{N_1}{N} \)
- What fraction of held-out words are seen \( r \) times in training?
  - \( \frac{(r+1)N_{r+1}}{N} \)
- So, in the future we expect \( \frac{(r+1)N_{r+1}}{N} \) of the words to be those with training count \( r \).
- There are \( N_r \) words with training count \( r \), so each should occur with probability:
  - \( \frac{(r+1)N_{r+1}}{N} N_r \)
- …or expected count \( r^* = \frac{(r+1)N_{r+1}}{N_r} \)
Estimated frequencies in AP newswire
(Church & Gale 1991)

<table>
<thead>
<tr>
<th>$r=f_{MLE}$</th>
<th>$f_{emprical}$</th>
<th>$f_{Lap}$</th>
<th>$f_{del}$</th>
<th>$f_{GT}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000027</td>
<td>0.000137</td>
<td>0.000037</td>
<td>0.000027</td>
</tr>
<tr>
<td>1</td>
<td>0.4480</td>
<td>0.000274</td>
<td>0.396</td>
<td>0.446</td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
<td>0.000411</td>
<td>1.24</td>
<td>1.26</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
<td>0.000548</td>
<td>2.23</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>3.23</td>
<td>0.000685</td>
<td>3.22</td>
<td>3.24</td>
</tr>
<tr>
<td>5</td>
<td>4.21</td>
<td>0.000822</td>
<td>4.22</td>
<td>4.22</td>
</tr>
<tr>
<td>6</td>
<td>5.23</td>
<td>0.000959</td>
<td>5.20</td>
<td>5.19</td>
</tr>
<tr>
<td>7</td>
<td>6.21</td>
<td>0.00109</td>
<td>6.21</td>
<td>6.21</td>
</tr>
<tr>
<td>8</td>
<td>7.21</td>
<td>0.00123</td>
<td>7.18</td>
<td>7.24</td>
</tr>
</tbody>
</table>
Discounting Methods

- Obtain held out probabilities.
- Absolute Discounting: Decrease all non-zero MLE frequencies by a small constant amount and assign the frequency so gained over unseen events uniformly.
- Linear Discounting: The non-zero MLE frequencies are scaled by a constant slightly less than one. The remaining probability mass is distributed among unseen events.
Combining Estimators

- Combining multiple probability estimates from various different models:
  - Simple Linear Interpolation
  - Katz’s backing-off
  - General Linear Interpolation
Simple Linear Interpolation

- Solve the sparseness in a trigram model by mixing with bigram and unigram models.
- Combine linearly: Termed linear interpolation, finite mixture models or deleted interpolation.
- \[ P_{li}(w_n|w_{n-2},w_{n-1}) = \lambda_1 P_1(w_n) + \lambda_2 P_2(w_n|w_{n-1}) + \lambda_3 P_3(w_n|w_{n-1},w_{n-2}) \] where \( 0 \leq \lambda_i \leq 1 \) and \( \sum_i \lambda_i = 1 \)
- Weights can be picked automatically using Expectation Maximization.
Katz’s Backing-Off

- Different models are consulted depending on their specificity.
- Use $n$-gram probability when the $n$-gram has appeared more than $k$ times ($k$ usu. = 0 or 1).
- If not, “back-off” to the $(n-1)$-gram probability
- Repeat till necessary.
General Linear Interpolation

- Make the weights a function of history:
  \[ P_{li}(w|h) = \sum_{i=1}^{k} \lambda_i(h) P_i(w|h) \]
  where \( 0 \leq \lambda_i(h) \leq 1 \) and \( \sum \lambda_i(h) = 1 \)

- Training a distinct set of weights for each history is not feasible. We need to form equivalence classes for the \( \lambda \)'s.
Language models for Austen

- Trained on Jane Austen novels.
- Tested on ‘persuasion’;
- Uses Katz backoff with Good Turing smoothing.

|         | $P(she|h)$ | $P(was|h)$ | $P(inferior|h)$ | $P(to|h)$ | $P(both|h)$ | $P(sisters|h)$ | $Product$   |
|---------|------------|------------|-----------------|------------|-------------|----------------|--------------|
| Unigram | 0.011      | 0.015      | 0.00005         | 0.032      | 0.0005      | 0.0003         | $3.96 \times 10^{-17}$ |
| Bigram  | 0.00529    | 0.1219     | 0.0000159       | 0.183      | 0.00049     | 0.00372        | $3.14 \times 10^{-15}$ |
| $n$ used| 2          | 2          | 1               | 2          | 2           | 2              |              |
| Trigram | 0.00529    | 0.0741     | 0.0000162       | 0.183      | 0.000384    | 0.00323        | $1.44 \times 10^{-15}$ |
| $n$ used| 2          | 3          | 1               | 2          | 2           | 2              |              |
Idea: replace the backoff to a unigram probability with a backoff to another kind of probability. Why?

Unigram models do not distinguish words that are very frequent but only occur in a restricted set of contexts, e.g. San Francisco from words which are less frequent, but occur in many more contexts. The latter may be more likely to finish up an unseen bigram:

I can’t see without my reading ______.

glasses is more probable here, but less frequent than Francisco, which almost always occurs after San.

The absolute discounting model picks Francisco, because it is the word with the higher unigram probability.
Continuation Probability

- Kneser-Ney pays attention to the number of histories a word occurs in.

\[ P_{\text{continuation}}(w_i) = \frac{| \{ w_{i-1} : C(w_{i-1}w_i) > 0 \} |}{\sum_{w_i} | \{ w_{i-1} : C(w_{i-1}w_i) > 0 \} |} \]

- Numerator: the number of word types seen to precede \( w_i \)
- Denominator: sum of numerator over all word types \( w_i \)
- A very frequent word like *Francisco* occurring only in one context (*San*) will have a very low continuation probability.
- Use this continuation probability in backoff or in interpolation
More data is always good

Chen, Goodman 98