Log-Linear Models
a.k.a. Logistic Regression, Maximum Entropy Models

Natural Language Processing
CS 6120—Spring 2014
Northeastern University

David Smith
(some slides from Jason Eisner and Dan Klein)

Probability is Useful

Probability is Useful

- We love probability distributions!

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1
- Pick best annotation T for a fixed input I

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1
- Pick best annotation T for a fixed input I
  - text categorization; parsing; POS tagging; language ID …

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1
- Pick best annotation T for a fixed input I
  - text categorization; parsing; POS tagging; language ID …
  - maximize $p(T \mid I)$; equivalently maximize joint probability $p(I,T)$

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1
- Pick best annotation T for a fixed input I
  - text categorization; parsing; POS tagging; language ID …
  - maximize p(T | I); equivalently maximize joint probability p(I,T)
    - often define p(I,T) by noisy channel: p(I,T) = p(T) * p(I | T)

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **<u>use</u>** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1
- Pick best annotation T for a fixed input I
  - text categorization; parsing; POS tagging; language ID …
  - maximize p(T | I); equivalently maximize joint probability p(I,T)
    - often define p(I,T) by noisy channel: p(I,T) = p(T) * p(I | T)
  - speech recognition & other tasks above are cases of this too:

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1
- Pick best annotation T for a fixed input I
  - text categorization; parsing; POS tagging; language ID …
  - maximize p(T | I); equivalently maximize joint probability p(I,T)
    - often define p(I,T) by noisy channel: p(I,T) = p(T) * p(I | T)
  - speech recognition & other tasks above are cases of this too:
    - we're maximizing an appropriate $p_1(T)$ defined by p(T | I)

summary of half of the course (statistics)

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1
- Pick best annotation T for a fixed input I
  - text categorization; parsing; POS tagging; language ID …
  - maximize p(T | I); equivalently maximize joint probability p(I,T)
    - often define p(I,T) by noisy channel: p(I,T) = p(T) * p(I | T)
  - speech recognition & other tasks above are cases of this too:
    - we're maximizing an appropriate $p_1(T)$ defined by p(T | I)
- Pick best probability distribution (a meta-problem!)

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1
- Pick best annotation T for a fixed input I
  - text categorization; parsing; POS tagging; language ID …
  - maximize p(T | I); equivalently maximize joint probability p(I,T)
    - often define p(I,T) by noisy channel: p(I,T) = p(T) * p(I | T)
  - speech recognition & other tasks above are cases of this too:
    - we're maximizing an appropriate $p_1(T)$ defined by p(T | I)
- Pick best probability distribution (a meta-problem!)
  - really, pick best parameters θ: train HMM, PCFG, n-grams, clusters …

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1
- Pick best annotation T for a fixed input I
  - text categorization; parsing; POS tagging; language ID …
  - maximize p(T | I); equivalently maximize joint probability p(I,T)
    - often define p(I,T) by noisy channel: p(I,T) = p(T) * p(I | T)
  - speech recognition & other tasks above are cases of this too:
    - we're maximizing an appropriate $p_1(T)$ defined by p(T | I)
- Pick best probability distribution (a meta-problem!)
  - really, pick best <u>parameters</u> θ: train HMM, PCFG, n-grams, clusters …
  - maximum likelihood; smoothing; EM if unsupervised (incomplete data)

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1
- Pick best annotation T for a fixed input I
  - text categorization; parsing; POS tagging; language ID …
  - maximize p(T | I); equivalently maximize joint probability p(I,T)
    - often define p(I,T) by noisy channel: p(I,T) = p(T) * p(I | T)
  - speech recognition & other tasks above are cases of this too:
    - we're maximizing an appropriate $p_1(T)$ defined by p(T | I)
- Pick best probability distribution (a meta-problem!)
  - really, pick best parameters θ: train HMM, PCFG, n-grams, clusters …
  - maximum likelihood; smoothing; EM if unsupervised (incomplete data)
  - Bayesian smoothing: max $p(\theta|\text{data})$ = max $p(\theta, \text{data}) = p(\theta)p(\text{data}|\theta)$

Probability is Flexible

Probability is Flexible

- We love probability distributions!

Probability is Flexible

- We love probability distributions!
  - We've learned how to **define** & use p(…) functions.

Probability is Flexible

- We love probability distributions!
  - We've learned how to **define** & use p(…) functions.
- We want p(…) to define probability of linguistic objects

Probability is Flexible

- We love probability distributions!
  - We've learned how to **define** & use p(…) functions.
- We want p(…) to define probability of linguistic objects
  - Trees of (non)terminals (PCFGs; CKY, Earley, pruning, inside-outside)

Probability is Flexible

- We love probability distributions!
  - We've learned how to **define** & use p(…) functions.
- We want p(…) to define probability of linguistic objects
  - Trees of (non)terminals (PCFGs; CKY, Earley, pruning, inside-outside)
  - Sequences of words, tags, morphemes, phonemes (n-grams, FSAs, FSTs; regex compilation, best-paths, forward-backward, collocations)

Probability is Flexible

- We love probability distributions!
  - We've learned how to **define** & use p(…) functions.
- We want p(…) to define probability of linguistic objects
  - Trees of (non)terminals (PCFGs; CKY, Earley, pruning, inside-outside)
  - Sequences of words, tags, morphemes, phonemes (n-grams, FSAs, FSTs; regex compilation, best-paths, forward-backward, collocations)
  - Vectors (clusters)

Probability is Flexible

- We love probability distributions!
  - We've learned how to **define** & use p(…) functions.
- We want p(…) to define probability of linguistic objects
  - Trees of (non)terminals (PCFGs; CKY, Earley, pruning, inside-outside)
  - Sequences of words, tags, morphemes, phonemes (n-grams, FSAs, FSTs; regex compilation, best-paths, forward-backward, collocations)
  - Vectors (clusters)
- NLP also includes some not-so-probabilistic stuff

Probability is Flexible

- We love probability distributions!
  - We've learned how to **define** & use p(…) functions.
- We want p(…) to define probability of linguistic objects
  - Trees of (non)terminals (PCFGs; CKY, Earley, pruning, inside-outside)
  - Sequences of words, tags, morphemes, phonemes (n-grams, FSAs, FSTs; regex compilation, best-paths, forward-backward, collocations)
  - Vectors (clusters)
- NLP also includes some not-so-probabilistic stuff
  - Syntactic features, morph. Could be stochasticized?

Probability is Flexible

- We love probability distributions!
  - We've learned how to **define** & use p(…) functions.
- We want p(…) to define probability of linguistic objects
  - Trees of (non)terminals (PCFGs; CKY, Earley, pruning, inside-outside)
  - Sequences of words, tags, morphemes, phonemes (n-grams, FSAs, FSTs; regex compilation, best-paths, forward-backward, collocations)
  - Vectors (clusters)
- NLP also includes some not-so-probabilistic stuff
  - Syntactic features, morph. Could be stochasticized?
  - Methods can be quantitative & data-driven but not fully probabilistic: transf.-based learning, bottom-up clustering, LSA, competitive linking

Probability is Flexible

- We love probability distributions!
  - We've learned how to **<u>define</u>** & use p(…) functions.
- We want p(…) to define probability of linguistic objects
  - Trees of (non)terminals (PCFGs; CKY, Earley, pruning, inside-outside)
  - Sequences of words, tags, morphemes, phonemes (n-grams, FSAs, FSTs; regex compilation, best-paths, forward-backward, collocations)
  - Vectors (clusters)
- NLP also includes some not-so-probabilistic stuff
  - Syntactic features, morph. Could be stochasticized?
  - Methods can be quantitative & data-driven but not fully probabilistic: transf.-based learning, bottom-up clustering, LSA, competitive linking
- But probabilities have wormed their way into most things

Probability is Flexible

- We love probability distributions!
  - We've learned how to **define** & use p(…) functions.
- We want p(…) to define probability of linguistic objects
  - Trees of (non)terminals (PCFGs; CKY, Earley, pruning, inside-outside)
  - Sequences of words, tags, morphemes, phonemes (n-grams, FSAs, FSTs; regex compilation, best-paths, forward-backward, collocations)
  - Vectors (clusters)
- NLP also includes some not-so-probabilistic stuff
  - Syntactic features, morph. Could be stochasticized?
  - Methods can be quantitative & data-driven but not fully probabilistic: transf.-based learning, bottom-up clustering, LSA, competitive linking
- But probabilities have wormed their way into most things
- **p(…) has to capture our intuitions about the ling. data**

An Alternative Tradition

An Alternative Tradition

- Old AI hacking technique:
  - Possible parses (or whatever) have scores.
  - Pick the one with the best score.
  - How do you define the score?
    - Completely ad hoc!
    - Throw anything you want into the stew
    - Add a bonus for this, a penalty for that, etc.

An Alternative Tradition

- Old AI hacking technique:
  - Possible parses (or whatever) have scores.
  - Pick the one with the best score.
  - How do you define the score?
    - Completely ad hoc!
    - Throw anything you want into the stew
    - Add a bonus for this, a penalty for that, etc.

- "Learns" over time – as you adjust bonuses and penalties by hand to improve performance. ☺

An Alternative Tradition

- Old AI hacking technique:
  - Possible parses (or whatever) have scores.
  - Pick the one with the best score.
  - How do you define the score?
    - Completely ad hoc!
    - Throw anything you want into the stew
    - Add a bonus for this, a penalty for that, etc.

- "Learns" over time – as you adjust bonuses and penalties by hand to improve performance. ☺
- Total kludge, but totally flexible too …
  - Can throw in **any** intuitions you might have

really so alternative?

An ~~Alternative~~ Tradition

- Old AI hacking technique:
  - Possible parses (or whatever) have scores.
  - Pick the one with the best score.
  - How do you define the score?
    - Completely ad hoc!
    - Throw anything you want into the stew
    - Add a bonus for this, a penalty for that, etc.
- "Learns" over time – as you adjust bonuses and penalties by hand to improve performance. ☺
- Total kludge, but totally flexible too …
  - Can throw in **any** intuitions you might have

really so alternative?

An ~~Alternative~~ Tradition

- Old AI hacking technique:
  - Possible parses (or whatever) have scores.
  - Pick the one with the best score.
  - How do you define the score?
    - Completely ad hoc!
    - Throw anything you want into the stew
    - Add a bonus for this, a penalty for that, etc.
- "Learns" over time – as you adjust bonuses and penalties by hand to improve performance. ☺
- Total kludge, but totally flexible too …
  - Can throw in **any** intuitions you might have

really so alternative?

An ~~Alternative~~ Tradition

- Old A
  - Pos
  - Pic
  - Ho
- "Lear ... uses and pena ... nce. ☺
- Total
  - Ca ... e

Exposé at 9

Probabilistic Revolution Not Really a Revolution, Critics Say

Log-probabilities no more than scores in disguise

"We're just adding stuff up like the old corrupt regime did," admits spokesperson

Nuthin' but adding weights

Nuthin' but adding weights

- n-grams: … + log p(w7 | w5,w6) + log(w8 | w6, w7) + …

Nuthin' but adding weights

- n-grams: … + log p(w7 | w5,w6) + log(w8 | w6, w7) + …
- PCFG: log p(NP VP | S) + log p(Papa | NP) + log p(VP PP | VP) …

Nuthin' but adding weights

- n-grams: … + log p(w7 | w5,w6) + log(w8 | w6, w7) + …
- PCFG: log p(NP VP | S) + log p(Papa | NP) + log p(VP PP | VP) …
- HMM tagging: … + log p(t7 | t5, t6) + log p(w7 | t7) + …

Nuthin' but adding weights

- n-grams: … + log p(w7 | w5,w6) + log(w8 | w6, w7) + …
- PCFG: log p(NP VP | S) + log p(Papa | NP) + log p(VP PP | VP) …
- HMM tagging: … + log p(t7 | t5, t6) + log p(w7 | t7) + …
- Noisy channel: [log p(source)] + [log p(data | source)]

Nuthin' but adding weights

- n-grams: … + log p(w7 | w5,w6) + log(w8 | w6, w7) + …
- PCFG: log p(NP VP | S) + log p(Papa | NP) + log p(VP PP | VP) …
- HMM tagging: … + log p(t7 | t5, t6) + log p(w7 | t7) + …
- Noisy channel: [log p(source)] + [log p(data | source)]
- Cascade of FSTs:

  [log p(A)] + [log p(B | A)] + [log p(C | B)] + …

Nuthin' but adding weights

- n-grams: … + log p(w7 | w5,w6) + log(w8 | w6, w7) + …
- PCFG: log p(NP VP | S) + log p(Papa | NP) + log p(VP PP | VP) …
- HMM tagging: … + log p(t7 | t5, t6) + log p(w7 | t7) + …
- Noisy channel: [log p(source)] + [log p(data | source)]
- Cascade of FSTs:
  [log p(A)] + [log p(B | A)] + [log p(C | B)] + …
- Naïve Bayes:
  log p(Class) + log p(feature1 | Class) + log p(feature2 | Class) …

Nuthin' but adding weights

- n-grams: … + log p(w7 | w5,w6) + log(w8 | w6, w7) + …
- PCFG: log p(NP VP | S) + log p(Papa | NP) + log p(VP PP | VP) …
- HMM tagging: … + log p(t7 | t5, t6) + log p(w7 | t7) + …
- Noisy channel: [log p(source)] + [log p(data | source)]
- Cascade of FSTs:

  [log p(A)] + [log p(B | A)] + [log p(C | B)] + …
- Naïve Bayes:

  log p(Class) + log p(feature1 | Class) + log p(feature2 | Class) …
- *Note:* Today we'll use +logprob not –logprob: i.e., bigger weights are **better**.

Nuthin' but adding weights

- n-grams: … + log p(w7 | w5,w6) + log(w8 | w6, w7) + …
- PCFG: log p(NP VP | S) + log p(Papa | NP) + log p(VP PP | VP) …
  - Can regard any linguistic object as a collection of features (here, tree = a collection of context-free rules)
  - Weight of the object = total weight of features
  - Our weights have always been conditional log-probs (≤ 0)
    - but that is going to change in a few minutes!
- HMM tagging: … + log p(t7 | t5, t6) + log p(w7 | t7) + …
- Noisy channel: [log p(source)] + [log p(data | source)]
- Cascade of FSTs:

  [log p(A)] + [log p(B | A)] + [log p(C | B)] + …
- Naïve Bayes:

  log(Class) + log(feature1 | Class) + log(feature2 | Class) + …

Probabilists Rally Behind Paradigm

Probabilists Rally Behind Paradigm

".2, .4, .6, .8! We're not gonna take your bait!"

Probabilists Rally Behind Paradigm

".2, .4, .6, .8! We're not gonna take your bait!"

1. Can estimate <u>our</u> parameters automatically
   - e.g., log p(t7 | t5, t6) (trigram tag probability)
   - from supervised or unsupervised data

Probabilists Rally Behind Paradigm

".2, .4, .6, .8! We're not gonna take your bait!"

1. Can estimate <u>our</u> parameters automatically
   - e.g., log p(t7 | t5, t6) (trigram tag probability)
   - from supervised or unsupervised data
2. <u>Our</u> results are more meaningful
   - Can use probabilities to place bets, quantify risk
   - e.g., how sure are we that this is the correct parse?

Probabilists Rally Behind Paradigm

".2, .4, .6, .8! We're not gonna take your bait!"

1. Can estimate our parameters automatically
   - e.g., log p(t7 | t5, t6) (trigram tag probability)
   - from supervised or unsupervised data
2. Our results are more meaningful
   - Can use probabilities to place bets, quantify risk
   - e.g., how sure are we that this is the correct parse?
3. Our results can be meaningfully combined ⇒ modularity!
   - Multiply indep. conditional probs – normalized, unlike scores
   - p(English text) * p(English phonemes | English text) * p(Jap. phonemes | English phonemes) * p(Jap. text | Jap. phonemes)
   - p(semantics) * p(syntax | semantics) * p(morphology | syntax) * p(phonology | morphology) * p(sounds | phonology)

Probabilists Rally Behind ^83% of^ Paradigm

".2, .4, .6, .8! We're not gonna take your bait!"

1. Can estimate <u>our</u> parameters automatically
   - e.g., log p(t7 | t5, t6) (trigram tag probability)
   - from supervised or unsupervised data
2. <u>Our</u> results are more meaningful
   - Can use probabilities to place bets, quantify risk
   - e.g., how sure are we that this is the correct parse?
3. <u>Our</u> results can be meaningfully combined ⇒ modularity!
   - Multiply indep. conditional probs – normalized, unlike scores
   - p(English text) * p(English phonemes | English text) * p(Jap. phonemes | English phonemes) * p(Jap. text | Jap. phonemes)
   - p(semantics) * p(syntax | semantics) * p(morphology | syntax) * p(phonology | morphology) * p(sounds | phonology)

Probabilists Regret Being Bound by Principle

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`
- Some useful features:
  - Contains `Buy`
  - Contains `supercalifragilistic`
  - Contains a dollar amount under `$100`
  - Contains an imperative sentence
  - Reading level = 8th grade
  - Mentions money (use word classes and/or regexp to detect this)

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`
- Some useful features:
  - Contains `Buy`
  - Contains `supercalifragilistic`
  - Contains a dollar amount under `$100`
  - Contains an imperative sentence
  - Reading level = 8th grade
  - Mentions money (use word classes and/or regexp to detect this)
- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`
- Some useful features:
  - Contains `Buy`
  - Contains `supercalifragilistic`
  - Contains a dollar amount under `$100`
  - Contains an imperative sentence
  - Reading level = 8^{th} grade
  - Mentions money (use word classes and/or regexp to detect this)
- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`
- Some useful features:

spam | ling

  - Contains `Buy`
  - .5 | .02 Contains `supercalifragilistic`
  - Contains a dollar amount under `$100`
  - Contains an imperative sentence
  - .9 | .1 Reading level = 8th grade
  - Mentions money (use word classes and/or regexp to detect this)
- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`
- Some useful features:

spam | ling

.5 | .02
  - Contains a dollar amount under `$100`

.9 | .1
  - Mentions money
- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`
- Some useful features:

spam ling

50% of spam has this – 25x more likely than in ling

spam	ling	
.5	.02	Contains a dollar amount under `$100`
.9	.1	Mentions money

- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`
- Some useful features:

spam ling

50% of spam has this – 25x more likely than in ling

.5 | .02

  - Contains a dollar amount under `$100`

90% of spam has this – 9x more likely than in ling

.9 | .1

  - Mentions money
- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`
- Some useful features:

spam	ling	
.5	.02	Contains a dollar amount under `$100` — 50% of spam has this – 25x more likely than in ling
.9	.1	Mentions money — 90% of spam has this – 9x more likely than in ling

Naïve Bayes claims .5*.9=45% of spam has **both** features – 25*9=225x more likely than in ling.

- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`
- Some useful features:

spam	ling
.5	.02
.9	.1

  - Contains a dollar amount under `$100`
    - 50% of spam has this – 25x more likely than in ling
    - *but **here** are the emails with both features – only 25x!*
  - Mentions money
    - 90% of spam has this – 9x more likely than in ling

Naïve Bayes claims .5*.9=45% of spam has **both** features – 25*9=225x more likely than in ling.

- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Probabilists Regret Being Bound by Principle

- But ad-hoc approach does have one advantage
  - **Can adjust scores to compensate for feature overlap …**
- Some useful features of this message:

spam | ling

.5 | .02
  - Contains a dollar amount under $100

.9 | .1
  - Mentions money
- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Probabilists Regret Being Bound by Principle

- But ad-hoc approach does have one advantage
  - **Can adjust scores to compensate for feature overlap …**
- Some useful features of this message:

log prob

spam	ling	feature	spam	ling
.5	.02	Contains a dollar amount under `$100`	-1	-5.6
.9	.1	Mentions money	-.15	-3.3

- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Probabilists Regret Being Bound by Principle

- But ad-hoc approach does have one advantage
  - **Can adjust scores to compensate for feature overlap …**
- Some useful features of this message:

Feature	spam	ling	log prob spam	log prob ling	adjusted spam	adjusted ling
Contains a dollar amount under `$100`	.5	.02	-1	-5.6	-.85	-2.3
Mentions money	.9	.1	-.15	-3.3	-.15	-3.3

- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Revolution Corrupted by Bourgeois Values

Revolution Corrupted by Bourgeois Values

- Naïve Bayes needs overlapping but independent features

Revolution Corrupted by Bourgeois Values

- Naïve Bayes needs overlapping but independent features
- But not clear how to restructure these features like that:
  - Contains `Buy`
  - Contains `supercalifragilistic`
  - Contains a dollar amount under `$100`
  - Contains an imperative sentence
  - Reading level = 7^{th} grade
  - Mentions money (use word classes and/or regexp to detect this)
  - …

Revolution Corrupted by Bourgeois Values

- Naïve Bayes needs overlapping but independent features
- But not clear how to restructure these features like that:
  - Contains `Buy`
  - Contains `supercalifragilistic`
  - Contains a dollar amount under `$100`
  - Contains an imperative sentence
  - Reading level = 7th grade
  - Mentions money (use word classes and/or regexp to detect this)
  - …
- Boy, we'd like to be able to throw all that useful stuff in without worrying about feature overlap/independence.

Revolution Corrupted by Bourgeois Values

- Naïve Bayes needs overlapping but independent features
- But not clear how to restructure these features like that:
  - Contains `Buy`
  - Contains `supercalifragilistic`
  - Contains a dollar amount under `$100`
  - Contains an imperative sentence
  - Reading level = 7^{th} grade
  - Mentions money (use word classes and/or regexp to detect this)
  - …
- Boy, we'd like to be able to throw all that useful stuff in without worrying about feature overlap/independence.
- Well, maybe we can add up scores and <u>pretend</u> like we got a log probability:

Revolution Corrupted by Bourgeois Values

- Naïve Bayes needs overlapping but independent features
- But not clear how to restructure these features like that:

  +4 - Contains `Buy`

  +0.2 - Contains `supercalifragilistic`

  +1 - Contains a dollar amount under `$100`

  +2 - Contains an imperative sentence

  -3 - Reading level = 7th grade

  +5 - Mentions money (use word classes and/or regexp to detect this)

  … - …

  total: 5.77

- Boy, we'd like to be able to throw all that useful stuff in without worrying about feature overlap/independence.
- Well, maybe we can add up scores and <u>pretend</u> like we got a log probability: **log p(feats | spam) = 5.77**

Revolution Corrupted by Bourgeois Values

- Naïve Bayes needs overlapping but independent features
- But not clear how to restructure these features like that:
  - +4 Contains `Buy`
  - +0.2 Contains `supercalifragilistic`
  - +1 Contains a dollar amount under `$100`
  - +2 Contains an imperative sentence
  - -3 Reading level = 7th grade
  - +5 Mentions money (use word classes and/or regexp to detect this)
  - … …

total: 5.77

- Boy, we'd like to be able to throw all that useful stuff in without worrying about feature overlap/independence.
- Well, maybe we can add up scores and pretend like we got a log probability: **log p(feats | spam) = 5.77**
- Oops, then p(feats | spam) = exp 5.77 = 320.5

Renormalize by 1/Z to get a Log-Linear Model

- p(feats | spam) = exp 5.77 = 320.5

Renormalize by 1/Z to get a Log-Linear Model

- p(feats | spam) = exp 5.77 = 320.5

scale down so everything < 1 and sums to 1!

Renormalize by 1/Z to get a Log-Linear Model

scale down so everything < 1 and sums to 1!

- p(feats | spam) = exp 5.77 = 320.5
- p(m | spam) = (1/Z(λ)) exp $\sum_i \lambda_i f_i(m)$ where

  m is the email message

  λ_i is weight of feature i

  $f_i(m) \in \{0,1\}$ according to whether m has feature i

  More generally, allow $f_i(m)$ = count or strength of feature.

  1/Z(λ) is a normalizing factor making $\sum_m$ p(m | spam)=1

  (summed over all possible messages m! hard to find!)

Renormalize by 1/Z to get a Log-Linear Model

scale down so everything < 1 and sums to 1!

- p(feats | spam) = exp 5.77 = 320.5
- $p(m \mid spam) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m)$ where

  m is the email message

  λ_i is weight of feature i

  $f_i(m) \in \{0,1\}$ according to whether m has feature i

  More generally, allow $f_i(m)$ = count or strength of feature.

  $1/Z(\lambda)$ is a normalizing factor making $\sum_m p(m \mid spam)=1$

  (summed over all possible messages m! hard to find!)
- The weights we add up are basically arbitrary.

Renormalize by 1/Z to get a Log-Linear Model

scale down so everything < 1 and sums to 1!

- p(feats | spam) = exp 5.77 = 320.5
- p(m | spam) = (1/Z(λ)) exp $\sum_i \lambda_i f_i(m)$ where

  m is the email message

  λ_i is weight of feature i

  $f_i(m) \in \{0,1\}$ according to whether m has feature i

  More generally, allow $f_i(m)$ = count or strength of feature.

  1/Z(λ) is a normalizing factor making $\sum_m$ p(m | spam)=1

  (summed over all possible messages m! hard to find!)
- The weights we add up are basically arbitrary.
- They don't have to mean anything, so long as they give us a good probability.

Renormalize by 1/Z to get a Log-Linear Model

scale down so everything < 1 and sums to 1!

- p(feats | spam) = exp 5.77 = 320.5
- $p(m \mid spam) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m)$ where

  m is the email message

  λ_i is weight of feature i

  $f_i(m) \in \{0,1\}$ according to whether m has feature i

  More generally, allow $f_i(m)$ = count or strength of feature.

  $1/Z(\lambda)$ is a normalizing factor making $\sum_m p(m \mid spam)=1$

  (summed over all possible messages m! hard to find!)
- The weights we add up are basically arbitrary.
- They don't have to mean anything, so long as they give us a good probability.
- Why is it called "log-linear"?

Why Bother?

Why Bother?

- Gives us probs, not just scores.
  - Can use ’em to bet, or combine w/ other probs.

Why Bother?

- Gives us probs, not just scores.
  - Can use 'em to bet, or combine w/ other probs.
- We can now learn weights from data!
  - Choose weights λ_i that maximize logprob of labeled training data = $\log \prod_j p(c_j)\, p(m_j \mid c_j)$
    - where $c_j \in \{\text{ling},\text{spam}\}$ is classification of message m_j
    - and $p(m_j \mid c_j)$ is log-linear model from previous slide
  - <u>Convex</u> function – easy to maximize! (why?)

Why Bother?

- Gives us probs, not just scores.
  - Can use 'em to bet, or combine w/ other probs.
- We can now learn weights from data!
  - Choose weights λ_i that maximize logprob of labeled training data = $\log \prod_j p(c_j)\, p(m_j \mid c_j)$
    - where $c_j \in \{\text{ling}, \text{spam}\}$ is classification of message m_j
    - and $p(m_j \mid c_j)$ is log-linear model from previous slide
  - Convex function – easy to maximize! (why?)
- **But:** $p(m_j \mid c_j)$ for a given λ requires $Z(\lambda)$: hard!

Attempt to Cancel out Z

Attempt to Cancel out Z

- Set weights to maximize $\prod_j p(c_j)\, p(m_j \mid c_j)$
  - where $p(m \mid spam) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m)$
  - **But** normalizer $Z(\lambda)$ is awful sum over all possible emails

Attempt to Cancel out Z

- Set weights to maximize $\prod_j p(c_j)\, p(m_j \mid c_j)$
  - where $p(m \mid spam) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m)$
  - **But** normalizer $Z(\lambda)$ is awful sum over all possible emails

- So instead: Maximize $\prod_j p(c_j \mid m_j)$
  - Doesn't model the emails m_j, only their classifications c_j
  - Makes more sense anyway given our feature set

Attempt to Cancel out Z

- Set weights to maximize $\prod_j p(c_j)\, p(m_j \mid c_j)$
  - where $p(m \mid \text{spam}) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m)$
  - **But** normalizer $Z(\lambda)$ is awful sum over all possible emails

- So instead: Maximize $\prod_j p(c_j \mid m_j)$
  - Doesn't model the emails m_j, only their classifications c_j
  - Makes more sense anyway given our feature set

Attempt to Cancel out Z

- Set weights to maximize $\prod_j p(c_j)\, p(m_j \mid c_j)$
  - where $p(m \mid spam) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m)$
  - **But** normalizer $Z(\lambda)$ is awful sum over all possible emails

- So instead: Maximize $\prod_j p(c_j \mid m_j)$
  - Doesn't model the emails m_j, only their classifications c_j
  - Makes more sense anyway given our feature set

- $p(spam \mid m) = p(spam)p(m|spam) / (p(spam)p(m|spam)+p(ling)p(m|ling))$

Attempt to Cancel out Z

- Set weights to maximize $\prod_j p(c_j)\, p(m_j \mid c_j)$
  - where $p(m \mid spam) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m)$
  - **But** normalizer $Z(\lambda)$ is awful sum over all possible emails

- So instead: Maximize $\prod_j p(c_j \mid m_j)$
  - Doesn't model the emails m_j, only their classifications c_j
  - Makes more sense anyway given our feature set

- $p(spam \mid m) = p(spam)p(m|spam) / (p(spam)p(m|spam)+p(ling)p(m|ling))$
- Z appears in both numerator and denominator

Attempt to Cancel out Z

- Set weights to maximize $\prod_j p(c_j)\, p(m_j \mid c_j)$
  - where $p(m \mid spam) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m)$
  - **But** normalizer $Z(\lambda)$ is awful sum over all possible emails

- So instead: Maximize $\prod_j p(c_j \mid m_j)$
  - Doesn't model the emails m_j, only their classifications c_j
  - Makes more sense anyway given our feature set

- $p(spam \mid m) = p(spam)p(m|spam) / (p(spam)p(m|spam)+p(ling)p(m|ling))$
- Z appears in both numerator and denominator
- Alas, doesn't cancel out because Z differs for the spam and ling models

Attempt to Cancel out Z

- Set weights to maximize $\prod_j p(c_j)\ p(m_j \mid c_j)$
  - where $p(m \mid spam) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m)$
  - **But** normalizer $Z(\lambda)$ is awful sum over all possible emails

- So instead: Maximize $\prod_j p(c_j \mid m_j)$
  - Doesn't model the emails m_j, only their classifications c_j
  - Makes more sense anyway given our feature set

- $p(spam \mid m) = p(spam)p(m|spam) / (p(spam)p(m|spam)+p(ling)p(m|ling))$
- Z appears in both numerator and denominator
- Alas, doesn't cancel out because Z differs for the spam and ling models
- But we can fix this …

So: Modify Setup a Bit

So: Modify Setup a Bit

- Instead of having separate models

  p(m|spam)*p(spam) vs. p(m|ling)*p(ling)

So: Modify Setup a Bit

- Instead of having separate models

  p(m|spam)*p(spam) vs. p(m|ling)*p(ling)
- Have just one joint model p(m,c)

  gives us both p(m,spam) and p(m,ling)

So: Modify Setup a Bit

- Instead of having separate models

  p(m|spam)*p(spam) vs. p(m|ling)*p(ling)
- Have just one joint model p(m,c)

  gives us both p(m,spam) and p(m,ling)
- Equivalent to changing feature set to:
  - spam
  - spam and Contains `Buy`
  - spam and Contains `supercalifragilistic`
  - …
  - ling
  - ling and Contains `Buy`
  - ling and Contains `supercalifragilistic`

So: Modify Setup a Bit

- Instead of having separate models

  p(m|spam)*p(spam) vs. p(m|ling)*p(ling)
- Have just one joint model p(m,c)

  gives us both p(m,spam) and p(m,ling)
- Equivalent to changing feature set to:
  - spam
  - spam and Contains `Buy`
  - spam and Contains `supercalifragilistic`
  - …
  - ling
  - ling and Contains `Buy`
  - ling and Contains `supercalifragilistic`
- No <u>real</u> change, but 2 categories now share single feature set and single value of $Z(\lambda)$

So: Modify Setup a Bit

- Instead of having separate models

  p(m|spam)*p(spam) vs. p(m|ling)*p(ling)
- Have just one joint model p(m,c)

  gives us both p(m,spam) and p(m,ling)
- Equivalent to changing feature set to:
  - spam
  - spam and Contains `Buy` ←old spam model's weight for "contains Buy"
  - spam and Contains `supercalifragilistic`
  - …
  - ling
  - ling and Contains `Buy` ←old ling model's weight for "contains Buy"
  - ling and Contains `supercalifragilistic`
- No <u>real</u> change, but 2 categories now share single feature set and single value of Z(λ)

So: Modify Setup a Bit

- Instead of having separate models

  p(m|spam)*p(spam) vs. p(m|ling)*p(ling)
- Have just one joint model p(m,c)

  gives us both p(m,spam) and p(m,ling)
- Equivalent to changing feature set to:
  - spam ← weight of this feature is log p(spam) + a constant
  - spam and Contains `Buy` ←old spam model's weight for "contains Buy"
  - spam and Contains `supercalifragilistic`
  - …
  - ling ← weight of this feature is log p(ling) + a constant
  - ling and Contains `Buy` ←old ling model's weight for "contains Buy"
  - ling and Contains `supercalifragilistic`
- No real change, but 2 categories now share single feature set and single value of Z(λ)

Now we can cancel out Z

Now we can cancel out Z

Now $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$ where $c \in \{\text{ling, spam}\}$

Now we can cancel out Z

Now $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$ where $c \in \{\text{ling, spam}\}$

- Old: choose weights λ_i that maximize prob of labeled training data = $\prod_j p(m_j, c_j)$

Now we can cancel out Z

Now $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$ where $c \in \{\text{ling, spam}\}$

- Old: choose weights λ_i that maximize prob of labeled training data = $\prod_j p(m_j, c_j)$
- New: choose weights λ_i that maximize prob of labels given messages = $\prod_j p(c_j \mid m_j)$

Now we can cancel out Z

Now $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$ where $c \in \{\text{ling, spam}\}$

- Old: choose weights λ_i that maximize prob of labeled training data = $\prod_j p(m_j, c_j)$
- New: choose weights λ_i that maximize prob of labels given messages = $\prod_j p(c_j \mid m_j)$

Now we can cancel out Z

Now $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$ where $c \in \{\text{ling}, \text{spam}\}$

- Old: choose weights λ_i that maximize prob of labeled training data = $\prod_j p(m_j, c_j)$
- New: choose weights λ_i that maximize prob of labels given messages = $\prod_j p(c_j \mid m_j)$
- Now Z cancels out of conditional probability!

Now we can cancel out Z

Now $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$ where $c \in \{ling, spam\}$

- Old: choose weights λ_i that maximize prob of labeled training data = $\prod_j p(m_j, c_j)$
- New: choose weights λ_i that maximize prob of labels given messages = $\prod_j p(c_j \mid m_j)$
- Now Z cancels out of conditional probability!
  - p(spam | m) = p(m,spam) / (p(m,spam) + p(m,ling))

Now we can cancel out Z

Now $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$ where $c \in \{\text{ling, spam}\}$

- Old: choose weights λ_i that maximize prob of labeled training data = $\prod_j p(m_j, c_j)$
- New: choose weights λ_i that maximize prob of labels given messages = $\prod_j p(c_j \mid m_j)$

- Now Z cancels out of conditional probability!
  - $p(\text{spam} \mid m) = p(m,\text{spam}) / (p(m,\text{spam}) + p(m,\text{ling}))$

    $= \exp \sum_i \lambda_i f_i(m,\text{spam}) / (\exp \sum_i \lambda_i f_i(m,\text{spam}) + \exp \sum_i \lambda_i f_i(m,\text{ling}))$

Now we can cancel out Z

Now $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$ where $c \in \{\text{ling, spam}\}$

- Old: choose weights λ_i that maximize prob of labeled training data = $\prod_j p(m_j, c_j)$
- New: choose weights λ_i that maximize prob of labels given messages = $\prod_j p(c_j \mid m_j)$

- Now Z cancels out of conditional probability!
  - $p(\text{spam} \mid m) = p(m,\text{spam}) / (p(m,\text{spam}) + p(m,\text{ling}))$

    $= \exp \sum_i \lambda_i f_i(m,\text{spam}) / (\exp \sum_i \lambda_i f_i(m,\text{spam}) + \exp \sum_i \lambda_i f_i(m,\text{ling}))$
  - Easy to compute now …

Now we can cancel out Z

Now $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$ where $c \in \{\text{ling, spam}\}$

- Old: choose weights λ_i that maximize prob of labeled training data = $\prod_j p(m_j, c_j)$
- New: choose weights λ_i that maximize prob of labels given messages = $\prod_j p(c_j \mid m_j)$

- Now Z cancels out of conditional probability!
  - $p(\text{spam} \mid m) = p(m,\text{spam}) / (p(m,\text{spam}) + p(m,\text{ling}))$

    $= \exp \sum_i \lambda_i f_i(m,\text{spam}) / (\exp \sum_i \lambda_i f_i(m,\text{spam}) + \exp \sum_i \lambda_i f_i(m,\text{ling}))$
  - Easy to compute now …
  - $\prod_j p(c_j \mid m_j)$ is still convex, so easy to maximize too

Generative vs. Conditional

- What is the most likely label for a given input?
- How likely is a given label for a given input?
- What is the most likely input value?
- How likely is a given input value?
- How likely is a given input value with a given label?
- What is the most likely label for an input that might have one of two values (but we don't know which)?

Generative vs. Conditional

- What is the most likely label for a given input?
- How likely is a given label for a given input?
- What is the most likely input value?
- How likely is a given input value?
- How likely is a given input value with a given label?
- What is the most likely label for an input that might have one of two values (but we don't know which)?

Maximum Entropy

Maximum Entropy

- Suppose there are 10 classes, A through J.

Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don't give you any other information.

Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don't give you any other information.
- Question: Given message m: what is your guess for $p(C \mid m)$?

Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don't give you any other information.
- Question: Given message m: what is your guess for $p(C \mid m)$?

Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don't give you any other information.
- Question: Given message m: what is your guess for p(C | m)?

- Suppose I tell you that 55% of all messages are in class A.

Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don't give you any other information.
- Question: Given message m: what is your guess for p(C | m)?

- Suppose I tell you that 55% of all messages are in class A.
- Question: Now what is your guess for p(C | m)?

Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don't give you any other information.
- Question: Given message m: what is your guess for p(C | m)?

- Suppose I tell you that 55% of all messages are in class A.
- Question: Now what is your guess for p(C | m)?

Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don't give you any other information.
- Question: Given message m: what is your guess for p(C | m)?

- Suppose I tell you that 55% of all messages are in class A.
- Question: Now what is your guess for p(C | m)?

- Suppose I <u>also</u> tell you that 10% of all messages contain `Buy` and 80% of these are in class A or C.

Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don't give you any other information.
- Question: Given message m: what is your guess for p(C | m)?

- Suppose I tell you that 55% of all messages are in class A.
- Question: Now what is your guess for p(C | m)?

- Suppose I <u>also</u> tell you that 10% of all messages contain `Buy` and 80% of these are in class A or C.
- Question: Now what is your guess for p(C | m), if m contains `Buy`?

Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don't give you any other information.
- Question: Given message m: what is your guess for p(C | m)?

- Suppose I tell you that 55% of all messages are in class A.
- Question: Now what is your guess for p(C | m)?

- Suppose I <u>also</u> tell you that 10% of all messages contain `Buy` and 80% of these are in class A or C.
- Question: Now what is your guess for p(C | m),
  if m contains `Buy`?
- OUCH!

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	0.051	0.0025	0.029	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
Other	0.499	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446

- Column A sums to 0.55 ("55% of all messages are in class A")

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	0.051	0.0025	0.029	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
Other	0.499	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446

- Column A sums to 0.55
- Row `Buy` sums to 0.1 ("10% of all messages contain `Buy`")

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	0.051	0.0025	0.029	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
Other	0.499	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446

- Column A sums to 0.55
- Row `Buy` sums to 0.1
- (`Buy`, A) and (`Buy`, C) cells sum to 0.08 ("80% of the 10%")

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	0.051	0.0025	0.029	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
Other	0.499	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446

- Column A sums to 0.55
- Row `Buy` sums to 0.1
- (`Buy`, A) and (`Buy`, C) cells sum to 0.08 ("80% of the 10%")
- Given these constraints, fill in cells "as equally as possible": maximize the entropy (related to cross-entropy, perplexity)

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	0.051	0.0025	0.029	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
Other	0.499	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446

- Column A sums to 0.55
- Row `Buy` sums to 0.1
- (`Buy`, A) and (`Buy`, C) cells sum to 0.08 ("80% of the 10%")
- Given these constraints, fill in cells "as equally as possible": maximize the entropy (related to cross-entropy, perplexity)

Entropy = -.051 log .051 - .0025 log .0025 - .029 log .029 - …

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	0.051	0.0025	0.029	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
Other	0.499	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446

- Column A sums to 0.55
- Row `Buy` sums to 0.1
- (`Buy`, A) and (`Buy`, C) cells sum to 0.08 ("80% of the 10%")
- Given these constraints, fill in cells "as equally as possible": maximize the entropy (related to cross-entropy, perplexity)

Entropy = -.051 log .051 - .0025 log .0025 - .029 log .029 - …

Largest if probabilities are evenly distributed

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	0.051	0.0025	0.029	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025
Other	0.499	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446

- Column A sums to 0.55
- Row `Buy` sums to 0.1
- (`Buy`, A) and (`Buy`, C) cells sum to 0.08 ("80% of the 10%")
- Given these constraints, fill in cells "as equally as possible": maximize the entropy
- Now p(`Buy`, C) = .029 and p(C | `Buy`) = .29
- We got a compromise: p(C | `Buy`) < p(A | `Buy`) < .55

Generalizing to More Features

<$100

Other

	A	B	C	D	E	F	G	H	...
Buy	0.051	0.0025	0.029	0.0025	0.0025	0.0025	0.0025	0.0025	
Other	0.499	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	0.0446	

What we just did

What we just did

- For each feature (“contains Buy”), see what fraction of training data has it

What we just did

- For each feature ("contains Buy"), see what fraction of training data has it
- Many distributions p(c,m) would predict these fractions (including the unsmoothed one where all mass goes to feature combos we've actually seen)

What we just did

- For each feature ("contains Buy"), see what fraction of training data has it
- Many distributions p(c,m) would predict these fractions (including the unsmoothed one where all mass goes to feature combos we've actually seen)
- Of these, pick distribution that has max entropy

What we just did

- For each feature ("contains Buy"), see what fraction of training data has it
- Many distributions p(c,m) would predict these fractions (including the unsmoothed one where all mass goes to feature combos we've actually seen)
- Of these, pick distribution that has max entropy

What we just did

- For each feature ("contains Buy"), see what fraction of training data has it
- Many distributions p(c,m) would predict these fractions (including the unsmoothed one where all mass goes to feature combos we've actually seen)
- Of these, pick distribution that has max entropy

- Amazing Theorem: This distribution has the form
  $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$
  - So it is log-linear. In fact it is the same log-linear distribution that maximizes $\prod_j p(m_j, c_j)$ as before!

What we just did

- For each feature ("contains Buy"), see what fraction of training data has it
- Many distributions p(c,m) would predict these fractions (including the unsmoothed one where all mass goes to feature combos we've actually seen)
- Of these, pick distribution that has max entropy

- Amazing Theorem: This distribution has the form

  $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$

  - So it is log-linear. In fact it is the same log-linear distribution that maximizes $\prod_j p(m_j, c_j)$ as before!

  - Gives another motivation for the log-linear approach.

Log-linear form derivation

- Say we are given some ***constraints*** in the form of feature expectations:

$$\sum_x p(x) f_i(x) = \alpha_i$$

- In general, there may be many distributions p(x) that satisfy the constraints. Which one to pick?
- The one with maximum entropy (making fewest possible additional assumptions---Occum's Razor)
- This yields an optimization problem

$$\max H(p(x)) = -\sum_x p(x) \log p(x)$$

$$\text{Subject to } \sum_x p(x) f_i(x) = \alpha_i, \forall i \text{ and } \sum_x p(x) = 1$$

Log-linear form derivation

- To solve the maxent problem, we use Lagrange multipliers:

$$L = -\sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x}) - \sum_i \theta_i \left(\sum_{\mathbf{x}} p(\mathbf{x}) f_i(\mathbf{x}) - \alpha_i \right) - \mu \left(\sum_{\mathbf{x}} p(\mathbf{x}) - 1 \right)$$

$$\frac{\partial L}{\partial p(\mathbf{x})} = 1 + \log p(\mathbf{x}) - \sum_i \theta_i f_i(\mathbf{x}) - \mu$$

$$p^*(\mathbf{x}) = e^{\mu - 1} \exp \left\{ \sum_i \theta_i f_i(\mathbf{x}) \right\}$$

$$Z(\theta) = e^{1-\mu} = \sum_{\mathbf{x}} \exp \left\{ \sum_i \theta_i f_i(\mathbf{x}) \right\}$$

$$p(\mathbf{x}|\theta) = \frac{1}{Z(\theta)} \exp \left\{ \sum_i \theta_i f_i(\mathbf{x}) \right\}$$

- So feature constraints + maxent implies exponential family.
- Problem is convex, so solution is unique.

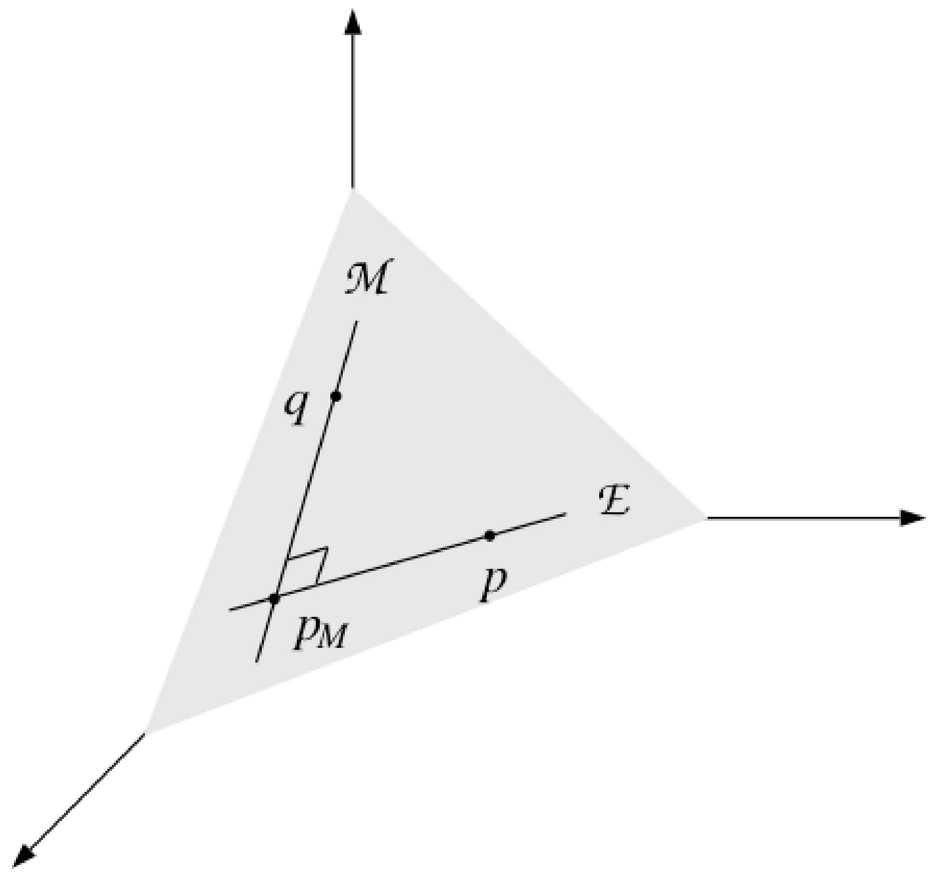
MaxEnt = Max Likelihood

Define two submanifolds on the probability simplex $p(\mathbf{x})$.

The first is $\mathcal{E}$, the set of all exponential family distributions based on a particular set of features $f_i(\mathbf{x})$.

The second is $\mathcal{M}$, the set of all distributions that satisfy the feature expectation constraints.

They intersect at a single distribution p_M, the maxent, maximum likelihood



Exponential Model Likelihood

- Maximum Likelihood (Conditional) Models :
  - Given a model form, choose values of parameters to maximize the (conditional) likelihood of the data.

- Exponential model form, for a data set (C,D):

$$\log P(C \mid D, \lambda) = \sum_{(c,d)\in(C,D)} \log P(c \mid d, \lambda) = \sum_{(c,d)\in(C,D)} \log \frac{\exp \sum_i \lambda_i f_i(c,d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c',d)}$$

Building a Maxent Model

- Define features (indicator functions) over data points.
  - Features represent sets of data points which are distinctive enough to deserve model parameters.
  - Usually features are added incrementally to “target” errors.

- For any given feature weights, we want to be able to calculate:
  - Data (conditional) likelihood
  - Derivative of the likelihood wrt each feature weight
    - Use expectations of each feature according to the model

- Find the optimum feature weights (next part).

The Likelihood Value

- The (log) conditional likelihood is a function of the iid data (C,D) and the parameters λ:

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

- If there aren't many values of *c*, it's easy to calculate:

$$\log P(C \mid D, \lambda) = \sum_{(c,d) \in (C,D)} \log \frac{\exp \sum_i \lambda_i f_i(c,d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c,d)}$$

- We can separate this into two components:

$$\log P(C \mid D, \lambda) = \sum_{(c,d) \in (C,D)} \log \exp \sum_i \lambda_i f_i(c,d) - \sum_{(c,d) \in (C,D)} \log \sum_{c'} \exp \sum_i \lambda_i f_i(c',d)$$

$$\log P(C \mid D, \lambda) = N(\lambda) - M(\lambda)$$

- The derivative is the difference between the derivatives of each component

The Derivative I: Numerator

$$\frac{\partial N(\lambda)}{\partial \lambda_i} = \frac{\partial \sum_{(c,d)\in(C,D)} \log \exp \sum_i \lambda_{ci} f_i(c,d)}{\partial \lambda_i} = \frac{\partial \sum_{(c,d)\in(C,D)} \sum_i \lambda_i f_i(c,d)}{\partial \lambda_i}$$

$$= \sum_{(c,d)\in(C,D)} \frac{\partial \sum_i \lambda_i f_i(c,d)}{\partial \lambda_i}$$

$$= \sum_{(c,d)\in(C,D)} f_i(c,d)$$

Derivative of the numerator is: the empirical count(f_i, c)

The Derivative II: Denominator

$$\frac{\partial M(\lambda)}{\partial \lambda_i} = \frac{\partial \sum_{(c,d)\in(C,D)} \log \sum_{c'} \exp \sum_i \lambda_i f_i(c',d)}{\partial \lambda_i}$$

$$= \sum_{(c,d)\in(C,D)} \frac{1}{\sum_{c''} \exp \sum_i \lambda_i f_i(c'',d)} \frac{\partial \sum_{c'} \exp \sum_i \lambda_i f_i(c',d)}{\partial \lambda_i}$$

$$= \sum_{(c,d)\in(C,D)} \frac{1}{\sum_{c''} \exp \sum_i \lambda_i f_i(c'',d)} \sum_{c'} \frac{\exp \sum_i \lambda_i f_i(c',d)}{1} \frac{\partial \sum_i \lambda_i f_i(c',d)}{\partial \lambda_i}$$

$$= \sum_{(c,d)\in(C,D)} \sum_{c'} \frac{\exp \sum_i \lambda_i f_i(c',d)}{\sum_{c''} \exp \sum_i \lambda_i f_i(c'',d)} \frac{\partial \sum_i \lambda_i f_i(c',d)}{\partial \lambda_i}$$

$$= \sum_{(c,d)\in(C,D)} \sum_{c'} P(c'|d,\lambda) f_i(c',d)$$ = predicted count(f_i, λ)

The Derivative III

$$\frac{\partial \log P(C \mid D, \lambda)}{\partial \lambda_i} = \text{actual count}(f_i, C) - \text{predicted count}(f_i, \lambda)$$

- The optimum parameters are the ones for which each feature's predicted expectation equals its empirical expectation. The optimum distribution is:
  - Always unique (but parameters may not be unique)
  - Always exists (if features counts are from actual data).

- Features can have high model expectations (predicted counts) either because they have large weights or because they occur with other features which have large weights.

Summary

- We have a function to optimize:

$$\log P(C \mid D, \lambda) = \sum_{(c,d)\in(C,D)} \log \frac{\exp \sum_i \lambda_i f_i(c,d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c,d)}$$

- We know the function's derivatives:

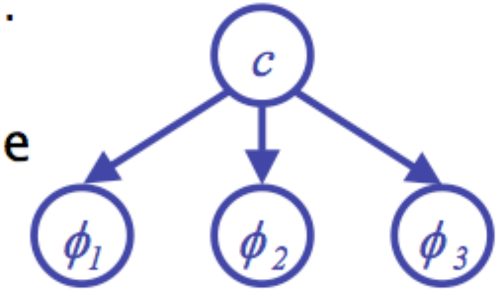
$$\partial \log P(C \mid D, \lambda) / \partial \lambda_i = \text{actual count}(f_i, C) - \text{predicted count}(f_i, \lambda)$$

- Perfect situation for general optimization (Part II)

By gradient ascent or conjugate gradient.

Comparison to Naïve-Bayes

- Naïve-Bayes is another tool for classification:
  - We have a bunch of random variables (data features) which we would like to use to predict another variable (the class):



  - The Naïve-Bayes likelihood over classes is:

$$P(c \mid d, \lambda) = \frac{P(c)\prod_i P(\phi_i \mid c)}{\sum_{c'} P(c')\prod_i P(\phi_i \mid c')} \Rightarrow \frac{\exp\left[\log P(c) + \sum_i \log P(\phi_i \mid c)\right]}{\sum_{c'} \exp\left[\log P(c') + \sum_i \log P(\phi_i \mid c')\right]}$$

Naïve-Bayes is just an exponential model.

$$\Rightarrow \frac{\exp\left[\sum_i \lambda_{ic} f_{ic}(d,c)\right]}{\sum_{c'} \exp\left[\sum_i \lambda_{ic'} f_{ic'}(d,c')\right]}$$

Comparison to Naïve-Bayes

- The primary differences between Naïve-Bayes and maxent models are:

Naïve-Bayes	Maxent
Trained to maximize joint likelihood of data and classes.	Trained to maximize the conditional likelihood of classes.
Features assumed to supply independent evidence.	Features weights take feature dependence into account.
Feature weights can be set independently.	Feature weights must be mutually estimated.
Features must be of the conjunctive $\Phi(d) \wedge c = c_i$ form.	Features need not be of the conjunctive form (but usually are).

Overfitting

- If we have too many features, we can choose weights to model the training data perfectly.

- If we have a feature that only appears in spam training, not ling training, it will get weight ∞ to maximize p(spam | feature) at 1.

- These behaviors overfit the training data.
- Will probably do poorly on test data.

Solutions to Overfitting

Solutions to Overfitting

1. Throw out rare features.
   - Require every feature to occur > 4 times, and > 0 times with ling, and > 0 times with spam.

Solutions to Overfitting

1. Throw out rare features.
   - Require every feature to occur > 4 times, and > 0 times with ling, and > 0 times with spam.
2. Only keep 1000 features.
   - Add one at a time, always greedily picking the one that most improves performance on held-out data.

Solutions to Overfitting

1. Throw out rare features.
   - Require every feature to occur > 4 times, and > 0 times with ling, and > 0 times with spam.
2. Only keep 1000 features.
   - Add one at a time, always greedily picking the one that most improves performance on held-out data.
3. Smooth the observed feature counts.

Solutions to Overfitting

1. Throw out rare features.
   - Require every feature to occur > 4 times, and > 0 times with ling, and > 0 times with spam.
2. Only keep 1000 features.
   - Add one at a time, always greedily picking the one that most improves performance on held-out data.
3. Smooth the observed feature counts.
4. Smooth the weights by using a prior.
   - $\max p(\lambda|\text{data}) = \max p(\lambda, \text{data}) = p(\lambda)p(\text{data}|\lambda)$
   - decree $p(\lambda)$ to be high when most weights close to 0

Smoothing: Priors (MAP)

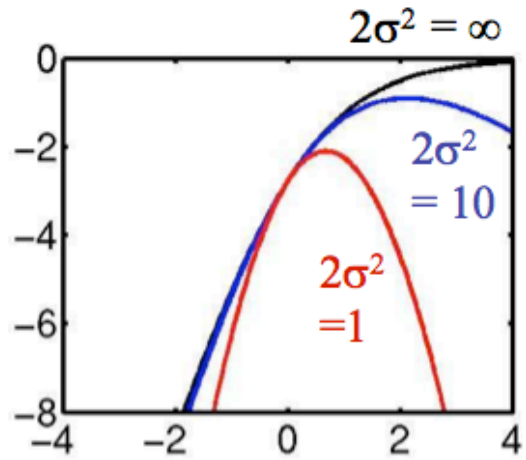
- What if we had a prior expectation that parameter values wouldn't be very large?
- We could then balance evidence suggesting large parameters (or infinite) against our prior.
- The evidence would never totally defeat the prior, and parameters would be smoothed (and kept finite!).
- We can do this explicitly by changing the optimization objective to maximum posterior likelihood:

$$\log P(C, \lambda \mid D) = \log P(\lambda) + \log P(C \mid D, \lambda)$$

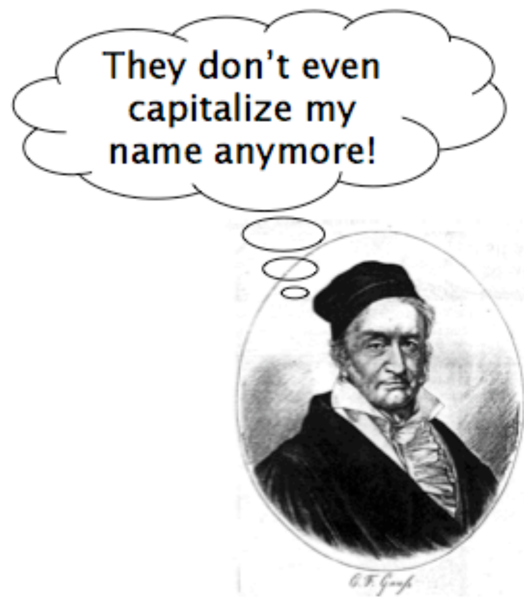
Posterior Prior Evidence

Smoothing: Priors

- Gaussian, or quadratic, priors:
  - Intuition: parameters shouldn't be large.
  - Formalization: prior expectation that each parameter will be distributed according to a gaussian with mean μ and variance σ^2.



$$P(\lambda_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(- \frac{(\lambda_i - \mu_i)^2}{2\sigma_i^2} \right)$$



  - Penalizes parameters for drifting to far from their mean prior value (usually μ=0).
  - $2\sigma^2$=1 works surprisingly well.

Recipe for a Conditional MaxEnt Classifier

1. Gather *constraints* from training data:

$$\alpha_{iy} = \tilde{E}[f_{iy}] = \sum_{x_j, y_j \in D} f_{iy}(x_j, y_j)$$

2. Initialize all parameters to zero.
3. Classify training data with current parameters. Calculate *expectations*.

$$E_{\Theta}[f_{iy}] = \sum_{x_j \in D} \sum_{y'} p_{\Theta}(y'|x_j) f_{iy}(x_j, y')$$

4. Gradient is $\tilde{E}[f_{iy}] - E_{\Theta}[f_{iy}]$
5. Take a step in the direction of the gradient
6. Until convergence, return to step 3.