Recap: Classification with logistic regression

- Choose the class that has highest probability according to

\[
P(y|\vec{x}) = \frac{1}{Z} \exp \left( \sum_i w_i f_i(\vec{x}, y) \right)
\]

- Which just means: when \( \sum_i w_i f_i(\vec{x}, y) \) goes up, \( P(y|\vec{x}) \) goes up.

- Put another way, if we want \( P(y|\vec{x}) \) to be high, then during training we should set the weights so that \( \sum_i w_i f_i(\vec{x}, y) \) is large.

Two cases to consider

- Computing \( P(y = 1|\vec{x}) \):
  - Here, \( f_1 = 1 \) and \( f_2 = 0 \).
  - We would expect the probability to be relatively high.
  - Can be achieved by having a positive value for \( w_1 \).
  - Since \( f_2 = 0 \), its weight has no effect on the final probability.

- Computing \( P(y = 2|\vec{x}) \):
  - Here, \( f_1 = 0 \) and \( f_2 = 1 \).
  - We would expect the probability to be close to zero, because sense 2 is a verb sense, and here we have a noun.
  - Can be achieved by having a large negative value for \( w_2 \).
  - By doing so, \( f_2 \) says: “If I am active, do not choose sense 2!”.
Today’s lecture

- A different way to use logistic regression: re-ranking
- How do we train the weights? Practical issues to watch out for.
- Relationship to neural network models.

MaxEnt for n-best re-ranking

- So far, we’ve used logistic regression for classification.
  - Fixed set of classes, same for all inputs.
- Word sense disambiguation:
  
<table>
<thead>
<tr>
<th>Input</th>
<th>Possible outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>word in doc1</td>
<td>sense 1, sense 2, sense 3</td>
</tr>
<tr>
<td>word in doc2</td>
<td>sense 1, sense 2, sense 3</td>
</tr>
</tbody>
</table>

- Dependency parsing:
  
<table>
<thead>
<tr>
<th>Input</th>
<th>Possible outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>parser config1</td>
<td>action 1, . . . action n</td>
</tr>
<tr>
<td>parser config2</td>
<td>action 1, . . . action n</td>
</tr>
</tbody>
</table>

MaxEnt for n-best re-ranking

- In NLP, we often use it for picking the best of \( n \) options, where the options depend on the input. For example, with \( n = 2 \):
  - Input: healthy dogs and cats
  - Possible outputs:

  ![Diagram of possible outputs for healthy dogs and cats](image)

- In NLP, we often use it for picking the best of \( n \) options, where the options depend on the input. For example, with \( n = 2 \):
  - Input: ate pizza with cheese
  - Possible outputs:

  ![Diagram of possible outputs for ate pizza with cheese](image)
MaxEnt for n-best re-ranking

• This scenario is why we defined our features as a function of both inputs and outputs: because the outputs may not be pre-defined.

• Typical usage: use a generative model to produce the options, then re-rank using a discriminative model.
  – Generative models typically faster to train and run, but can’t use arbitrary features.
  – In NLP, MaxEnt models may have so many features that extracting them from each example can be time-consuming (in both training and test).

MaxEnt for constituency parsing

• Now we have $y = \text{parse tree}, x = \text{sentence}$.

• Features can mirror parent-annotated/lexicalized PCFG:
  – counts of each CFG rule used in $y$
  – pairs of words in head-head dependency relations in $y$
  – each word in $x$ with its parent and grandparent categories in $y$.

• Note these are no longer binary features.

Global features

• Features can also capture global structure. E.g., from Charniak and Johnson (2005):
  – length difference of coordinated conjuncts

Global features

• Features can also capture global structure. E.g., from Charniak and Johnson (2005):
  – $\text{cat} = c \& \text{len} = \ell \& \text{end} \_\text{sent} \& \text{before} \_\text{punc}$
  – i.e., are heavy (long) noun phrases at the end of the sentence?

  I gave him the book about NLP
  vs
  I gave the book about NLP to him
Features for parsing

- Altogether, Charniak and Johnson (2005) use 13 feature templates
  - with a total of 1,148,697 features
  - and that is after removing features occurring less than five times

- One important feature not mentioned earlier: the log prob of the parse under the generative model!

- So, how does it do?

Parser performance

- \( F_1 \)-measure (from precision/recall on constituents) on WSJ test:
  - standard PCFG \( \sim 80\% \)
  - lexicalized PCFG (Charniak, 2000) 89.7%
  - re-ranked LPCFG (Charniak and Johnson, 2005) 91.0%

- Best WSJ parser is now 93.8%, combining NNets and ideas from parsing, language modelling (Choe and Charniak, 2016)

- But as discussed earlier, other languages/domains are still much worse.

1Figure from (Charniak, 1996): assumes POS tags as input

Training the model

Two ways to think about training:

- **What** is the goal of training (training objective)?

- **How** do we achieve that goal (training algorithm)?

Training generative models

- Easy to think in terms of how: counts/smoothing.

- But don’t forget the what:
  - Maximize the likelihood \( \Rightarrow \) take raw counts and normalize.
  - Other objectives \( \Rightarrow \) use smoothed counts.

\(^1\)Historically, smoothing methods were originally introduced purely as how: that is, without any particular justification as optimizing some objective function. However, as alluded to earlier, it was later discovered that many of these smoothing methods correspond to optimizing Bayesian objectives. So the what was discovered after the how.
Training logistic regression

Possible training objective:

• Given annotated data, choose weights that make the labels most probable under the model.

• That is, given items \(x^{(1)} \ldots x^{(N)}\) with labels \(y^{(1)} \ldots y^{(N)}\), choose

\[
\hat{w} = \text{argmax} \sum_j \log P(y^{(j)}|x^{(j)})
\]

• This is conditional maximum likelihood estimation (CMLE).

Regularization

• Like MLE for generative models, CMLE can easily overfit the training data.

  – For example, if some particular feature is only active for a single training example.

• So it’s a good idea to add a regularization term to the equation, which encourages weights closer to 0 unless there is a lot of evidence to shift them.

• We won’t discuss the details of different regularization methods, in practice it may require some experimentation (dev set!) to choose which method and how strongly to penalize large weights.

Optimizing (regularized) cond. likelihood

• Unlike generative models, we can’t simply count and normalize.

• Instead, we use gradient-based methods, which iteratively update the weights.

  – Our objective is a function whose value depends on the weights.

  – So, compute the gradient (derivative) of the function with respect to the weights.

  – Update the weights to move toward the optimum of the objective function.

Visual intuition

• Changing \(\hat{w}\) changes the value of the objective function.\(^2\)

  – Follow the gradients to optimize the objective (“hill-climbing”).

\[^2\text{Here, we are maximizing an objective such as log prob. Using an objective such as negative log prob would require minimizing; in this case the objective function is also called a loss function.}\]
But what if...?

- If there are multiple **local optima**, we won’t be guaranteed to find the **global optimum**.

Guarantees

- Luckily, (supervised) logistic regression does not have this problem.
  - With or without standard regularization, the objective has a single global optimum.
  - Good: results are easier to reproduce, don’t depend on initialization.

- But it is worth worrying about in general!
  - Unsupervised learning often has this problem (e.g., for HMMs, PCFGs, and logistic regression); so do neural networks.
  - Bad: results may depend on initialization, can vary from run to run.

Logistic regression: summary

- Model $P(y|x)$ only, have no generative process
- Can use arbitrary local/global features, including correlated ones
- Can use for classification, or choosing from n-best list.
- Training involves iteratively updating the weights, so typically slower than for generative models (especially if very many features, or if time-consuming to extract).
- Training objective has a single global optimum.

Similar ideas can be used for more complex models, e.g., sequence models for taggers that use spelling features.

Extension to neural network

- Logistic regression can be viewed as a building block of neural networks (a perceptron).
- Pictorially:
Extension to neural network

- Adding a **fully-connected layer** creates one of the simplest types of neural network: a **multi-layer perceptron (MLP)**.

\[
\frac{1}{Z} \exp\left( \sum \text{inputs} \right)
\]

\[f(\sum \text{inputs})\]

Key features of MLP

- Contains one or more **hidden layers**
  - Each node applies a non-linear function to the sum of its inputs
  - Hidden layers can be viewed as learned representations (**embeddings**) of the input

- A non-linear classifier: more powerful than logistic regression.

- Also trained using gradient-based methods, but vulnerable to local optima, so can be more difficult to train.

- Like other NNet architectures, really just a complex function computed by multiplying weights by inputs and passing through non-linearities. Not magic.
References


