Homework

- Read section 16.2 (Check errata on web page!)
- Do exercise 16.7, 16.8, and 16.9

Naive Bayes

- Bayes Decision Rule minimizes expected error:
  \[
  \hat{c} = \arg\max_{c \in C} p(c|x) \quad = \quad \arg\max_{c \in C} p(x, c)
  \]

  - We can split \( p(x, c) \) into two parts: the class prior \( p(c) \), and \( p(x|c) \), where:
    \[
    p(x|c) = \prod_i p(x_i|c)
    \]

  - Or, we can try other ways to get from \( p(x_i|c) \) to \( p(x|c) \)

Maximum Entropy

- We often need to build probability models without having access to all the required information
- In general, our probability estimates should reflect what we know and what we don’t know: ignorance is preferable to error
- Shannon’s entropy is a measure of ignorance
- Jaynes (1957): “The least informative probability distribution maximizes the entropy \( S \) subject to known constraints.”
- Wallis derivation

Maximum entropy

- A bit of terminology: let’s say \( p(x, w) \) is the ‘real’ probability of event \( x \) in context \( w \), and our predicted probability is \( q(x, w) \)
- We suppose we can get reasonable estimates of \( E_p[f_i] \) for each feature \( f_i \) from our training data
- These are our constraints:
  \[
  \sum_{x, w} p(x, w) f_i(x, w) = \sum_{x, w} q(x, w) f_i(x, w)
  \]
We used the method of Lagrange multipliers to derive a general solution for the distribution which satisfies these constraints (what we know) while maximizing the entropy (what we don’t know).

The parametric form of the distribution is:

$$q(x; \lambda) = \frac{\exp \sum_i \lambda_i f_i(x)}{\sum_x \exp \sum_i \lambda_i f_i(x)}$$

But, evaluating the partition function requires summing over all possible configurations, which is often impractical or impossible.

One way to avoid this problem is to limit ourselves to just those configurations which actually occur in the training data.

We use these constraints instead:

$$E_p[f_i] = E_q[f_i]$$

$$\sum_{x, w} p(x, w) f_i(x, w) = \sum_{x, w} p(w) q(x | w) f_i(x, w)$$

This gives us the conditional maximum entropy model:

$$q(x | w; \lambda) = \frac{\exp \sum_i \lambda_i f_i(x, w)}{\sum_x \exp \sum_i \lambda_i f_i(x, w)}$$

Given this general form for the distribution, we still need to find $\lambda$ for any given set of training data.

The form of the distribution maximizes the entropy.

What’s left to do is satisfy the constraints: we need to select values for $\lambda$ which accurately predict our feature expectations.

That means, we want to minimize the KL divergence:

$$D(p || q) = \sum_{x, w} p(x, w) \log \frac{p(x, w)}{q(x | w; \lambda)}$$

$$= \sum_{x, w} p(x, w) \log \frac{p(w) p(x | w)}{p(w) q(x | w; \lambda)}$$

$$= \sum_{x, w} p(x, w) \log \frac{p(x | w)}{q(x | w; \lambda)}$$

$$= \sum_{x, w} p(x, w) (\log p(x | w) - \log q(x | w; \lambda))$$

$$= \sum_{x, w} p(x, w) \log p(x | w) - \sum_{x, w} p(x, w) \log q(x | w; \lambda)$$
Parameter estimation

- Or, in other words, we want to maximize the log-likelihood:

\[
L(\lambda) = \sum_{x,w} p(x, w) \log q(x|w; \lambda) \\
= \sum_{x,w} p(x, w) \log \exp \sum_{z} \exp \sum_{i} \lambda_i f_i(z, w) \\
= \sum_{x,w} p(x, w) \sum_{i} \lambda_i f_i(x, w) - \sum_{x,w} p(x, w) \log \sum_{z} \exp \sum_{i} \lambda_i f_i(z, w)
\]

- So, we need to find the gradient of the log likelihood

\[
G(\lambda) = \nabla L(\lambda) \text{ and find a stationary point.}
\]

Some reminders:

\[
\begin{align*}
\frac{d}{dx}[f(x) g(x)] &= f(x) g'(x) + g(x) f'(x) \\
\frac{d}{dx}[\log f(x)] &= \frac{1}{f(x)} f'(x) \\
\frac{d}{dx}[\exp f(x)] &= f'(x) \exp f(x)
\end{align*}
\]

Parameter estimation

- The log-likelihood function \( L \) is convex

- That means that its value is maximized at \( \lambda^* \) where \( G(\lambda^*) = 0 \).

- The partial derivative of \( L(\lambda) \) for any \( \lambda_i \) depends on all the other \( \lambda \)'s, so there is no closed form solution

- Instead we proceed iteratively.

Parameter estimation

- So, for the gradient we get:

\[
\frac{\partial L(\lambda)}{\partial \lambda_i} = \sum_{x,w} p(x, w) f_i(x, w) - \\
\sum_{x,w} p(x, w) \sum_{z} \exp \sum_{k} \lambda_k f_k(z, w) \frac{\lambda_i f_i(z, w)}{\sum_{y} \exp \sum_{k} \lambda_k f_k(y, w)} f_i(z, w)
\]

\[
= \sum_{x,w} p(x, w) f_i(x, w) - \sum_{w} \left( \sum_{x} p(x, w) \right) \sum_{z} q(z|w; \lambda) f_i(z, w)
\]

\[
= \sum_{x,w} p(x, w) f_i(x, w) - \sum_{w,z} p(w) q(z|w; \lambda) f_i(z, w)
\]

\[
= E_p[f_i] - E_q[f_i]
\]

which should be reassuring

Parameter estimation
Parameter estimation

\textbf{ESTIMATE}(p)
1 \quad \lambda^{0} \leftarrow 0
2 \quad k \leftarrow 0
3 \quad \textbf{repeat}
4 \quad \text{compute } q^{(k)} \text{ from } \lambda^{(k)}
5 \quad \text{compute update } \delta^{(k)}
6 \quad \lambda^{(k+1)} \leftarrow \lambda^{(k)} + \delta^{(k)}
7 \quad k \leftarrow k + 1
8 \quad \textbf{until} \text{ converged}
9 \quad \textbf{return} \lambda^{(k)}

Iterative scaling

- Generalized Iterative Scaling (Darroch and Ratcliff 1972):
  \[
  \delta^{(k)} = \log \left( \frac{E_p[f]}{E_q[f]} \right)^{\frac{1}{p}}
  \]
- Descended from Iterative Proportional Fitting (Deming and Stephan 1940)
- Learning rate $C$ is the maximum sum of the values of all the features:
  \[
  C = \max_{x,w} \sum_i f_i(x, w)
  \]
- Easy to compute, doesn’t require evaluating gradient, or even probabilities

Iterative scaling

- Improved Iterative Scaling (Della Pietra, Della Pietra, Lafferty 1997) relaxes requirement for constant $C$
- Perform iterative scaling in each dimension in parallel, to find $\delta_i^{(k)}$ such that:
  \[
  E_p[f_i] = \sum_{x,w} p(x, w) q^{(k)}(x|w) f_i(x, w) \exp(C(x, w) \delta_i^{(k)})
  \]
- This one-dimensional optimization problem can itself be solved iteratively
- Improved Iterative Scaling also only requires computation of expectations.
- But, for this problem, iterative scaling updates are as expensive to compute as the gradient.

First order methods

- The simplest first-order method follows the gradient to find the direction of steepest ascent, with the step size $\alpha^{(k)}$ selected by line search:
  \[
  \delta^{(k)} = \alpha^{(k)} G(\lambda^{(k)})
  \]
- Steepest ascent is locally optimal, in a narrow sense
- Steepest ascent considers the same search directions repeatedly, leading to slow convergence.
First order methods

Conjugate gradient methods such as the Fletcher-Reeves or Polak-Ribière algorithms avoid this.

Search direction $p$ is a function of the previous search direction and the steepest ascent direction:

$$
\beta^{(k)} = \frac{G(\lambda^{(k)})^T G(\lambda^{(k)})}{G(\lambda^{(k-1)})^T G(\lambda^{(k-1)})} \\
p^{(k)} = G(\lambda^{(k)}) + \beta^{(k)} p^{(k-1)}
$$

As with steepest ascent, optimal step size is found by a line search:

$$
\delta^{(k)} = \alpha^{(k)} p^{(k)}
$$

Second order methods

We can improve on first-order methods by taking the second derivative into account.

If we locally model our log likelihood as a quadratic function, then the Taylor series approximation gives us:

$$
L(\lambda + \delta) \approx L(\lambda) + \delta^T G(\lambda) + \frac{1}{2} \delta^T H(\lambda) \delta
$$

We want to find the $\delta$ which maximizes this, so:

$$
0 + G(\lambda) + \delta^T H(\lambda) = 0 \\
\delta^T H(\lambda) = -G(\lambda) \\
\delta^T = \frac{G(\lambda)}{H(\lambda)}
$$

This yields Newton’s method:

$$
\delta^{(k)} = -H^{-1}(\lambda^{(k)}) G(\lambda^{(k)})
$$

This update rule provides both a direction and a step size, so a line search is generally unnecessary.

Under certain conditions, $\delta^{(k)}$ will not be an ascent direction, so to guarantee convergence a line search is sometimes required.

Newton’s method converges quickly (in one step, for a quadratic objective function).
Our log likelihood is twice differentiable, with the Hessian matrix:

\[ H_{ij}(\lambda) = E_{q_{\lambda}}[f_i f_j] - E_{q_{\lambda}}[f_i] E_{q_{\lambda}}[f_j] \]

(This is the variance-covariance matrix for \( f \).)

A variant of this (Fisher scoring) is used to fit log-linear models for statistical analysis.

For models with lots of parameters, \( H \) is too expensive to compute and invert on each iteration.

As we get close to a solution, we will be computing the gradient \( G \) at lots of closely spaced points.

We can use these gradients to estimate \( H \) (analogous to finite differencing).

Quasi-Newton methods replace inverse Hessian with:

\[ \delta^{(k)} = B^{(k)} G(\lambda^{(k)}) \]

where \( B^{(k)} \) is a symmetric, positive definite matrix which satisfies the equation:

\[ B^{(k)} y^{(k)} = \delta^{(k-1)} \]

with

\[ y^{(k)} = G(\lambda^{(k)}) - G(\lambda^{(k-1)}) \]

**Quasi-Newton methods** update an approximation of \( H^{-1} \) on each iteration, saving the cost of recomputing it.

But, we still need to store \( B \): for 100,000 features, this would require more than 74gb!

**Limited memory variable metric** methods store \( B^{(k)} \) in a compact form, using the previous \( m \) values of \( y^{(k)} \) and \( \delta^{(k)} \).

In practice, \( m \leq 5 \) works well, converging almost as fast as Newton’s method with much more modest computational requirements.
Second order methods

Evaluation

- Reduces parameter estimation to well-known problems (non-linear optimization, sparse matrix-vector products).
- PETSc and TAO (part of DoE’s ACTS Toolkit) provide the basis for efficient, highly scalable parameter estimation software, optimized for workstations, clusters, and parallel supercomputers.
- Data sets used for evaluation:

<table>
<thead>
<tr>
<th>dataset</th>
<th>classes</th>
<th>contexts</th>
<th>features</th>
<th>non-zeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>rules</td>
<td>29,602</td>
<td>2,525</td>
<td>246</td>
<td>732,384</td>
</tr>
<tr>
<td>lex</td>
<td>42,509</td>
<td>2,547</td>
<td>135,182</td>
<td>3,930,406</td>
</tr>
<tr>
<td>summary</td>
<td>24,044</td>
<td>12,022</td>
<td>198,467</td>
<td>396,626</td>
</tr>
<tr>
<td>shallow</td>
<td>8,625,782</td>
<td>375,034</td>
<td>264,142</td>
<td>55,192,723</td>
</tr>
</tbody>
</table>

Evaluation

- Time
- Ascent cg(fr) cg(prp) gis iis lmvm
- Evals
- Shallow summary
Results

- Advantages of IIS over GIS are slim.
- CG and LMVM show similar convergence properties, but LMVM tends to take less time per iteration.
- Both methods converge substantially faster than iterative scaling.
- Some algorithms are more robust than others to problems with the training data.
- High-quality numerical libraries offer many advantages for NLP
- Software available: estimate and classify

Smoothing

- As described, this is will find the maximum likelihood estimate, and runs into all the usual problems
- In fact, it’s worse, since MaxEnt models can’t represent probabilities of 0 or 1 with finite feature values
- Smoothing is just as important with MaxEnt models as any other probabilistic models
- All the usual smoothing methods can be applied in computing the empirical expectation $E_p[f_i]$
Another option is to use MAP estimation:

$$\lambda^* = \arg\max_{\lambda} q(x;\lambda) p(\lambda)$$

The parameter prior $p(\lambda)$ is the probability of a particular parameter vector independent from the training data.

MLE implicitly assumes a uniform prior over parameters.

A Gaussian prior with $\mu = 0$ will tend to prefer uniform models.

If $L(\lambda)$ is the log likelihood we use for ML estimation, we can construct a penalized likelihood:

$$L'(\lambda) = L(\lambda) + \sum_i \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{\lambda_i}{2\sigma^2} \right)$$

$$= L(\lambda) - \sum_i \frac{\lambda_i^2}{2\sigma^2} + C$$

And the gradient $G'$ is:

$$G'(\lambda) = G(\lambda) - \sum_i \frac{\lambda_i}{\sigma^2}$$