SI485i : NLP

Set 12
Features and Prediction
What is NLP, really?

• Many of our tasks boil down to finding intelligent **features** of language.

• We do lots of machine learning over **features**

• NLP researchers also use linguistic insights, deep language processing, and semantics.
  • But really, semantics and deep language processing ends up being shoved into **feature representations**
What is a feature?

- **Features** are elementary pieces of evidence that link aspects of what we observe \(d\) with a category \(c\) that we want to predict.

- A feature has a (bounded) real value: \(f: C \times D \rightarrow R\)
Naïve Bayes had features

• Bigram Model
  • The features are the two-word phrases
  • “the dog”, “dog ran”, “ran out”
  • Each feature has a numeric value, such as how many times each bigram was seen.

• You calculated probabilities for each feature.
  • These are the feature weights

• \[ P(d \mid \text{Dickens}) = P(\text{“the dog”} \mid \text{dickens}) \times P(\text{“dog ran”} \mid \text{dickens}) \times P(\text{“ran out”} \mid \text{dickens}) \]
What is a feature-based model?

- Predicting class $c$ is dependent solely on the features $f$ taken from your data $d$.

- In author prediction
  - Class $c$: “Dickens”
  - Data $d$: a document
  - Features $f$: the n-grams you extract

- In sentiment classification
  - Class $c$: “negative sentiment”
  - Data $d$: a tweet
  - Features $f$: the words
Features appear everywhere

- Distributional learning.
- “drink” is represented by a vector of feature counts.

<table>
<thead>
<tr>
<th>Object</th>
<th>Count</th>
<th>PMI assoc</th>
<th>Object</th>
<th>Count</th>
<th>PMI assoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>bunch beer</td>
<td>2</td>
<td>12.34</td>
<td>wine</td>
<td>2</td>
<td>9.34</td>
</tr>
<tr>
<td>tea</td>
<td>2</td>
<td>11.75</td>
<td>water</td>
<td>7</td>
<td>7.65</td>
</tr>
<tr>
<td>Pepsi</td>
<td>2</td>
<td>11.75</td>
<td>anything</td>
<td>3</td>
<td>5.15</td>
</tr>
<tr>
<td>champagne</td>
<td>4</td>
<td>11.75</td>
<td>much</td>
<td>3</td>
<td>5.15</td>
</tr>
<tr>
<td>liquid</td>
<td>2</td>
<td>10.53</td>
<td>it</td>
<td>3</td>
<td>1.25</td>
</tr>
<tr>
<td>beer</td>
<td>5</td>
<td>10.20</td>
<td>&lt;SOME AMOUNT&gt;</td>
<td>2</td>
<td>1.22</td>
</tr>
</tbody>
</table>

- The words in the grammatical object make up a vector of counts. The words are the features and the counts/PMI scores are the weights.
Feature-Based Model

- Decision 1: what features do I use?
- Decision 2: how do I weight the features?
- Decision 3: how do I combine the weights to make a prediction?

- Decisions 2 and 3 often go hand in hand.
  - The “model” is typically defined by how 2 and 3 are defined
  - Finding “features” is a separate task.
Feature-Based Model

- Naïve Bayes Model
  - Decision 1: features are n-grams (or other features too!)
  - Decision 2: weight features using MLE: $P(n\text{-gram} \mid \text{class})$
  - Decision 3: multiply weights together

- Vector-Based Distributional Model
  - Decision 1: features are words, syntax, etc.
  - Decision 2: weight features with PMI scores
  - Decision 3: put features in a vector, and use cosine similarity
MaxEnt Model

- An important classifier in NLP…an exponential model
- This is not Naïve Bayes, but it does calculate probabilities.

\[ P(c|d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c', d)} \]

- Features are \( f_i(c, d) \)
- Feature weights are \( \lambda_i \)

- Don’t be frightened. This is easier than it looks.
Naïve Bayes is like MaxEnt

- Naïve Bayes is an exponential model too.

\[
P(c|d, \lambda) = \frac{P(c) \prod_i P(f_i|c)}{\sum_{c', P(c')} \prod_i P(f_i|c')}
\]

\[
= \frac{\exp(\log P(c) + \sum_i \log P(f_i|c))}{\sum_{c'} \log P(c') + \sum_i \log P(f_i|c')}
\]

\[
= \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c', d)}
\]

You know this definition. Just add \(\exp(\log(x))\).

The lambdas are the \(\log(P(x))\). The \(f(c,d)\) is the seen feature!
MaxEnt

- So Naïve Bayes is just features with weights.
  - The weights are probabilities.

- MaxEnt: “stop requiring weights to be probabilities”
- Learn the best weights for $P(c|d)$
  - Learn weights that optimize your $c$ guesses
  - How? Not this semester…
  - Hint: take derivatives for the lambdas, find the maximum

$$P(c|d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c', d)}$$
MaxEnt: learn the weights

- This is the probability of a class $c$

$$P(c|d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c', d)}$$

- Then we want to maximize the data

$$\log P(C|D, \lambda) = \sum_{(c,d)} \log P(c|d, \lambda)$$

$$= \sum_{(c,d)} \log \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c', d)}$$

$$= \sum_i \sum_{(c,d)} \lambda_i f_i(c, d) - \sum_{(c,d)} \log \sum_{c'} \exp \sum_i \lambda_i f_i(c', d)$$
## MaxEnt vs Naïve Bayes

<table>
<thead>
<tr>
<th>Naïve Bayes</th>
<th>MaxEnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Trained to maximize joint likelihood of data and classes: $P(c,d)$</td>
<td>• Trained to maximize conditional likelihood of classes: $P(c</td>
</tr>
<tr>
<td>• Features assumed to supply independent evidence.</td>
<td>• Feature weights take feature dependence into account.</td>
</tr>
<tr>
<td>• Feature weights can be set independently.</td>
<td>• Feature weights must be mutually estimated.</td>
</tr>
</tbody>
</table>
# MaxEnt vs Naïve Bayes

<table>
<thead>
<tr>
<th>Naïve Bayes</th>
<th>MaxEnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Trained to maximize joint likelihood of data and classes: $P(c,d)$</td>
<td>• Trained to maximize conditional likelihood of classes: $P(c</td>
</tr>
<tr>
<td>• $P(c</td>
<td>d) = P(c,d)/P(d)$</td>
</tr>
<tr>
<td>• So it learns the entire joint model $P(c,d)$ even though we only care about $P(c</td>
<td>d)$</td>
</tr>
</tbody>
</table>
What should you use?

- MaxEnt usually outperforms Naïve Bayes
- Why? MaxEnt learns better weights for the features
  - Naïve Bayes makes too many assumptions on the features, and so the model is too generalized
  - MaxEnt learns “optimal” weights, so they may be too specific to your training set and not work in the wild!

- Use MaxEnt, or at least try it to see which is best for your task. Several available implementations online:
Exercise on Features

Deep down here by the dark water lived old **Gollum**, a small, slimy creature. He was dark as darkness, except for two big, round, pale eyes in his thin face. He lived on a slimy island of rock in the middle of the lake. **Bilbo** could not see him, but **Gollum** was watching him now from the distance, with his pale eyes like telescopes.

- The word “Bilbo” is a person.
- What features would help a computer identify it as a person token?
  - **Classes**: person, location, organization, none
  - **Data**: the text, specifically the word “Bilbo”
  - **Features**: ???
Sequence Models

• This exercise brings us to sequence models.

• Sometimes classifying one word helps classify the next word (Markov chains!).

• “Bilbo Baggins said …”
  • If your classifier thought Bilbo was a name, then use that as a feature when you try to classify Baggins. This will boost the chance that you also label Baggins as a name.
  • Feature = “was the previous word a name?”
Sequence Models

• We don’t have time to cover sequence models. See your textbook.

• These are very influential and appear in several places:
  • Speech recognition
  • Named entity recognition (labeling names, as in our exercise)
  • Information extraction