A linear classifier predicts the label based on a weighted, linear combination of the features:

\[\text{prediction} = w_0 + w_1 f_1 + w_2 f_2 + \ldots + w_m f_m\]

For two classes, a linear classifier can be viewed as a plane (hyperplane) in the feature space.

The Naive Bayes Classifier

**Conditional Independence Assumption:** features are independent of each other given the class.

Learn parameters by maximum likelihood estimation (i.e., maximize likelihood of the training data):

\[\text{label} = \arg \max_{l \in \text{Labels}} p(f_1 | l) p(f_2 | l) \ldots p(f_n | l) p(l)\]
NB is a linear classifier

\[
\text{label} = \arg\max_{l \in \text{Labels}} \log P(f_1 | l) P(f_2 | l) \ldots P(f_n | l) P(l)
\]

\[
= \arg\max_{l \in \text{Labels}} \log P(f_1 | l) + \log P(f_2 | l) + \ldots + \log P(f_n | l) + \log P(l)
\]

\[
= \arg\max_{l \in \text{Labels}} f_1 w_1 + f_2 w_2 + \ldots + w_0
\]

Linear regression

Predict the response based on a weighted, linear combination of the features:

\[
h(\hat{f}) = w_0 + w_1 f_1 + w_2 f_2 + \ldots + w_m f_m
\]

Learn weights by minimizing the square error on the training data:

\[
error(h) = \sum_{i=1}^{n} (y_i - (w_0 + w_1 f_1 + w_2 f_2 + \ldots + w_m f_m))^2
\]

3 views of logistic regression

\[
\log \frac{P(1 | x_1, x_2, \ldots, x_n)}{1 - P(1 | x_1, x_2, \ldots, x_n)} = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \quad \text{linear classifier}
\]

\[
\ldots
\]

\[
P(1 | x_1, x_2, \ldots, x_n) = \frac{e^{w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n}}{1 + e^{w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n}} \quad \text{exponential model (log-linear model)}
\]

\[
\ldots
\]

\[
P(1 | x_1, x_2, \ldots, x_n) = \frac{1}{1 + e^{-w_0 - w_1 x_1 - w_2 x_2 - \ldots - w_n x_n}} \quad \text{logistic}
\]

Logistic regression

\[
\square \text{ Find the best fit of the data based on a logistic function}
\]

![Logistic regression graph]
Training logistic regression models

- How should we learn the parameters for logistic regression (i.e., the $w$'s)?

\[
\log \frac{P(1 \mid x_1, x_2, \ldots, x_m)}{1 - P(1 \mid x_1, x_2, \ldots, x_m)} = w_0 + w_1 x_2 + w_2 x_2 + \ldots + w_m x_m
\]

- Idea 1: minimize the squared error (like linear regression)
  - Any problems?
  - We don’t know what the actual probability values are!

- Idea 2: maximum likelihood training
  - $\text{MLE}(\text{data}) = \arg \max \ p(x) \ \text{data}$
  - $\arg \max \ \sum \ p(w | \text{label}_i, f_i)$
  - $\arg \max \ \sum \ \log p(w | \text{label}_i, f_i)$
    1. plug in our logistic equation
    2. take partial derivatives and solve

Unfortunately, no closed form solution.

Convex functions

- Convex functions look something like:

- What are some nice properties about convex functions?
- How can we find the minimum/maximum of a convex function?
Finding the minimum

You’re blindfolded, but you can see out of the bottom of the blindfold to the ground right by your feet. I drop you off somewhere and tell you that you’re in a convex shaped valley and escape is at the bottom/minimum. How do you get out?

One approach: gradient descent

- Partial derivatives give us the slope in that dimension
- Approach:
  - pick a starting point ($w$)
  - repeat until likelihood can’t increase in any dimension:
    - pick a dimension
    - move a small amount in that dimension towards increasing likelihood (using the derivative)

Gradient descent

- pick a starting point ($w$)
- repeat until loss doesn’t decrease in all dimensions:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

$$w_j = w_j - \alpha \frac{d}{dw_j} \text{error}(w)$$

learning rate (how much we want to move in the error direction)

Solving convex functions

- Gradient descent is just one approach
- A whole field called convex optimization
  - http://www.stanford.edu/~boyd/cvxbook/
- Lots of well known methods
  - Conjugate gradient
  - Generalized Iterative Scaling (GIS)
  - Improved Iterative Scaling (IIS)
  - Limited-memory quasi-Newton (L-BFGS)
- The key: if we get an error function that is convex, we can minimize/maximize it (eventually)
Another thought experiment

What is a 100,000-dimensional space like?

You're a 1-D creature, and you decide to buy a 2-unit apartment

2 rooms (very, skinny rooms)

Another thought experiment

What is a 100,000-dimensional space like?

Your job's going well and you're making good money. You upgrade to a 2-D apartment with 2-units per dimension

4 rooms (very, flat rooms)

Another thought experiment

What is a 100,000-dimensional space like?

You get promoted again and start having kids and decide to upgrade to another dimension.

8 rooms (very, normal rooms)

Each time you add a dimension, the amount of space you have to work with goes up exponentially.

Another thought experiment

What is a 100,000-dimensional space like?

Larry Page steps down as CEO of Google and they ask you if you'd like the job. You decide to upgrade to a 100,000 dimensional apartment.

How much room do you have?
Can you have a big party?

$2^{100,000}$ rooms (it's very quiet and lonely...) $\approx 10^{30}$ rooms per person if you invited everyone on the planet
The challenge

- Because logistic regression has fewer constraints (than, say NB) it has a lot more options
- We’re trying to find 100,000 $w$ values (or a point in a 100,000 dimensional space)
- It’s easy for logistic regression to fit to nuances with the data: overfitting

Preventing overfitting

$$\log \frac{P(y=1|x_1, x_2, ..., x_n)}{1 - P(y=1|x_1, x_2, ..., x_n)} = w_0 + w_1 x_2 + w_2 x_2 + ... + w_n x_n$$

We want to avoid any one features have too much weight

$$\text{MLE}(\text{data}) = \arg \max_\theta \sum_{i=1}^n \log p_\theta(y_i | \tilde{x})$$

ideas?

Preventing overfitting

$$\log \frac{P(y=1|x_1, x_2, ..., x_n)}{1 - P(y=1|x_1, x_2, ..., x_n)} = w_0 + w_1 x_2 + w_2 x_2 + ... + w_n x_n$$

We want to avoid any one features have too much weight

$$\text{MLE}(\text{data}) = \arg \max_\theta \sum_{i=1}^n \log p_\theta(y_i | \tilde{x})$$

normal MLE

$$\text{MLE}(\text{data}) = \arg \max_\theta \sum_{i=1}^n \log p_\theta(y_i | \tilde{x}) - \alpha \sum_{j=1}^m w_j^2$$

regularized MLE
Preventing overfitting: regularization

$$\text{MLE}(\text{data}) = \arg \max_{\theta} \sum_{i=1}^{n} \log p(y_i | \theta) - \alpha \sum_{j=1}^{m} w_j^2$$

Regularized MLE

- What effect will this have on the learned weights assuming a positive $\alpha$?
- Penalize large weights, encourage smaller weights
- Still a convex problem!
- Equivalent to assuming your $w_j$ are distributed from a Gaussian with mean 0

NB vs. Logistic regression

- NB and logistic regression look very similar
  - Both are probabilistic models
  - Both are linear
  - Both learn parameters that maximize the log-likelihood of the training data
- How are they different?

Some historical perspective

http://www.reputation.com/blog/2010/02/17/privacy-a-historical-perspective/
Old school optimization

- 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 mix
  - Add a bonus for this, a penalty for that, etc.
  - Think about state evaluation function for Mancala…

“Learning”
- Adjust bonuses and penalties by hand to improve performance. 😊
- Total kludge, but totally flexible too …
- Can throw in any intuitions you might have
- But we’re purists… we only use probabilities!

New “revolution”?

- Probabilities!

Explored 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
3/4/11

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(\(T_7 | T_5, T_6\)) (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)

83% of Probabilists Regret Being Bound by Principle

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for spam 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)

Any problem with these features for NB?

<table>
<thead>
<tr>
<th></th>
<th>Spam</th>
<th>not-Spam</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; $100</td>
<td>0.5</td>
<td>0.02</td>
</tr>
<tr>
<td>Money amount</td>
<td>0.9</td>
<td>0.1</td>
</tr>
</tbody>
</table>

How likely is it to see both features in either class using NB? Is this right?

<table>
<thead>
<tr>
<th></th>
<th>Spam</th>
<th>not-Spam</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; $100</td>
<td>0.5</td>
<td>0.02</td>
</tr>
<tr>
<td>Money amount</td>
<td>0.9</td>
<td>0.1</td>
</tr>
</tbody>
</table>

0.5*0.9=0.45 0.02*0.1=0.002

Overestimates! The problem is that the features are not independent
NB vs. Logistic regression

- Logistic regression allows us to put in features that overlap and adjust the probabilities accordingly

- Which to use?
  - NB is better for small data sets: strong model assumptions keep the model from overfitting
  - Logistic regression is better for larger data sets: can exploit the fact that NB assumption is rarely true

Logistic regression with more classes

- NB works on multiple classes
- Logistic regression only works on two classes
- Idea: something like logistic regression, but with more classes
  - Like NB, one model per each class
  - The model is a weight vector

\[
P(\text{class} | x_1, x_2, \ldots, x_n) = e^{w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + w_0}
\]

\[
P(\text{class} | x_1, x_2, \ldots, x_n) = e^{w_2 x_1 + w_3 x_2 + \ldots + w_n x_n + w_0}
\]

\[
P(\text{class} | x_1, x_2, \ldots, x_n) = e^{w_3 x_1 + w_4 x_2 + \ldots + w_n x_n + w_0}
\]

... anything wrong with this?
These are supposed to be probabilities!

\[
P(class_1 | x_1, x_2, \ldots, x_n) = e^{w_1 x_1 + w_2 x_2 + \ldots + w_n x_n} / \sum_{i=1}^{C} P(class_i | x_1, x_2, \ldots, x_n)
\]

- still just a linear combination of feature weightings
- class specific features

Normalize each class probability by the sum over all the classes

\[
P(class_i | x_1, x_2, \ldots, x_n) = e^{w_i x_1 + w_i x_2 + \ldots + w_i x_n} / \sum_{i=1}^{C} e^{w_i x_1 + w_i x_2 + \ldots + w_i x_n}
\]

- Can use maximum likelihood training
  
  \[
  MLE(data) = \arg \max \sum \log p(label | \hat{f})
  \]

- Use regularization
  
  \[
  MLE(data) = \arg \max \sum \log p(label | \hat{f}) - cR(\theta)
  \]

- Plug into a convex optimization package
  
  there are a few complications, but this is the basic idea
Maximum Entropy

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

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

Suppose I also tell you that 10% of all examples 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 principle: given the constraints, pick the probabilities as “equally as possible”

Qualitatively

Quantitatively

Maximum entropy: given the constraints, pick the probabilities so as to maximize the entropy

\[
\text{Entropy(model)} = \sum_c p(c) \log p(c)
\]
Maximum Entropy

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy</td>
<td>.051</td>
<td>.0025</td>
<td>.029</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
</tr>
<tr>
<td>Other</td>
<td>.499</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
</tr>
</tbody>
</table>

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

Generalizing to More Features

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy</td>
<td>.051</td>
<td>.0025</td>
<td>.029</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>.499</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td></td>
</tr>
</tbody>
</table>

- 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

Given these constraints, fill in cells "as equally as possible": maximize the entropy
- Given the 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
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
- Of these, pick distribution that has max entropy

- Amazing Theorem: The maximum entropy model is the same as the maximum likelihood model!
  - If we calculate the maximum likelihood parameters, we’re also calculating the maximum entropy model

What to take home...

- Many learning approaches
  - Bayesian approaches (of which NB is just one)
  - Linear regression
  - Logistic regression
  - Maximum Entropy (multinomial logistic regression)
  - SVMs
  - Decision trees
  - …
- Different models have different strengths/weaknesses/uses
  - Understand what the model is doing
  - Understand what assumptions the model is making
  - Pick the model that makes the most sense for your problem/data
  - Feature selection is important

Articles discussion

- What are some challenges?
- Will it work?
- Any concerns/problems with using this type of technology?
- Gaming the system?