General Naïve Bayes

- A general *naive Bayes* model:

\[
P(\text{Cause, Effect}_1 \ldots \text{Effect}_n) = \\
P(\text{Cause}) \prod_i P(\text{Effect}_i | \text{Cause})
\]

- We only specify how each feature depends on the class
- Total number of parameters is *linear* in \( n \)
Example: OCR

\[ P(C) \]

\begin{array}{c|c}
1 & 0.1 \\
2 & 0.1 \\
3 & 0.1 \\
4 & 0.1 \\
5 & 0.1 \\
6 & 0.1 \\
7 & 0.1 \\
8 & 0.1 \\
9 & 0.1 \\
0 & 0.1 \\
\end{array}

\[ P(F_{3,1} = \text{on}|C) \quad P(F_{5,5} = \text{on}|C) \]

\begin{array}{c|c}
1 & 0.01 \\
2 & 0.05 \\
3 & 0.05 \\
4 & 0.30 \\
5 & 0.80 \\
6 & 0.90 \\
7 & 0.05 \\
8 & 0.60 \\
9 & 0.50 \\
0 & 0.80 \\
\end{array}

\begin{array}{c|c}
1 & 0.05 \\
2 & 0.01 \\
3 & 0.90 \\
4 & 0.80 \\
5 & 0.90 \\
6 & 0.90 \\
7 & 0.25 \\
8 & 0.85 \\
9 & 0.60 \\
0 & 0.80 \\
\end{array}
Example: Overfitting

\[ P(\text{features}, C = 2) \]
\[ P(C = 2) = 0.1 \]
\[ P(\text{on}|C = 2) = 0.8 \]
\[ P(\text{on}|C = 2) = 0.1 \]
\[ P(\text{off}|C = 2) = 0.1 \]
\[ P(\text{on}|C = 2) = 0.01 \]

\[ P(\text{features}, C = 3) \]
\[ P(C = 3) = 0.1 \]
\[ P(\text{on}|C = 3) = 0.8 \]
\[ P(\text{on}|C = 3) = 0.9 \]
\[ P(\text{off}|C = 3) = 0.7 \]
\[ P(\text{on}|C = 3) = 0.0 \]

2 wins!!
Example: Spam Filtering

- **Model:** \( P(C, W_1 \ldots W_n) = P(C) \prod_i P(W_i|C) \)
- **Parameters:**

| \( P(C) \) | \( P(W|\text{spam}) \) | \( P(W|\text{ham}) \) |
|---|---|---|
| ham : 0.66 | the : 0.016 | the : 0.021 |
| spam: 0.33 | to : 0.015 | to : 0.013 |
| | and : 0.012 | and : 0.011 |
| | ... | ... |
| | free : 0.005 | free : 0.001 |
| | click : 0.004 | click : 0.001 |
| | ... | ... |
| | screens : 0.001 | screens : 0.000 |
| | minute : 0.001 | minute : 0.000 |
| | ... | ... |
Example: Spam Filtering

- Raw probabilities alone don’t affect the posteriors; relative probabilities (odds ratios) do:

\[
\frac{P(W|\text{ham})}{P(W|\text{spam})} \quad \frac{P(W|\text{spam})}{P(W|\text{ham})}
\]

<table>
<thead>
<tr>
<th>south-west : inf</th>
<th>screens   : inf</th>
</tr>
</thead>
<tbody>
<tr>
<td>nation : inf</td>
<td>minute      : inf</td>
</tr>
<tr>
<td>morally : inf</td>
<td>guaranteed  : inf</td>
</tr>
<tr>
<td>nicely : inf</td>
<td>$205.00     : inf</td>
</tr>
<tr>
<td>extent : inf</td>
<td>delivery    : inf</td>
</tr>
<tr>
<td>seriously : inf</td>
<td>signature   : inf</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

What went wrong here?
Generalization and Overfitting

- Relative frequency parameters will overfit the training data!
  - Unlikely that every occurrence of “minute” is 100% spam
  - Unlikely that every occurrence of “seriously” is 100% ham
  - What about all the words that don’t occur in the training set?
  - In general, we can’t go around giving unseen events zero probability

- As an extreme case, imagine using the entire email as the only feature
  - Would get the training data perfect (if deterministic labeling)
  - Wouldn’t generalize at all
  - Just making the bag-of-words assumption gives us some generalization, but isn’t enough

- To generalize better: we need to smooth or regularize the estimates
Estimation: Smoothing

- Problems with maximum likelihood estimates:
  - If I flip a coin once, and it’s heads, what’s the estimate for $P(\text{heads})$?
  - What if I flip 10 times with 8 heads?
  - What if I flip 10M times with 8M heads?

- Basic idea:
  - We have some prior expectation about parameters (here, the probability of heads)
  - Given little evidence, we should skew towards our prior
  - Given a lot of evidence, we should listen to the data
Estimation: Smoothing

- Relative frequencies are the maximum likelihood estimates

\[
\hat{\theta}_{ML} = \arg \max_\theta P(X|\theta) = \arg \max_\theta \prod_i P_\theta(X_i)
\]

\[
P(x) = \frac{\text{count}(x)}{\text{total samples}}
\]

- In Bayesian statistics, we think of the parameters as just another random variable, with its own distribution

\[
\hat{\theta}_{MAP} = \arg \max_\theta P(\theta|X)
\]

\[
= \arg \max_\theta P(X|\theta) P(\theta) / P(X) \quad \Rightarrow \quad ????
\]

\[
= \arg \max_\theta P(X|\theta) P(\theta)
\]
Estimation: Laplace Smoothing

- Laplace’s estimate:
  - Pretend you saw every outcome once more than you actually did

\[ P_{LAP}(x) = \frac{c(x) + 1}{\sum_x [c(x) + 1]} = \frac{c(x) + 1}{N + |X|} \]

- Can derive this as a MAP estimate with Dirichlet priors (see cs281a)
Estimation: Laplace Smoothing

- Laplace’s estimate (extended):
  - Pretend you saw every outcome k extra times

  \[ P_{LAP,k}(x) = \frac{c(x) + k}{N + k|X|} \]

  - What’s Laplace with k = 0?
  - k is the strength of the prior

- Laplace for conditionals:
  - Smooth each condition independently:

  \[ P_{LAP,k}(x|y) = \frac{c(x, y) + k}{c(y) + k|X|} \]
Estimation: Linear Interpolation

- In practice, Laplace often performs poorly for $P(X|Y)$:
  - When $|X|$ is very large
  - When $|Y|$ is very large

- Another option: linear interpolation
  - Also get $P(X)$ from the data
  - Make sure the estimate of $P(X|Y)$ isn’t too different from $P(X)$

$$P_{LIN}(x|y) = \alpha \hat{P}(x|y) + (1.0 - \alpha) \hat{P}(x)$$

- What if $\alpha$ is 0? 1?

- For even better ways to estimate parameters, as well as details of the math see cs281a, cs294
Real NB: Smoothing

- For real classification problems, smoothing is critical
- New odds ratios:

\[
\frac{P(W|\text{ham})}{P(W|\text{spam})} \quad \frac{P(W|\text{spam})}{P(W|\text{ham})}
\]

<table>
<thead>
<tr>
<th></th>
<th>Helvetica</th>
<th>Seems</th>
<th>Group</th>
<th>Ago</th>
<th>Areas</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ham</td>
<td>11.4</td>
<td>10.8</td>
<td>10.2</td>
<td>8.4</td>
<td>8.3</td>
<td>...</td>
</tr>
<tr>
<td>Spam</td>
<td>28.8</td>
<td>28.4</td>
<td>27.2</td>
<td>26.9</td>
<td>26.5</td>
<td>...</td>
</tr>
</tbody>
</table>

Do these make more sense?
Tuning on Held-Out Data

- Now we’ve got two kinds of unknowns
  - Parameters: the probabilities $P(Y|X)$, $P(Y)$
  - Hyperparameters, like the amount of smoothing to do: $k$, $\alpha$

- Where to learn?
  - Learn parameters from training data
  - Must tune hyperparameters on different data
    - Why?
  - For each value of the hyperparameters, train and test on the held-out data
  - Choose the best value and do a final test on the test data
Baselines

- **First task: get a baseline**
  - Baselines are very simple “straw man” procedures
  - Help determine how hard the task is
  - Help know what a “good” accuracy is

- **Weak baseline: most frequent label classifier**
  - Gives all test instances whatever label was most common in the training set
  - E.g. for spam filtering, might label everything as ham
  - Accuracy might be very high if the problem is skewed

- **For real research, usually use previous work as a (strong) baseline**
Confidences from a Classifier

- **The confidence of a probabilistic classifier:**
  - Posterior over the top label
  
  \[
  \text{confidence}(x) = \max_y P(y|x)
  \]

  - Represents how sure the classifier is of the classification
  - Any probabilistic model will have confidences
  - No guarantee confidence is “correct”

- **Calibration**
  - Weak calibration: higher confidences mean higher accuracy
  - Strong calibration: confidence predicts accuracy rate
  - What’s the use of calibration?
Generative vs. Discriminative

- **Generative classifiers:**
  - E.g. naïve Bayes
  - We build a causal model of the variables
  - We then query that model for causes, given evidence

- **Discriminative classifiers:**
  - E.g. perceptron (next)
  - No causal model, no Bayes rule, often no probabilities
  - Try to predict output directly
  - Loosely: mistake driven rather than model driven
Errors, and What to Do

- **Examples of errors**

Dear GlobalSCAPE Customer,

GlobalSCAPE has partnered with ScanSoft to offer you the latest version of OmniPage Pro, for just $99.99* - the regular list price is $499! The most common question we've received about this offer is - Is this genuine? We would like to assure you that this offer is authorized by ScanSoft, is genuine and valid. You can get the . . .

. . . To receive your $30 Amazon.com promotional certificate, click through to

http://www.amazon.com/apparel

and see the prominent link for the $30 offer. All details are there. We hope you enjoyed receiving this message. However, if you'd rather not receive future e-mails announcing new store launches, please click . . .
What to Do About Errors?

- Need more features—words aren’t enough!
  - Have you emailed the sender before?
  - Have 1K other people just gotten the same email?
  - Is the sending information consistent?
  - Is the email in ALL CAPS?
  - Do inline URLs point where they say they point?
  - Does the email address you by (your) name?

- Naïve Bayes models can incorporate a variety of features, but tend to do best in homogeneous cases (e.g. all features are word occurrences)
Features

- A feature is a function which signals a property of the input.

- Examples:
  - ALL_CAPS: value is 1 iff email in all caps
  - HAS_URL: value is 1 iff email has a URL
  - NUM_URLS: number of URLs in email
  - VERY_LONG: 1 iff email is longer than 1K
  - SUSPICIOUS_SENDER: 1 iff reply-to domain doesn’t match originating server

- Features are anything you can think of code to evaluate on an input.
  - Some cheap, some very very expensive to calculate
  - Can even be the output of another classifier
  - Domain knowledge goes here!

- In naïve Bayes, how did we encode features?
Feature Extractors

- A feature extractor maps inputs to feature vectors

Dear Sir.
First, I must solicit your confidence in this transaction, this is by virtue of its nature as being utterly confidencial and top secret. ...

| W=dear   | 1  |
| W=sir    | 1  |
| W=this   | 2  |
| ...      | ...|
| W=wish   | 0  |
| ...      | ...|
| MISSPELLED | 2 |
| NAMELESS | 1  |
| ALL_CAPS | 0  |
| NUM_URLS | 0  |

- Many classifiers take feature vectors as inputs
- Feature vectors usually very sparse, use sparse encodings (i.e. only represent non-zero keys)
Some (Vague) Biology

- Very loose inspiration: human neurons
The Binary Perceptron

- Inputs are feature values
- Each feature has a weight
- Sum is the activation

\[
activation_w(x) = \sum_i w_i \cdot f_i(x)
\]

- If the activation is:
  - Positive, output 1
  - Negative, output 0
Example: Spam

- Imagine 4 features:
  - Free (number of occurrences of “free”)
  - Money (occurrences of “money”)
  - BIAS (always has value 1)

\[
\begin{array}{ccc}
  x & f(x) & w \\
  \text{BIAS} & 1 & \text{BIAS} \quad -3 \\
  \text{free} & 1 & \text{free} \quad 4 \\
  \text{money} & 1 & \text{money} \quad 2 \\
  \text{the} & 0 & \text{the} \quad 0 \\
  \ldots & \ldots & \ldots \\
\end{array}
\]

\[
\sum_{i} w_i \cdot f_i(x) = 3
\]
Binary Decision Rule

- In the space of feature vectors
  - Any weight vector is a hyperplane
  - One side will be class 1
  - Other will be class -1

\[ w \]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIAS</td>
<td>-3</td>
</tr>
<tr>
<td>free</td>
<td>4</td>
</tr>
<tr>
<td>money</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Multiclass Decision Rule

- If we have more than two classes:
  - Have a weight vector for each class
  - Calculate an activation for each class

\[
\text{activation}_w(x, c) = \sum_i w_{c,i} \cdot f_i(x)
\]

- Highest activation wins

\[
c = \arg \max_c (\text{activation}_w(x, c))
\]
Example

```
Example

“win the vote”

<table>
<thead>
<tr>
<th>wSPORTS</th>
<th>wPOLITICS</th>
<th>wTECH</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIAS : -2</td>
<td>BIAS : 1</td>
<td>BIAS : 2</td>
</tr>
<tr>
<td>win : 4</td>
<td>win : 2</td>
<td>win : 0</td>
</tr>
<tr>
<td>game : 4</td>
<td>game : 0</td>
<td>game : 2</td>
</tr>
<tr>
<td>vote : 0</td>
<td>vote : 4</td>
<td>vote : 0</td>
</tr>
<tr>
<td>the : 0</td>
<td>the : 0</td>
<td>the : 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```
The Perceptron Update Rule

- Start with zero weights
- Pick up training instances one by one
- Try to classify
  \[ c = \arg \max_c \ w_c \cdot f(x) \]
  \[ = \arg \max_c \ \sum_i w_{c,i} \cdot f_i(x) \]
- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer
  \[ w_c = w_c - f(x) \]
  \[ w_{c^*} = w_{c^*} + f(x) \]
Example

“win the vote”
“win the election”
“win the game”

<table>
<thead>
<tr>
<th>wSPORTS</th>
<th>wPOLITICS</th>
<th>wTECH</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIAS :</td>
<td>BIAS :</td>
<td>BIAS :</td>
</tr>
<tr>
<td>win :</td>
<td>win :</td>
<td>win :</td>
</tr>
<tr>
<td>game :</td>
<td>game :</td>
<td>game :</td>
</tr>
<tr>
<td>vote :</td>
<td>vote :</td>
<td>vote :</td>
</tr>
<tr>
<td>the :</td>
<td>the :</td>
<td>the :</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Examples: Perceptron

- Separable Case
Examples: Perceptron

- Separable Case
Mistake-Driven Classification

- In naïve Bayes, parameters:
  - From data statistics
  - Have a causal interpretation
  - One pass through the data

- For the perceptron parameters:
  - From reactions to mistakes
  - Have a discriminative interpretation
  - Go through the data until held-out accuracy maxes out
Properties of Perceptrons

- **Separability:** some parameters get the training set perfectly correct
- **Convergence:** if the training is separable, perceptron will eventually converge (binary case)
- **Mistake Bound:** the maximum number of mistakes (binary case) related to the margin or degree of separability

\[ \text{mistakes} < \frac{1}{\delta^2} \]
Examples: Perceptron

- Non-Separable Case
Examples: Perceptron

- Non-Separable Case
Issues with Perceptrons

- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining isn’t quite as bad as overfitting, but is similar

- Regularization: if the data isn’t separable, weights might thrash around
  - Averaging weight vectors over time can help (averaged perceptron)

- Mediocre generalization: finds a “barely” separating solution
Fixing the Perceptron

- Main problem with perceptron:
  - Update size $\tau$ is uncontrolled
  - Sometimes update way too much
  - Sometimes update way too little

- Solution: choose an update size which
  fixes the current mistake (by 1)...

\[
\min_w \frac{1}{2} \sum_c ||w_c - w'_c||^2
\]

\[
w_{c^*} \cdot f(x) \geq w_c \cdot f(x) + 1
\]

- ... but, choose the minimum change
Minimum Correcting Update

\[
\min_w \frac{1}{2} \sum_c \|w_c - w'_c\|^2 \\
w_{c^*} \cdot f \geq w_c \cdot f + 1
\]

\[
\min_\tau \|\tau f\|^2 \\
w_{c^*} \cdot f \geq w_c \cdot f + 1
\]

\[
(w'_{c^*} + \tau f) \cdot f = (w'_c - \tau f) \cdot f + 1
\]

\[
\tau = \frac{(w'_c - w'_{c^*}) \cdot f + 1}{2f \cdot f}
\]

\[
w_c = w'_c - \tau f(x) \\
w_{c^*} = w'_{c^*} + \tau f(x)
\]

\[
w_{c^*} \cdot f \geq w_c \cdot f + 1
\]

\[
\tau = 0
\]

min not \( \tau = 0 \), or would not have made an error, so min will be where equality holds
MIRA

- In practice, it’s bad to make updates that are too large
  - Example may be labeled incorrectly
  - Solution: cap the maximum possible value of $\tau$

$$\tau^* = \min \left( \frac{(w_c' - w_{c*}') \cdot f + 1}{2f \cdot f}, C \right)$$

- This gives an algorithm called MIRA
  - Usually converges faster than perceptron
  - Usually performs better, especially on noisy data
Linear Separators

- Which of these linear separators is optimal?
Support Vector Machines

- Maximizing the margin: good according to intuition and theory.
- Only support vectors matter; other training examples are ignorable.
- Support vector machines (SVMs) find the separator with max margin.
- Basically, SVMs are MIRA where you optimize over all examples at once.

**MIRA**

\[
\min_w \frac{1}{2}||w - w'||^2 \\
 w_{c^*} \cdot f(x_i) \geq w_c \cdot f(x_i) + 1
\]

**SVM**

\[
\min_w \frac{1}{2}||w||^2 \\
\forall i, c \ w_{c^*} \cdot f(x_i) \geq w_c \cdot f(x_i) + 1
\]
Summary

- **Naïve Bayes**
  - Build classifiers using model of training data
  - Smoothing estimates is important in real systems
  - Classifier confidences are useful, when you can get them

- **Perceptrons / MIRA:**
  - Make less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data
Similarity Functions

- Similarity functions are very important in machine learning

- Topic for next class: kernels
  - Similarity functions with special properties
  - The basis for a lot of advance machine learning (e.g. SVMs)