Essential Information Theory

Kullback-Leibler Distance (Relative Entropy)

- Remember:
  - long series of experiments... c_i/T_i oscillates around some number... we can only estimate it... to get a distribution q.
- So we get a distribution q; (sample space Ω, r.v. X)

  the true distribution is, however, p. (same Ω, X)

  $\Rightarrow$ how big error are we making?
- D(p||q) (the Kullback-Leibler distance):

$$D(p||q) = \sum_{x \in \Omega} p(x) \log_2 (p(x)/q(x)) = E_p \log_2 (p(x)/q(x))$$

Comments on Relative Entropy

- Conventions:
  - $0 \log 0 = 0$
  - $p \log (p/0) = \infty$ (for $p > 0$)
- Distance? (less "misleading": Divergence)
  - not quite:
    - **not symmetric: $D(p||q) \neq D(q||p)$**
    - **does not satisfy the triangle inequality**
  - but useful to look at it that way
- $H(p) + D(p||q)$: bits needed for encoding p if q is used

Mutual Information (MI)

in terms of relative entropy

- Random variables X, Y; $p_{X \cap Y}(x,y)$, $p_X(x)$, $p_Y(y)$
- Mutual information (between two random variables X,Y):

$$I(X,Y) = D(p(x,y) \,||\, p(x)p(y))$$

- I(X,Y) measures how much (our knowledge of) Y contributes (on average) to easing the prediction of X
- or, how <u>p(x,y)</u> deviates from (independent) <u>p(x)p(y)</u>

Mutual Information: the Formula

- Rewrite the definition: [recall: $D(r||s) = \sum_{v \in \Omega} r(v) \log_2 (r(v)/s(v))$;
  substitute $r(v) = p(x,y)$, $s(v) = p(x)p(y)$; $<v> \sim <x,y>$]

$$\boxed{\begin{aligned} I(X,Y) &= D(p(x,y) \,||\, p(x)p(y)) = \\ &= \sum_{x \in \Omega} \sum_{y \in \Psi} p(x,y) \log_2 (p(x,y)/p(x)p(y)) \quad \textbf{!} \end{aligned}}$$

- Measured in bits (what else? :-)

Properties of MI vs. Entropy

- $I(X,Y) = H(X) \underline{- H(X|Y)}$ = number of bits the knowledge of Y lowers the entropy of X

  $= H(Y) - H(Y|X)$ (prev. foil, symmetry)

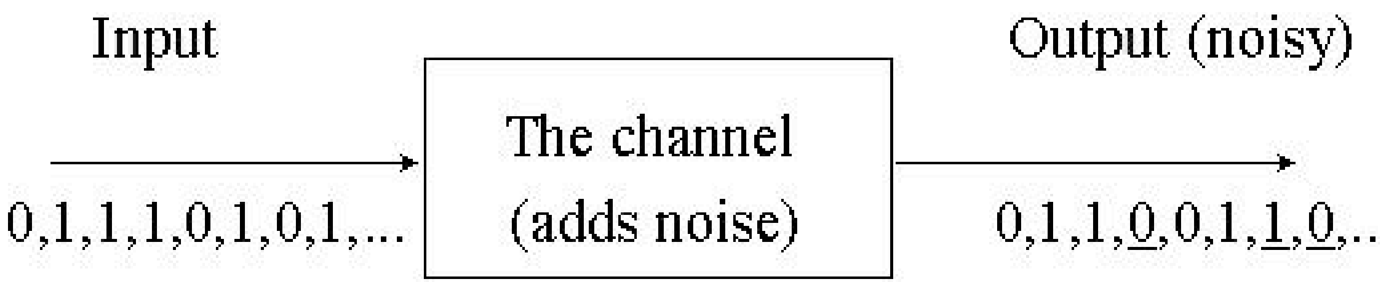
Recall: $H(X,Y) = H(X|Y) + H(Y) \Rightarrow -H(X|Y) = H(Y) - H(X,Y) \Rightarrow$

- $I(X,Y) = H(X) + \underline{H(Y) - H(X,Y)}$
- $I(X,X) = H(X)$ (since $H(X|X) = 0$)
- $I(X,Y) = I(Y,X)$ (just for completeness)
- $I(X,Y) \geq 0$ ... let's prove that now (as promised).

Language Modeling (and) the Noisy Channel

The Noisy Channel

- Prototypical case:



- Model: probability of error (noise):
- Example: $p(0|1) = .3$ $p(1|1) = .7$ $p(1|0) = .4$ $p(0|0) = .6$
- <u>The Task</u>:

known: the noisy output; want to know: the input (***<u>decoding</u>***)

Noisy Channel Applications

- OCR
  - straightforward: text → print (adds noise), scan → image
- Handwriting recognition
  - text → neurons, muscles (“noise”), scan/digitize → image
- Speech recognition (dictation, commands, etc.)
  - text → conversion to acoustic signal (“noise”) → acoustic waves
- Machine Translation
  - text in target language → translation (“noise”) → source language
- Also: Part of Speech Tagging
  - sequence of tags → selection of word forms → text

Noisy Channel: The Golden Rule of ...

OCR, ASR, HR, MT, ...

- Recall:

  $p(A|B) = p(B|A)\ p(A) / p(B)$ (Bayes formula)

  $A_{best} = \mathrm{argmax}_A\ p(B|A)\ p(A)$ (The Golden Rule)

- $p(B|A)$: the acoustic/image/translation/lexical model
  - application-specific name
  - will explore later
- $p(A)$: <u>***the language model***</u>

The Perfect Language Model

- Sequence of word forms [forget about tagging for the moment]
- Notation: $A \sim W = (w_1, w_2, w_3, \ldots, w_d)$
- The big (modeling) question:

$$p(W) = ?$$

- Well, we know (Bayes/chain rule $\rightarrow$):

$$p(W) = p(w_1, w_2, w_3, \ldots, w_d) =$$

$$= p(w_1) \times p(w_2|w_1) \times p(w_3|w_1, w_2) \times \ldots \times p(w_d|w_1, w_2, \ldots, w_{d-1})$$

- Not practical (even short $W \rightarrow$ too many parameters)

Markov Chain

- Unlimited memory (cf. previous foil):
  - for w_i, we know <u>all</u> its predecessors $w_1, w_2, w_3, \ldots, w_{i-1}$
- Limited memory:
  - we disregard "too old" predecessors
  - remember only *k* previous words: $w_{i-k}, w_{i-k+1}, \ldots, w_{i-1}$
  - called "k^{th} order Markov approximation"
- + stationary character (no change over time):

$$p(W) \cong \prod_{i=1..d} p(w_i | w_{i-k}, w_{i-k+1}, \ldots, w_{i-1}),\ d = |W|$$

n-gram Language Models

- $(n-1)^{th}$ order Markov approximation $\rightarrow$ n-gram LM:

$$\boxed{p(W) =_{df} \prod_{i=1..d} p(\overset{\text{prediction}}{w_i} | \overbrace{w_{i-n+1}, w_{i-n+2}, \ldots, w_{i-1}}^{\text{history}}) \quad !}$$

- In particular (assume vocabulary $|V| = 60k$):

0-gram LM: uniform model,	$p(w) = 1/\lvert V\rvert$,	**1 parameter**
1-gram LM: unigram model,	$p(w)$,	**6×10^4 parameters**
2-gram LM: bigram model,	$p(w_i\|w_{i-1})$	**3.6×10^9 parameters**
3-gram LM: trigram model,	$p(w_i\|w_{i-2},w_{i-1})$	**2.16×10^{14} parameters**

LM: Observations

- How large *n*?
  - nothing is enough (theoretically)
  - but anyway: as much as possible ($\rightarrow$ close to "perfect" model)
  - empirically: ***3***
    - parameter estimation? (reliability, data availability, storage space, ...)
    - 4 is too much: $|V|=60k \rightarrow 1.296\times10^{19}$ parameters
    - but: 6-7 would be (almost) ideal (having enough data): in fact, one can recover original from 7-grams!
- Reliability ~ (1 / Detail) ($\rightarrow$ need compromise)
- For now, keep word forms (no "linguistic" processing)

The Length Issue

- $\forall n; \Sigma_{w \in \Omega^n} p(w) = 1 \Rightarrow \Sigma_{n=1..\infty} \Sigma_{w \in \Omega^n} p(w) >> 1 \ (\rightarrow \infty)$
- We want to model all sequences of words
  - for "fixed" length tasks: no problem - n fixed, sum is 1
    - tagging, OCR/handwriting (if words identified ahead of time)
  - for "variable" length tasks: have to account for
    - discount shorter sentences
- General model: for each sequence of words of length n,

  define $p'(w) = \lambda_n p(w)$ such that $\Sigma_{n=1..\infty} \lambda_n = 1 \Rightarrow$

$$\Sigma_{n=1..\infty} \Sigma_{w \in \Omega^n} p'(w) = 1$$

  e.g., estimate λ_n from data; or use normal or other distribution

Parameter Estimation

- Parameter: numerical value needed to compute p(w|h)
- From data (how else?)
- Data preparation:
    - **get rid of formatting etc. ("text cleaning")**
    - **define words (separate but include punctuation, call it "word")**
    - **define sentence boundaries (insert "words" <s> and </s>)**
    - **letter case: keep, discard, or be smart:**
        - name recognition
        - number type identification

      [these are huge problems per se!]
    - **numbers: keep, replace by <num>, or be smart (form ~ pronunciation)**

Maximum Likelihood Estimate

- MLE: Relative Frequency...
  - ...best predicts the data at hand (the "training data")
- Trigrams from Training Data T:
  - count sequences of three words in T: $c_3(w_{i-2},w_{i-1},w_i)$
    - [NB: notation: just saying that the three words follow each other]
  - count sequences of two words in T: $c_2(w_{i-1},w_i)$:
    - **either use $c_2(y,z) = \Sigma_w\, c_3(y,z,w)$**
    - **or count differently at the beginning (& end) of data!**

$$\boxed{p(w_i|w_{i-2},w_{i-1}) =_{est.} c_3(w_{i-2},w_{i-1},w_i) \,/\, c_2(w_{i-2},w_{i-1})}\ !$$

Character Language Model

- Use individual characters instead of words:

$$p(W) =_{df} \prod_{i=1..d} p(c_i | c_{i-n+1}, c_{i-n+2}, \ldots, c_{i-1})$$

- Same formulas etc.
- Might consider 4-grams, 5-grams or even more
- Good only for language comparison
- Transform cross-entropy between letter- and word-based models:

$$H_S(p_c) = H_S(p_w) \,/\, \text{avg. \# of characters/word in S}$$

LM: an Example

- Training data:

  $<s> <s>$ He can buy the can of soda.

  - Unigram: $p_1(\text{He}) = p_1(\text{buy}) = p_1(\text{the}) = p_1(\text{of}) = p_1(\text{soda}) = p_1(.) = .125$

    $p_1(\text{can}) = .25$
  - Bigram: $p_2(\text{He}|<s>) = 1$, $p_2(\text{can}|\text{He}) = 1$, $p_2(\text{buy}|\text{can}) = .5$,

    $p_2(\text{of}|\text{can}) = .5$, $p_2(\text{the}|\text{buy}) = 1, \ldots$
  - Trigram: $p_3(\text{He}|<s>,<s>) = 1$, $p_3(\text{can}|<s>,\text{He}) = 1$,

    $p_3(\text{buy}|\text{He},\text{can}) = 1$, $p_3(\text{of}|\text{the},\text{can}) = 1$, ..., $p_3(.|\text{of},\text{soda}) = 1$.
  - Entropy: $H(p_1) = 2.75$, $H(p_2) = .25$, $H(p_3) = 0$ $\leftarrow$ Great?!

LM: an Example (The Problem)

- Cross-entropy:
- S = <s> <s> It was the greatest buy of all.
- Even $H_S(p_1)$ fails (= $H_S(p_2) = H_S(p_3) = \infty$), because:
  - all unigrams but p_1(the), p_1(buy), p_1(of) and p_1(.) are 0.
  - all bigram probabilities are 0.
  - all trigram probabilities are 0.
- We want: to make all (theoretically possible*) probabilities non-zero.

The Zero Problem

- "Raw" n-gram language model estimate:
  - necessarily, some zeros
    - !many: trigram model $\rightarrow 2.16\times10^{14}$ parameters, data ~ 10^9 words
  - which are true 0?
    - optimal situation: even the least frequent trigram would be seen several times, in order to distinguish it's probability vs. other trigrams
    - optimal situation cannot happen, unfortunately (open question: how many data would we need?)
  - $\rightarrow$ we don't know
  - we must eliminate the zeros
- Two kinds of zeros: $p(w|h) = 0$, or even $p(h) = 0$!

Why do we need Nonzero Probs?

- To avoid infinite Cross Entropy:
  - happens when an event is found in test data which has not been seen in training data

    $H(p) = \infty$: prevents comparing data with ≥ 0 "errors"
- To make the system more robust
  - low count estimates:
    - they typically happen for "detailed" but relatively rare appearances
  - high count estimates: reliable but less "detailed"

Eliminating the Zero Probabilities: Smoothing

- Get new p'(w) (same Ω): almost p(w) but no zeros
- Discount w for (some) p(w) > 0: new p'(w) < p(w)

  $$\Sigma_{w \in \text{discounted}} (p(w) - p'(w)) = D$$
- Distribute D to all w; p(w) = 0: new p'(w) > p(w)
  - possibly also to other w with low p(w)
- For some w (possibly): p'(w) = p(w)
- Make sure $\Sigma_{w \in \Omega} p'(w) = 1$
- There are many ways of ***smoothing***

Smoothing by Adding 1

- Simplest but not really usable:
  - Predicting words w from a vocabulary V, training data T:

    $p'(w|h) = (c(h,w) + 1) \ / \ (c(h) + |V|)$

    - for non-conditional distributions: $p'(w) = (c(w) + 1) / (|T| + |V|)$
  - Problem if $|V| > c(h)$ (as is often the case; even >> c(h)!)
- Example: **Training data:** <s> what is it what is small ? $|T| = 8$
    - V = { what, is, it, small, ?, <s>, flying, birds, are, a, bird, . }, $|V| = 12$
    - p(it)=.125, p(what)=.25, p(.)=0 p(what is it?) = $.25^2 \times .125^2 \cong$ .001

      p(it is flying.) = $.125 \times .25 \times 0^2 = 0$
    - p'(it) =.1, p'(what) =.15, p'(.)=.05 p'(what is it?) = $.15^2 \times .1^2 \cong$ .0002

      p'(it is flying.) = $.1 \times .15 \times .05^2 \cong .00004$

Adding *less than* 1

- Equally simple:
  - Predicting words w from a vocabulary V, training data T:

    $p'(w|h) = (c(h,w) + \lambda) \; / \; (c(h) + \lambda|V|), \lambda < 1$

    - for non-conditional distributions: $p'(w) = (c(w) + \lambda) / (|T| + \lambda|V|)$
- Example: Training data: <s> what is it what is small ? $|T| = 8$
  - $V = \{$ what, is, it, small, ?, <s>, flying, birds, are, a, bird, . $\}$, $|V| = 12$
  - p(it)=.125, p(what)=.25, p(.)=0 p(what is it?) = $.25^2 \times .125^2 \cong .001$

    p(it is flying.) = $.125 \times .25 \times 0^2 = 0$
  - Use $\lambda = .1$:
  - p'(it)$\cong$ .12, p'(what)$\cong$ .23, p'(.)$\cong$ .01 p'(what is it?) = $.23^2 \times .12^2 \cong .0007$

    p'(it is flying.) = $.12 \times .23 \times .01^2 \cong .000003$

Good - Turing

- Suitable for estimation from large data
  - similar idea: discount/boost the relative frequency estimate:

    $p_r(w) = (c(w) + 1) \times N(c(w) + 1) / (|T| \times N(c(w)))$,

    where N(c) is the count of words with count c (count-of-counts)

    specifically, for c(w) = 0 (unseen words), $p_r(w) = N(1) / (|T| \times N(0))$
  - good for small counts (< 5-10, where N(c) is high)
  - variants (see MS)
  - normalization! (so that we have $\Sigma_w p'(w) = 1$)

Good-Turing: An Example

- Example: remember: $p_r(w) = (c(w) + 1) \times N(c(w) + 1) / (|T| \times N(c(w)))$

  Training data: <s> what is it what is small ? $|T| = 8$

  - V = { what, is, it, small, ?, <s>, flying, birds, are, a, bird, . }, $|V| = 12$

    p(it)=.125, p(what)=.25, p(.)=0 p(what is it?) = $.25^2 \times .125^2 \cong$ .001

    p(it is flying.) = $.125 \times .25 \times 0^2 =$ 0

  - Raw reestimation (N(0) = 6, N(1) = 4, N(2) = 2, N(i) = 0 for i > 2):

    $p_r(\text{it}) = (1+1) \times N(1+1)/(8 \times N(1)) = 2 \times 2/(8 \times 4) = .125$

    $p_r(\text{what}) = (2+1) \times N(2+1)/(8 \times N(2)) = 3 \times 0/(8 \times 2) = 0$: keep orig. p(what)

    $p_r(.) = (0+1) \times N(0+1)/(8 \times N(0)) = 1 \times 4/(8 \times 6) \cong .083$

  - Normalize (divide by $1.5 = \Sigma_{w \in |V|} p_r(w)$) and compute:

    p'(it)$\cong$ .08, p'(what)$\cong$ .17, p'(.)$\cong$ .06 p'(what is it?) = $.17^2 \times .08^2 \cong$ .0002

    p'(it is flying.) = $.08 \times .17 \times .06^2 \cong$ .00004

Smoothing by Combination: Linear Interpolation

- Combine what?
  - distributions of various level of detail vs. reliability
- n-gram models:
  - use (n-1)gram, (n-2)gram, ..., uniform

$\longrightarrow$ reliability

$\longleftarrow$ detail

- Simplest possible combination:
  - sum of probabilities, normalize:
    - $p(0|0) = .8$, $p(1|0) = .2$, $p(0|1) = 1$, $p(1|1) = 0$, $p(0) = .4$, $p(1) = .6$:
    - $p'(0|0) = .6$, $p'(1|0) = .4$, $p'(0|1) = .7$, $p'(1|1) = .3$

Typical n-gram LM Smoothing

- Weight in less detailed distributions using $\lambda=(\lambda_0,\lambda_1,\lambda_2,\lambda_3)$:

  $$p'_\lambda(w_i| w_{i-2},w_{i-1}) = \lambda_3 p_3(w_i| w_{i-2},w_{i-1}) + \lambda_2 p_2(w_i| w_{i-1}) + \lambda_1 p_1(w_i) + \lambda_0 /|V|$$

- Normalize:

  $\lambda_i > 0,\ \Sigma_{i=0..n}\lambda_i = 1$ is sufficient ($\lambda_0 = 1 - \Sigma_{i=1..n}\lambda_i$) (n=3)

- Estimation using MLE:
  - fix the p_3, p_2, p_1 and $|V|$ parameters as estimated from the training data
  - then find such $\{\lambda_i\}$which minimizes the cross entropy (maximizes probability of data): $-(1/|D|)\Sigma_{i=1..|D|}\log_2(p'_\lambda(w_i|h_i))$

The Formulas

- Repeat: minimizing $-(1/|H|)\Sigma_{i=1..|H|}\log_2(p'_\lambda(w_i|h_i))$ over λ

$$\boxed{\begin{aligned} p'_\lambda(w_i|\, h_i) = p'_\lambda(w_i|\, w_{i-2}, w_{i-1}) = \lambda_3\, p_3(w_i|\, w_{i-2}, w_{i-1}) + \\ \lambda_2\, p_2(w_i|\, w_{i-1}) + \lambda_1\, p_1(w_i) + \lambda_0 / |V| \quad ! \end{aligned}}$$

- "Expected Counts (of lambdas)": $j = 0..3$

$$\boxed{c(\lambda_j) = \Sigma_{i=1..|H|} (\lambda_j p_j(w_i|h_i) / p'_\lambda(w_i|h_i)) \quad !}$$

- "Next λ": $j = 0..3$

$$\boxed{\lambda_{j,next} = c(\lambda_j) / \Sigma_{k=0..3} (c(\lambda_k)) \quad !}$$

The (Smoothing) EM Algorithm

1. Start with some λ, such that $\lambda_j > 0$ for all $j \in 0..3$.
2. Compute "Expected Counts" for each λ_j.
3. Compute new set of λ_j, using the "Next λ" formula.
4. Start over at step 2, unless a termination condition is met.

- Termination condition: convergence of λ.
  - Simply set an ε, and finish if $|\lambda_j - \lambda_{j,next}| < \varepsilon$ for each j (step 3).
- Guaranteed to converge:

  follows from Jensen's inequality, plus a technical proof.

Remark on Linear Interpolation Smoothing

- "Bucketed" smoothing:
  - use several vectors of λ instead of one, based on (the frequency of) history: $\lambda(h)$
    - e.g. for h = (micrograms,per) we will have

      $\lambda(h) = (.999,.0009,.00009,.00001)$

      (because "cubic" is the only word to follow...)
  - actually: not a separate set for each history, but rather a set for "similar" histories ("bucket"):

    $\lambda(b(h))$, where b: $V^2 \rightarrow N$ (in the case of trigrams)

    <u>b</u> classifies histories according to their reliability (~ frequency)

Some More Technical Hints

- Set V = {all words from training data}.
  - You may also consider $V = T \cup H$, but it does not make the coding in any way simpler (in fact, harder).
  - But: you must ***never*** use the test data for you vocabulary!
- Prepend two "words" in front of all data:
  - avoids beginning-of-data problems
  - call these index -1 and 0: then the formulas hold exactly
- When $c_n(w,h) = 0$:
  - Assign 0 probability to $p_n(w|h)$ where $c_{n-1}(h) > 0$, but a uniform probability $(1/|V|)$ to those $p_n(w|h)$ where $c_{n-1}(h) = 0$ [this must be done both when working on the heldout data during EM, as well as when computing cross-entropy on the test data!]