Road Map/Cheat Sheet

Martin Emms

December 3, 2018

Probability Basics

Joint Probability \( P(X, Y) \)
Marginal Probability \( P(X) = \sum_Y P(X, Y) \)
Conditional Probability \( P(Y|X) = \frac{P(X, Y)}{P(X)} \) \ldots really \( \frac{\text{count}(X, Y)}{\text{count}(X)} \)
Product Rule \( P(X, Y) = P(Y|X) \times P(X) \)
Chain Rule \( P(X, Y, Z) = p(Z|(X, Y)) \times P(X, Y) = p(Z|(X, Y)) \times P(Y|X) \times p(X) \)
Conditional Independence \( P(X|Y, Z) = P(X|Z) \) ie. \( X \) ignores \( Y \) given \( Z \)
Bayesian Inversion \( P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \)
Inference to infer \( X \) from \( Y \) choose \( X = \arg \max_X P(Y|X)P(X) \)

Maximum Likelihood Estimation general idea is that you have a sequence of data items \( d^{1:D} \), and probability model which specifies how likely that data is, \( P(d^{1:D}; \Theta) \), where \( \Theta \) are the parameters (eg. if \( d^{1:D} \) are coin-tosses of a particular coin then \( \Theta \) would contain the head prob. and the tail prob).

Different choices of \( \Theta \) make the data more and less likely under those settings. Abstractly if you ’plot’ \( P(d^{1:D}; \Theta) \) vs \( \Theta \), the plot must have maximum value at some setting of \( \Theta \). This is the so-called maximum likelihood estimate of \( \Theta \), based on the data \( d^{1:D} \)

Hidden Variable if \( z \) is hidden part of each data item, and \( x \) is visible part, the prob of all the data \( P(d^{1:D}; \Theta) \) is a product of sums:
\[
\prod_d P(x^d; \theta) = \prod_d \left( \sum_{k \in A(z)} P(z = k, x^d; \theta) \right)
\]

and generally there is no simple formula which gives Max. Likelihood Estimate of \( \theta \)

**EM algorithm**

(re)-estimation procedure, which repeatedly turns estimate \( \theta^n \) into estiate \( \theta^{n+1} \), by doing an E-step, followed by a M step:

**[E step]**

generate a virtual complete data corpus by treating each incomplete data item \( (x^d) \) as standing for all possible completions with values for \( z, (z = k, x^d) \), weighting each by its conditional probability \( P(z = k|x^d; \theta^n) \), under current parameters \( \theta^n \): often this quantity is called the responsibility. Use \( \gamma_d(k) \) for \( P(z = k|x^d) \).

**[M step]**

treating the 'responsibilities' \( \gamma_d(k) \) as if they were counts, apply maximum likelihood estimation (ie. take relevant ratios) to the virtual corpus to derive new estimates \( \theta^{n+1} \).

The E step gives weighted guesses, \( \gamma_d(k) \), for each way of completing each data point. These \( \gamma_d(k) \) are then treated as counts of virtual completed data, so each data point \( x^d \) is split into virtual population:

\[
\begin{align*}
    x^d & \quad \begin{cases}
    \text{virtual data} & \gamma_d(1) \\
    \vdots & \vdots \\
    (z = k, x^d) & \gamma_d(k)
    \end{cases}
\end{align*}
\]

**EM properties**

- the data gets likelier over the iterations : \( P(d; \theta^n) \leq P(d; \theta^{n+1}) \)
- because of this, the procedure converges to a final setting \( \theta^{final} \)
- \( \theta^{final} \) must give a local maximum of the data likelihood. There could be several maxima and which one you get will depend on where you start
- practically speaking if the range of values for the hidden variable is exponentially large then straightforward application of EM may be infeasible

**Noisy channel formulation of translation** finding a translation from \( o \) to \( s \) defined by

\[
\arg \max_s P(s|o) = \arg \max_s P(o|s) \times P(s)
\]  \hspace{1cm} (1)

NB: though translating from \( o \) to \( s \) it uses probs of \( o \) given \( s \)
IBM model 1 alignments

pair \((o, a, s)\) from sentence-aligned corpus seen as a partial version of the fully observed case: \((o, a, s)\) where \(a\) is an alignment between \(o\) and \(s\). The (IBM) alignment \(a\) is a function, from posn \(j\) in \(o\) to posn. \(i\) in \(s\) so \(a: j \rightarrow i\)

<table>
<thead>
<tr>
<th>o</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

• here \(o\) is English, and \(s\) is German
• the alignment goes up the page, English-to-German,
• alignments used though in a model of \(P(o|s)\), so down the page, German-to-English

IBM Model 1 Formulae

want joint prob of \(o\) and \(s\) and alignment \(a\), \(P(o, a, \ell_o, s)\), where \(\ell_o\) is length of \(o\), and \(a\) an alignment from \(o\) to \(s\). By chain rule have \(P(o, a, \ell_o, s) = P(o, a, \ell_o|s) \times P(s)\)

formula for \(P(o, a, \ell_o|s)\) is

\[
P(o, a, \ell_o|s) = \kappa_{\ell_o, \ell_s} \times \prod_j [p(o_j|s_{a(j)})] \tag{2}\]

The \(p(o_j|s_{a(j)})\) are word translation probabilities. \(\kappa_{\ell_o, \ell_s}\) is a constant not depending on the particular alignment, whose numerical value is not important for training\(^1\)

For example, with same example, supposing translation probs:

| t(\text{the}|\text{das}) | t(\text{house}|\text{Haus}) | t(\text{is}|\text{ist}) | t(\text{very}|\text{klitzeklein}) | t(\text{small}|\text{klitzeklein}) |
|-------------------------|-------------------------|-------------------------|---------------------------------|-------------------------|
| 0.7                     | 0.8                     | 0.8                     | 0.2                             | 0.4                     |

then

\[
p(o, a, \ell_o|s) = \kappa_{\ell_o, \ell_s} \times t(\text{the}|\text{das}) \times t(\text{house}|\text{Haus}) \times t(\text{is}|\text{ist}) \times t(\text{very}|\text{klitzeklein}) \times t(\text{small}|\text{klitzeklein})
\[
= \kappa_{\ell_o, \ell_s} \times 0.7 \times 0.8 \times 0.8 \times 0.2 \times 0.4
\]

IBM Model Training The E-step of brute force EM treats a whole alignment \(a\) as a hidden variable, calculates an alignment 'responsibility' \(\gamma_d(a) = p(a|\ell_o, s^d)\) and uses it to update apparent counts of word-pairings – see left below. The E-step of efficient EM treats the alignment destination of each position \(j\) as a hidden variable in its own right, calculates an alignment step 'responsibility' \(\gamma_d((j, i)) = p((j, i) \in a|\ell_o, s^d)\) for each different destination for \(j\) and uses these to update apparent counts of word-pairings – see right below:

\(^1\)it is \(p(\ell_o|\ell_s) / (\ell_s + 1)\)\(^{\alpha}a\)
for each pair \((o,s)\)

for each \(a\)
calculate \(p(a|o,s)\)

for each \(j \in 1 : \ell_o\)

\[
\#(o_j,s_{a(j)}) += p(a|o,s)
\]

The mini-responsibility \(p((j,i) \in a|o,s)\) turns out to have the simple formula:

\[
\gamma_d(j,i) = \frac{tr(o_j|s_i)}{\sum_r[tr(o_j|s_r)]}
\]

See the course-pages for worked examples of both of these on a sufficiently tiny examples to permit working out by hand.

You should understand what a sum-of-products to product-of-sums conversion is, for example allowing the conversion

\[
\sum_{a(1)=0}^I \cdots \sum_{a(J)=0}^I \prod_{j=1}^J t(o_j|s_{a(j)}) = \prod_{j=1}^J [\sum_{a(j)=0}^I t(o_j|s_{a(j)})]
\]

Phrase-based translation model

- pair \((o,s)\) from sentence aligned corpus seen as a partial version of \((\bar{o}_{1:K}, \tau, \bar{s}_{1:K})\) where phrase sequences \(\bar{o}_{1:K}\) and \(\bar{s}_{1:K}\) are segmentations of \(s\) and \(o\) into \(K\) phrases and \(\tau\) is a mapping from the phrases \(\bar{s}\) to the phrases \(\bar{o}\), which is 1-to-1, and generally not order preserving.

\[
\begin{array}{c}
\text{source} \\
\text{observed}
\end{array}
\]

\[
\begin{array}{c}
1 & 2 & 3 & 4 \\
\text{he} & \text{does not} & \text{go} & \text{home} \\
\text{er} & \text{geht} & \text{ja nicht} & \text{nach hause} \\
\tau(1) & \tau(3) & \tau(2) & \tau(4)
\end{array}
\]

the formula for \(p(\bar{o}, \tau, \bar{s})\) is:

\[
p(\bar{o}, \tau, \bar{s}) = p(\bar{o}, \tau|\bar{s}) \times p(\bar{s})
\]

\[
= \prod_{k=1}^K tr(\bar{o}_{\tau(k)}|\bar{s}_k) d(\bar{o}_{\tau(k-1)}, \bar{o}_{\tau(k)}) LM(s)
\]

- **Phrase translation**: term \(tr(\bar{o}_{\tau(k)}|\bar{s}_k)\) for the phrase-translation probabilities for an observed phrase \(\bar{o}_{\tau(k)}\) given a source phrase \(\bar{s}_k\)

- **Reordering**: how likely is the destination for a \(\bar{s}_k\) phrase given destination for previous \(\bar{s}_{k-1}\) phrase: \(d(\bar{o}_{\tau(k-1)}, \bar{o}_{\tau(k)})\), just standardly defined as an exponentially decaying function of the ‘distance’ \(x\) between end \(\bar{o}_{\tau(k-1)}\) and start of \(\bar{o}_{\tau(k)}\)

- **Language model**: probability of source phrases \(\bar{s}\) set equal to probability of the source sequence \(s\) as given by an \(n\)-gram model \(LM(s)\) (ie, \(\prod_i[P(s_t|s_{t-N} \ldots s_{t-1})]\))

Training for PB-SMT built on top of IBM models
1. for each training pair \((o, s)\) find best IBM alignments in both directions
2. merge these alignments
3. extract consistent phrase-pairs; small phrase-pairs combined if relevant parts are adjacent

4. from counts of phrase pairs from all training data, phrase-translation prob. simply defined by relative frequencies:

\[
tr(e|g) = \frac{\text{count}(e, g)}{\sum_e \text{count}(e', g)} \quad \text{and} \quad tr(g|e) = \frac{\text{count}(e, g)}{\sum_g \text{count}(e, g')}
\]

Decoding for PB-SMT phrases of \(\vec{s}_{1..K}\) picked in order, phrases of \(\vec{o}_{1..K}\) out of order

- ‘naive’ enumeration of hypotheses groups by number of phrase pairs; is exponentially expensive
- preferred enumeration of hypothesis groups by \(o\)-coverage: \(\text{bin}_L\) contains hypotheses that cover \(L\) words of \(o\); is also exponentially expensive but usually done in conjunction with pruning of each bin to hold no more than \(h_{max}\) best hypotheses – so-called ‘beam’ search – and then feasible.

For \(er\ \text{geht nach}\ haus\) assuming mini phrase table to left below, and impossible bigrams to right:

```plaintext
   | er  | geht | nach | haus |
---|-----|------|------|------|
he | he  | goes | he   | home |
```

place empty hypothesis into \(\text{bin}_0\)
for \(L = 0\) to \(\text{length}(o) - 1\) {
  for each hypothesis in \(\text{bin}_L\) {
    for each applicable pair \(s, \vec{o}\) {
      create new hypothesis (including score)
      place in \(\text{bin}_{L+\vec{x}}\) where \(\vec{o}\) has length \(x\)
      prune \(\text{bin}_{L+\vec{x}}\) if too big
    }
  }
}

bits of len of obs words used
Hidden Markov Models

An HMM fundamentally assigns a joint probability to any finite state+observation sequence \( P(o_1:T, s_1:T) \)

\[
P(o_1:T, s_1:T) = P(s_1)P(o_1|s_1) \times \prod_{t=2}^{T} P(s_t|s_{t-1})P(o_t|s_t)
\]

The key idea is that \( s_t \) depends only on \( s_{t-1} \) and \( o_t \) depends only on \( s_t \).

All the probs needed standardly specified by a triple \((\pi, A, B)\) concerning starts, transitions and observations:

\( \pi \): assigns to state \( i \) the prob \( P(s_1 = i) \), abbreviated \( \pi_i \) or \( \pi[i] \)

\( A \): assigns to states \( i, j \) the prob \( P(s_t = j|s_{t-1} = i) \), abbreviated \( a_{ij} \) or \( A[i,j] \)

\( B \): assigns to state+obs \( i, k \) the prob \( P(o_t = k|s_t = i) \), abbreviated \( b_{i}(k) \) or \( B[i,k] \)

With \( \pi, A, \) and \( B \), the joint prob could be expressed

\[
P(o_1:T, s_1:T) \quad = \quad \pi_{s_1}b_{s_1}(o_1) \times \prod_{t=2}^{T} a_{s_{t-1} s_t} b_{s_t}(o_t)
\]

or

\[
= \quad \pi[s_1]B[s_1, o_1] \times \prod_{t=2}^{T} A[s_{t-1}, s_t]B[s_t, o_t]
\]

HMM Forward Algorithm

The forward algorithm computes a time and state dependent quantity \( \alpha_t(i) \), whose meaning is:

\[
\alpha_t(i) = \text{the joint probability of being in state } i \text{ at time } t \text{ and emitting the observation symbols } o_1:t
\]

\[
= \sum_{s_1:t, s_t=i} P(s_1:t, o_1:t)
\]

\[
= P(o_1 \ldots o_t, s_1 = i)
\]

crucially \( \alpha_t(.) \) can be easily specified from \( \alpha_{t-1}(.) \). Its recursive definition is:

base \( \alpha_1(i) = \pi(i)b_i(o_1) \)

recursive \( \alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i)a_{ij}b_j(o_t) \), for \( t = 2, \ldots, T \)

6
which leads to the pseudo code algorithm

Initialisation:
```
for (i = 1; i \leq N; i++) {
    alpha(1, i) = \pi(i)b_i(o_1);
}
```

Iteration:
```
for (t = 2; t \leq T; t++) {
    for (j = 1; j \leq N; j++) {
        total = 0;
        for (i = 1; i \leq N; i++) {
            total += alpha(t - 1, i) \times a_{ij} \times b_j(o_t);
        }
        alpha(t, j) = total;
    }
}
```

The cost of this algorithm will be of the order of \( N^2T \), compared to the brute force cost \( NT \).

One use of \( \alpha_t(i) \) is to derive the total observation probability – \( P(o_1:T) \) – because

\[
P(o_1:T) = \sum_i (P(o_1 \ldots o_T, s_T = i)) = \sum_i \alpha_T(i)
\]

**EM training of HMMs**

Training is done based on a corpus of observation sequences \( o^1 \ldots o^D \). Akin to IBM model training, a brute-force EM approach would calculate a sequence level ‘responsibility’ \( \gamma_d(s) = p(s|o^d) \), but is exponentially expensive. The efficient Baum-Welch version uses ‘clock-tick’ level responsibilities, concerning states at a single time point \( t \), such as \( \gamma_t^d(i) = P(s_t = i|o^d) \) ie. the cond prob of state \( i \) at \( t \) given \( o^d \).

From these an expected count for the whole sequence (eg. \( E^d(i) \) the expected count of being in state \( i \) is got by summing over all \( t \) (eg. \( E^d(i) = \sum_{t=1}^T \gamma_t^d(i) \))

Unlike the IBM model case, there is no trivial formula\(^2\) giving \( P(s_t = i|o^d) \) directly from \( A, B \) and \( \pi \). Instead it is calculated from \( \alpha_t(i) \) and \( \beta_t(i) \) quantities (tables) which themselves can be efficiently calculated.

\[
\alpha_t(i) = \text{the joint probability of being in state } i \text{ at time } t \text{ and emitting the observation symbols } o_{1:t}
\]
\[
= P(o_1 \ldots o_t, s_t = i)
\]

\[
\beta_t(i) = \text{the conditional probability of emitting observation symbols } o_{t+1:T},
\]
\[
\text{given being in state } i \text{ at time } t
\]
\[
= P(o_{t+1} \ldots o_T|s_t = i)
\]

\(^2\)for IBM models there was a trivial formula \( P((j, i) \in a(o^d, s^d)) \)
\(\alpha_t(i)\) was already mentioned in connection with total observation probability. Here all its values at every \(t\) are calculated and kept. The \(\beta_t(i)\) values have no intrinsic use but have the crucial property when multiplied together with \(\alpha_t(i)\) that

\[
\alpha_t(i)\beta_t(i) = \text{the joint probability of emitting observations symbols } \mathbf{o}_{1:T} \text{ and being in state } i \text{ at time } t = P(\mathbf{o}_1 \ldots \mathbf{o}_t, s_t = i, \mathbf{o}_{t+1} \ldots \mathbf{o}_T)
\]

and this crucially allows \(\gamma^d_t(i)\) to be worked out as

\[
\gamma^d_t(i) = \frac{p(s_t = i | \mathbf{o}_{1:T}) \alpha_t(i) \beta_t(i)}{\sum_{i'} \alpha_T(i')}
\]

**Isolated Word Recognition** To make a system to recognise spoken versions of the digits 1, 2 and 3:

- The digits 1, 2 and 3 each have their own HMM \(M_1, M_2\) and \(M_3\)
- **Training (Baum-Welch)** using a set of obs. sequences \(\mathcal{O}_1\) for digit 1 (ie. recordings of '1'), the Baum-Welch resets the parameters of \(M_1\), to max. the prob. of these sequences according to model \(M_1\). Same done for \(M_2\) and \(M_3\) with their obs. sequences.
- **Recognition (Forward)** an unknown \(\mathbf{o}^{\text{test}}\) is recognised by using the Forward Algorithm, to find out which \(M_i\) gives the highest prob to \(\mathbf{o}^{\text{test}}\), ie. the \(P(\mathbf{o}^{\text{test}}; M_i)\) values are compared.

**Topic Models** A document is seen as a sequence of words \(\mathbf{w}_{1:T}\) paired with a sequence of topics \(\mathbf{z}_{1:T}\)

The topics in \(\mathbf{z}_{1:T}\) chosen independently of each other.

The word at position \(t\) chosen from a word distribution specific to the topic at \(t\)

\[
p(\mathbf{z}_{1:T}, \mathbf{w}_{1:T}) = \prod_t p(z^d_t)p(w^d_t | z^d_t) \quad \text{with parameter } \Theta^d[i] \text{ for } p(z^d_t = i) \quad \text{parameter } B[i, k] \text{ for } p(w^d_t = k | z^d_t = i)
\]

\(B[i, k]\) plays exactly the same role as it did for HMMs. \(\Theta^d[i]\) replaces the \(A[i, j]\) and \(\pi[i]\) parameters of HMMs
Topic models invariably are applied to a collection of documents $w^1 \ldots w^D$. The word-given-topic parameter $B[i, k]$ is shared by all documents (like HMMs). The topic probabilities themselves are specific to each $d$ – there is a particular $\Theta^d[i]$ parameter for each document (not like HMMs).

<table>
<thead>
<tr>
<th>topic probs</th>
<th>documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Theta^1_{1:M}$</td>
<td>$z^1_1 \ldots z^1_T$ $w^1_1 \ldots w^1_T$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$\Theta^d_{1:M}$</td>
<td>$z^d_1 \ldots z^d_T$ $w^d_1 \ldots w^d_T$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$\Theta^{D}_{1:M}$</td>
<td>$z^D_1 \ldots z^D_T$ $w^D_1 \ldots w^D_T$</td>
</tr>
</tbody>
</table>

$B$ $v_1 \ldots v_K$
$1$ $\vdots$
$M$ $\vdots$

**EM training of Topic Models**Params $\Theta^1 \ldots \Theta^D$ and $B$ are estimated just from a corpus of documents $w^1 \ldots w^D$

- As with HMMs, rather than working out $p(z|w^d)$ – which is exponentially costly – it makes sense to consider ‘clock-tick’ version $p(z_t = i|w^d)$
- This much easier than for HMMs as the simple independence of the topics makes it the case that $p(z_t = i|w^d) = p(z_t = i|w^d_t)$, and this has simple formula:

$$p(z_t = i|w^d_t) = \frac{p(z_t = i)p(w^d_t|z_t = i)}{\sum_{i'} p(z_t = i')p(w^d_t|z_t = i')} = \frac{\Theta^d[i]B[i, w^d_t]}{\sum_{i'} \Theta^d[i']B[i', w^d_t]}$$

**notation** The above summary of topic-models used $z$ and $w$, with $z$ being the hidden ‘topics’ and $w$ being the visible words. Replacing throughout with $s$ and $o$ makes the overlap in the maths with HMMs even more obvious.