CSEP 573: Artificial Intelligence

Machine Learning

Agent Testing Today!

slides adapted from
Stuart Russel, Dan Klein, Pieter Abbeel from ai.berkeley.edu
And Hanna Hajishirzi, Jared Moore, Dan Weld
Machine Learning

- Up until now: how use a model to make optimal decisions

- Machine learning: how to acquire a model from data / experience
  - Learning parameters (e.g. probabilities)
  - Learning structure (e.g. BN graphs)
  - Learning hidden concepts (e.g. clustering)

- Today: model-based classification with Naive Bayes
Classification
Example: Spam Filter

- Input: an email
- Output: spam/ham

Setup:
- Get a large collection of example emails, each labeled “spam” or “ham”
- Note: someone has to hand label all this data!
- Want to learn to predict labels of new, future emails

Features: The attributes used to make the ham/spam decision
- Words: FREE!
- Text Patterns: $dd, CAPS
- Non-text: SenderInContacts
- ...
Example: Digit Recognition

- **Input:** images / pixel grids
- **Output:** a digit 0-9

**Setup:**
- Get a large collection of example images, each labeled with a digit
- Note: someone has to hand label all this data!
- Want to learn to predict labels of new, future digit images

**Features:** The attributes used to make the digit decision
- Pixels: (6,8)=ON
- Shape Patterns: NumComponents, AspectRatio, NumLoops
- ...
Other Classification Tasks

- **Classification**: given inputs $x$, predict labels (classes) $y$

- **Examples**:
  - Spam detection (input: document, classes: spam / ham)
  - OCR (input: images, classes: characters)
  - Medical diagnosis (input: symptoms, classes: diseases)
  - Automatic essay grading (input: document, classes: grades)
  - Fraud detection (input: account activity, classes: fraud / no fraud)
  - Customer service email routing
  - ... many more

- Classification is an important commercial technology!
Model-Based Classification
Model-Based Classification

- **Model-based approach**
  - Build a model (e.g. Bayes’ net) where both the label and features are random variables
  - Instantiate any observed features
  - Query for the distribution of the label conditioned on the features

- **Challenges**
  - What structure should the BN have?
  - How should we learn its parameters?
Naïve Bayes for Digits

- Naïve Bayes: Assume all features are independent effects of the label

- Simple digit recognition version:
  - One feature (variable) $F_{ij}$ for each grid position $<i,j>$
  - Feature values are on / off, based on whether intensity is more or less than 0.5 in underlying image
  - Each input maps to a feature vector, e.g.
    
    $1 \rightarrow \{F_{0,0} = 0 \ F_{0,1} = 0 \ F_{0,2} = 1 \ F_{0,3} = 1 \ F_{0,4} = 0 \ \ldots F_{15,15} = 0\}$

  - Here: lots of features, each is binary valued

- Naïve Bayes model:
  
  $P(Y|F_{0,0} \ldots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y)$

- What do we need to learn?
A general Naïve Bayes model:

\[ P(Y, F_1 \ldots F_n) = P(Y) \prod_i P(F_i|Y) \]

- We only have to specify how each feature depends on the class
- Total number of parameters is \textit{linear} in \( n \)
- Model is very simplistic, but often works anyway
Inference for Naïve Bayes

- **Goal:** compute posterior distribution over label variable $Y$
  - Step 1: get joint probability of label and evidence for each label
    
    $P(Y, f_1 \ldots f_n) = \begin{bmatrix} P(y_1, f_1 \ldots f_n) \\ P(y_2, f_1 \ldots f_n) \\ \vdots \\ P(y_k, f_1 \ldots f_n) \end{bmatrix}$
    
    $\frac{P(y_1) \prod_i P(f_i|y_1)}{\sum_k P(y_k) \prod_i P(f_i|y_k)}$

- Step 2: sum to get probability of evidence

- Step 3: normalize by dividing Step 1 by Step 2
  
  $P(Y|f_1 \ldots f_n)$
General Naïve Bayes

What do we need in order to use Naïve Bayes?

- Inference method (we just saw this part)
  - Start with a bunch of probabilities: $P(Y)$ and the $P(F_i|Y)$ tables
  - Use standard inference to compute $P(Y|F_1...F_n)$
  - Nothing new here

- Estimates of local conditional probability tables
  - $P(Y)$, the prior over labels
  - $P(F_i|Y)$ for each feature (evidence variable)
  - These probabilities are collectively called the *parameters* of the model and denoted by $\theta$
  - Up until now, we assumed these appeared by magic, but...
  - ...they typically come from training data counts: we’ll look at this soon
Example: Conditional Probabilities

\[
P(Y)
\]

\[
P(F_{3,1} = \text{on}|Y) \quad P(F_{5,5} = \text{on}|Y)
\]
Naïve Bayes spam filter

Data:
- Collection of emails, labeled spam or ham
- Note: someone has to hand label all this data!
- Split into training, held-out, test sets

Classifiers
- Learn on the training set
- (Tune it on a held-out set)
- Test it on new emails
Naïve Bayes for Text

- **Bag-of-words Naïve Bayes:**
  - Features: $W_i$ is the word at position $i$
  - As before: predict label conditioned on feature variables (spam vs. ham)
  - As before: assume features are conditionally independent given label
  - New: each $W_i$ is identically distributed

- **Generative model:**
  \[ P(Y, W_1 \ldots W_n) = P(Y) \prod_i P(W_i|Y) \]

- **“Tied” distributions and bag-of-words**
  - Usually, each variable gets its own conditional probability distribution $P(F|Y)$
  - In a bag-of-words model
    - Each position is identically distributed
    - All positions share the same conditional probs $P(W|Y)$
    - Why make this assumption?
  - Called “bag-of-words” because model is insensitive to word order or reordering
Example: Spam Filtering

- **Model:**
  
  \[ P(Y, W_1 \ldots W_n) = P(Y) \prod_i P(W_i|Y) \]

- **What are the parameters?**

  \[
  \begin{array}{l}
  P(Y) \\
  \hline
  \text{ham} : 0.66 \\
  \text{spam} : 0.33 \\
  \hline
  \end{array}
  \]

  \[
  \begin{array}{l}
  P(W|\text{spam}) \\
  \hline
  \text{the} : 0.0156 \\
  \text{to} : 0.0153 \\
  \text{and} : 0.0115 \\
  \text{of} : 0.0095 \\
  \text{you} : 0.0093 \\
  \text{a} : 0.0086 \\
  \text{with} : 0.0080 \\
  \text{from} : 0.0075 \\
  \ldots \\
  \hline
  \end{array}
  \]

  \[
  \begin{array}{l}
  P(W|\text{ham}) \\
  \hline
  \text{the} : 0.0210 \\
  \text{to} : 0.0133 \\
  \text{of} : 0.0119 \\
  \text{2002} : 0.0110 \\
  \text{with} : 0.0108 \\
  \text{from} : 0.0107 \\
  \text{and} : 0.0105 \\
  \text{a} : 0.0100 \\
  \ldots \\
  \hline
  \end{array}
  \]

- **Where do these tables come from?**
### Spam Example

| Word    | P(w|spam) | P(w|ham) | Tot Spam | Tot Ham |
|---------|-----------|----------|----------|---------|
| (prior) | 0.33333   | 0.66666  | -1.1     | -0.4    |

\[
P(\text{spam} \mid w) = 98.9
\]
Training and Testing
Important Concepts

- **Data:** labeled instances, e.g. emails marked spam/ham
  - Training set
  - Held out set
  - Test set
- **Features:** attribute-value pairs which characterize each x
- **Experimentation cycle**
  - Learn parameters (e.g. model probabilities) on training set
  - (Tune hyperparameters on held-out set)
  - Compute accuracy of test set
  - Very important: never “peek” at the test set!
- **Evaluation**
  - Accuracy: fraction of instances predicted correctly
- **Overfitting and generalization**
  - Want a classifier which does well on test data
  - Overfitting: fitting the training data very closely, but not generalizing well
  - We’ll investigate overfitting and generalization formally in a few lectures
Generalization and Overfitting
Overfitting

Degree 15 polynomial
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: Overfitting

- Postiors determined by \textit{relative} probabilities (odds ratios):

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

| south-west | inf |
| nation     | inf |
| morally    | inf |
| nicely     | inf |
| extent     | inf |
| seriously  | inf |
| ...        | ... |

| screens    | inf |
| minute     | inf |
| guaranteed | inf |
| $205.00    | inf |
| delivery   | inf |
| signature  | inf |
| ...        | ... |

\textit{What went wrong here?}
Generalization and Overfitting

- Relative frequency parameters will overfit the training data!
  - Just because we never saw a 3 with pixel (15,15) on during training doesn’t mean we won’t see it at test time
  - 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 at all?
  - 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
Parameter Estimation
Parameter Estimation

- Estimating the distribution of a random variable

- **Elicitation:** ask a human (why is this hard?)

- **Empirically:** use training data (learning!)
  - E.g.: for each outcome $x$, look at the *empirical rate* of that value:
    \[
    P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}}
    \]
    - This is the estimate that maximizes the *likelihood of the data*

\[
L(x, \theta) = \prod_i P_\theta(x_i)
\]
Relative frequencies are the maximum likelihood estimates

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

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

Another option is to consider the most likely parameter value given the data

\[ \theta_{MAP} = \arg \max_{\theta} P(\theta|X) \]
\[ = \arg \max_{\theta} P(X|\theta)P(\theta)/P(X) \]
\[ = \arg \max_{\theta} P(X|\theta)P(\theta) \]
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 estimate with Dirichlet priors (see ML class)

\[
P_{ML}(X) =
\]

\[
P_{LAP}(X) =
\]
Laplace Smoothing

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

$$P_{LAP,k}(x) = \frac{c(x) + k}{N + k \mid X}$$

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

- Laplace for conditionals:
  - Smooth each condition independently:

$$P_{LAP,k}(x|y) = \frac{c(x, y) + k}{c(y) + k \mid X}$$

$$P_{LAP,0}(X) =$$

$$P_{LAP,1}(X) =$$

$$P_{LAP,100}(X) =$$
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 the empirical $P(X)$ from the data
  - Make sure the estimate of $P(X|Y)$ isn’t too different from the empirical $P(X)$

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

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

- For even better ways to estimate parameters, as well as details of the math, see advances ML/NLP classes
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>Term</th>
<th>New Odds Ratio</th>
<th>Term</th>
<th>New Odds Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>helvetica</td>
<td>11.4</td>
<td>verdana</td>
<td>28.8</td>
</tr>
<tr>
<td>seems</td>
<td>10.8</td>
<td>Credit</td>
<td>28.4</td>
</tr>
<tr>
<td>group</td>
<td>10.2</td>
<td>ORDER</td>
<td>27.2</td>
</tr>
<tr>
<td>ago</td>
<td>8.4</td>
<td>&lt;FONT&gt;</td>
<td>26.9</td>
</tr>
<tr>
<td>areas</td>
<td>8.3</td>
<td>money</td>
<td>26.5</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

*Do these make more sense?*
Tuning
Now we’ve got two kinds of unknowns
- Parameters: the probabilities $P(X|Y)$, $P(Y)$
- Hyperparameters: e.g. the amount / type of smoothing to do, $k$, $\alpha$

What should we learn where?
- Learn parameters from training data
- 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
Features

- 4 Wheels!
- Larger than a Breadbox
- Made of Metal
- 100,000-mile drivetrain warranty

*Batteries Not Included*
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?

- Can add these information sources as new variables in the NB model

- Next class we’ll talk about classifiers which let you easily add arbitrary features more easily
First step: 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
- E.g. calling everything “ham” gets 66%, so a classifier that gets 70% isn’t very good...

For real research, usually use previous work as a (strong) baseline
The confidence of a probabilistic classifier:
- Posterior over the top label
  \[
  \text{confidence}(x) = \max_y P(y|\mathbf{x})
  \]
- 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 value of calibration?
Bayes rule lets us do diagnostic queries with causal probabilities

The naïve Bayes assumption takes all features to be independent given the class label

We can build classifiers out of a naïve Bayes model using training data

Smoothing estimates is important in real systems

Classifier confidences are useful, when you can get them
Error-Driven Classification
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

- Problem: there’s still spam in your inbox

- Need more features – words aren’t enough!
  - Have you emailed the sender before?
  - Have 1M 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)
Linear Classifiers
Hello,
Do you want free printr cartriges? Why pay more when you can get them ABSOLUTELY FREE! Just

Hello,
Do you want free printr cartriges? Why pay more when you can get them ABSOLUTELY FREE! Just
Some (Simplified) Biology

- Very loose inspiration: human neurons
Linear Classifiers

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

$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output +1
  - Negative, output -1
Weights

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples

\[
\begin{align*}
\begin{array}{l}
\text{# free} : 4 \\
\text{YOUR NAME} : -1 \\
\text{MISSPELLED} : 1 \\
\text{FROM_FRIEND} : -3 \\
\ldots
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{l}
\text{# free} : 2 \\
\text{YOUR NAME} : 0 \\
\text{MISSPELLED} : 0 \\
\text{FROM_FRIEND} : 2 \\
\ldots
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{l}
\text{# free} : 0 \\
\text{YOUR NAME} : 1 \\
\text{MISSPELLED} : 1 \\
\text{FROM_FRIEND} : 1 \\
\ldots
\end{array}
\end{align*}
\]

Dot product \( w \cdot f \) positive means the positive class
Decision Rules
Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to $Y=+1$
  - Other corresponds to $Y=-1$

$$f \cdot w = 0$$

<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>...</td>
<td>...</td>
</tr>
</tbody>
</table>

-1 = HAM
+1 = SPAM
Weight Updates
Learning: Binary Perceptron

- Start with weights = 0
- For each training instance:
  - Classify with current weights
  - If correct (i.e., \( y = y^* \)), no change!
  - If wrong: adjust the weight vector
Learning: Binary Perceptron

- Start with weights = 0
- For each training instance:
  - Classify with current weights
    
    \[
    y = \begin{cases} 
    +1 & \text{if } w \cdot f(x) \geq 0 \\
    -1 & \text{if } w \cdot f(x) < 0 
    \end{cases}
    \]
  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

\[
w = w + y^* \cdot f
\]
Examples: Perceptron

- Separable Case
If we have multiple classes:

- A weight vector for each class:

  \[ w_y \]

- Score (activation) of a class \( y \):

  \[ w_y \cdot f(x) \]

- Prediction highest score wins

\[
y = \arg \max_y \ w_y \cdot f(x)
\]

Binary = multiclass where the negative class has weight zero
Learning: Multiclass Perceptron

- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

\[ y = \arg \max_y \ w_y \cdot f(x) \]

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

\[ w_y = w_y - f(x) \]
\[ w_{y^*} = w_{y^*} + f(x) \]
Example: Multiclass Perceptron

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

<table>
<thead>
<tr>
<th>( w_{SPORTS} )</th>
<th>( w_{POLITICS} )</th>
<th>( w_{TECH} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIAS : 1</td>
<td>BIAS : 0</td>
<td>BIAS : 0</td>
</tr>
<tr>
<td>win : 0</td>
<td>win : 0</td>
<td>win : 0</td>
</tr>
<tr>
<td>game : 0</td>
<td>game : 0</td>
<td>game : 0</td>
</tr>
<tr>
<td>vote : 0</td>
<td>vote : 0</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>
Properties of Perceptrons

- Separability: true if 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

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

- Non-Separable Case
Improving the Perceptron
Problems with the Perceptron

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

- Mediocre generalization: finds a “barely” separating solution

- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting
Fixing the Perceptron

- Idea: adjust the weight update to mitigate these effects
- MIRA*: choose an update size that fixes the current mistake...
- ... but, minimizes the change to $w$

$$\min_w \frac{1}{2} \sum_y ||w_y - w'_y||^2$$

$$w_{y^*} \cdot f(x) \geq w_y \cdot f(x) + 1$$

- The +1 helps to generalize

* Margin Infused Relaxed Algorithm
Minimum Correcting Update

\[
\min_w \frac{1}{2} \sum_y ||w_y - w'_y||^2 \\
w_{y^*} \cdot f \geq w_y \cdot f + 1
\]

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

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

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

\[
w_y = w'_y - \tau f(x) \\
w_{y^*} = w'_{y^*} + \tau f(x)
\]

\[
w_{y^*} \cdot f \geq w_y \cdot f + 1
\]

\[
\tau = 0
\]

min not \(\tau=0\), or would not have made an error, so min will be where equality holds
In practice, it’s also bad to make updates that are too large

- Example may be labeled incorrectly
- You may not have enough features
- Solution: cap the maximum possible value of \( \tau \) with some constant \( C \)

\[
\tau^* = \min \left( \frac{(w'_y - w'_y^*) \cdot f + 1}{2f \cdot f}, C \right)
\]

- Corresponds to an optimization that assumes non-separable data
- Usually converges faster than perceptron
- Usually better, especially on noisy data
Which of these linear separators is optimal?
Support Vector Machines

- Maximizing the margin: good according to intuition, theory, practice
- 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**

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

\[
w_y^* \cdot f(x_i) \geq w_y \cdot f(x_i) + 1
\]

**SVM**

\[
\min_w \frac{1}{2} ||w||^2
\]

\[
\forall i, y \ w_y^* \cdot f(x_i) \geq w_y \cdot f(x_i) + 1
\]
Classification: Comparison

- **Naïve Bayes**
  - Builds a model training data
  - Gives prediction probabilities
  - Strong assumptions about feature independence
  - One pass through data (counting)

- **Perceptrons / MIRA:**
  - Makes less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data (prediction)
  - Often more accurate
Web Search
Extension: Web Search

- Information retrieval:
  - Given information needs, produce information
  - Includes, e.g. web search, question answering, and classic IR

- Web search: not exactly classification, but rather ranking

$x = \text{“Apple Computers”}$
Feature-Based Ranking

Let $x = \text{"Apple Computer"}$

$$f(x, \ldots) = [0.3 \ 5 \ 0 \ 0 \ \ldots]$$

$$f(x, \ldots) = [0.8 \ 4 \ 2 \ 1 \ \ldots]$$
Perceptron for Ranking

- Inputs $x$
- Candidates $y$
- Many feature vectors: $f(x, y)$
- One weight vector: $w$
  - Prediction:
    $$y = \arg \max_y w \cdot f(x, y)$$
  - Update (if wrong):
    $$w = w + f(x, y^*) - f(x, y)$$
Case-Based Learning
Non-Separable Data
Case-Based Reasoning

- **Classification from similarity**
  - Case-based reasoning
  - Predict an instance’s label using similar instances

- **Nearest-neighbor classification**
  - 1-NN: copy the label of the most similar data point
  - K-NN: vote the k nearest neighbors (need a weighting scheme)
  - Key issue: how to define similarity
  - Trade-offs: Small k gives relevant neighbors, Large k gives smoother functions

http://www.cs.cmu.edu/~zhuxj/courseproject/knndemo/KNN.html
Parametric / Non-Parametric

- **Parametric models:**
  - Fixed set of parameters
  - More data means better settings

- **Non-parametric models:**
  - Complexity of the classifier increases with data
  - Better in the limit, often worse in the non-limit

- (K)NN is **non-parametric**

![Diagrams showing examples with increasing data](image-url)
Nearest Neighbor Classification

- Nearest neighbor for digits:
  - Take new image
  - Compare to all training images
  - Assign based on closest example

- Encoding: image is vector of intensities:
  \[ \mathbf{1} = (0.0 \ 0.0 \ 0.3 \ 0.8 \ 0.7 \ 0.1 \ldots \ 0.0) \]

- What’s the similarity function?
  - Dot product of two images vectors?
    \[ \text{sim}(x, x') = x \cdot x' = \sum_i x_i x'_i \]
  - Usually normalize vectors so \(|x| = 1\)
  - min = 0 (when?), max = 1 (when?)
Similarity Functions
Basic Similarity

- Many similarities based on feature dot products:

\[
sim(x, x') = f(x) \cdot f(x') = \sum_i f_i(x)f_i(x')
\]

- If features are just the pixels:

\[
sim(x, x') = x \cdot x' = \sum_i x_ix_i'
\]

- Note: not all similarities are of this form
Better similarity functions use knowledge about vision

Example: invariant metrics:

- Similarities are invariant under certain transformations
- Rotation, scaling, translation, stroke-thickness...
- E.g:

  ![Example images]

  - $16 \times 16 = 256$ pixels; a point in $256$-dim space
  - These points have small similarity in $\mathbb{R}^{256}$ (why?)
  - How can we incorporate such invariances into our similarities?
Rotation Invariant Metrics

- Each example is now a curve in $\mathbb{R}^{256}$
- Rotation invariant similarity:

$$s' = \max s( r(3), r(3) )$$

- E.g. highest similarity between images’ rotation lines
Template Deformation

- **Deformable templates:**
  - An “ideal” version of each category
  - Best-fit to image using min variance
  - Cost for high distortion of template
  - Cost for image points being far from distorted template

- **Used in many commercial digit recognizers**

Examples from [Hastie 94]
A Tale of Two Approaches...

- Nearest neighbor-like approaches
  - Can use fancy similarity functions
  - Don’t actually get to do explicit learning

- Perceptron-like approaches
  - Explicit training to reduce empirical error
  - Can’t use fancy similarity, only linear
  - Or can they? Let’s find out!
Non-Linearity
Non-Linear Separators

- Data that is linearly separable works out great for linear decision rules:

- But what are we going to do if the dataset is just too hard?

- How about... mapping data to a higher-dimensional space:

This and next few slides adapted from Ray Mooney, UT
Non-Linear Separators

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:

$$\Phi: x \rightarrow \phi(x)$$
Recap: Classification

- **Classification systems:**
  - Supervised learning
  - Make a *prediction* given evidence
  - We’ve seen several methods for this
  - Useful when you have *labeled data*
Clustering systems:
- Unsupervised learning
- Detect patterns in unlabeled data
  - E.g. group emails or search results
  - E.g. find categories of customers
  - E.g. detect anomalous program executions
- Useful when don’t know what you’re looking for
- Requires data, but no labels
- Often get gibberish
Clustering
Clustering

- Basic idea: group together similar instances
- Example: 2D point patterns

What could “similar” mean?
- One option: small (squared) Euclidean distance

\[
\text{dist}(x, y) = (x - y)^T (x - y) = \sum_i (x_i - y_i)^2
\]
K-Means
K-Means

- An iterative clustering algorithm
  - Pick K random points as cluster centers (means)
  - Alternate:
    - Assign data instances to closest mean
    - Assign each mean to the average of its assigned points
  - Stop when no points’ assignments change
K-Means Example
K-Means as Optimization

- Consider the total distance to the means:
  \[ \phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i}) \]

- Each iteration reduces phi

- Two stages each iteration:
  - Update assignments: fix means c, change assignments a
  - Update means: fix assignments a, change means c
Phase I: Update Assignments

- For each point, re-assign to closest mean:

\[ a_i = \arg\min_k \text{dist}(x_i, c_k) \]

- Can only decrease total distance phi!

\[ \phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i}) \]
Phase II: Update Means

- Move each mean to the average of its assigned points:
  \[ c_k = \frac{1}{|\{i : a_i = k\}|} \sum_{i : a_i = k} x_i \]

- Also can only decrease total distance... (Why?)

- Fun fact: the point \( y \) with minimum squared Euclidean distance to a set of points \( \{x\} \) is their mean
Initialization

- K-means is non-deterministic
  - Requires initial means
  - It does matter what you pick!
  - What can go wrong?

- Various schemes for preventing this kind of thing: variance-based split / merge, initialization heuristics
A local optimum:

Why doesn’t this work out like the earlier example, with the purple taking over half the blue?
K-Means Questions

- Will K-means converge?
  - To a global optimum?

- Will it always find the true patterns in the data?
  - If the patterns are very very clear?

- Will it find something interesting?

- Do people ever use it?

- How many clusters to pick?
Agglomerative Clustering
Agglomerative Clustering

- **Agglomerative clustering:**
  - First merge very similar instances
  - Incrementally build larger clusters out of smaller clusters

- **Algorithm:**
  - Maintain a set of clusters
  - Initially, each instance in its own cluster
  - Repeat:
    - Pick the two closest clusters
    - Merge them into a new cluster
    - Stop when there’s only one cluster left

- Produces not one clustering, but a family of clusterings represented by a **dendrogram**
Agglomerative Clustering

- How should we define “closest” for clusters with multiple elements?

- Many options
  - Closest pair (single-link clustering)
  - Farthest pair (complete-link clustering)
  - Average of all pairs
  - Ward’s method (min variance, like k-means)

- Different choices create different clustering behaviors