Announcements

• HW7: due **Today, Apr 2, 11:59 PM PT**

• Releasing later this week:
  • HW8: due **Tuesday, April 9, 11:59 PM PT**
  • Project 5: due **Tuesday, April 16, 11:59 PM PT**

• Releasing next week:
  • HW9: due **Tuesday, April 16, 11:59 PM PT**

• TA 1-1s: see announcement on Ed

Pre-scan attendance QR code now!
Recap: Training and Machine Learning

- Big idea: ML algorithms learn patterns between features and labels from data
  - You don’t have to reason about the data yourself
  - You’re given training data: lots of example datapoints and their actual labels

Training: Learn patterns from labeled data, and periodically test how well you’re doing

Eventually, use your algorithm to predict labels for unlabeled data
Classification: Ham vs. Spam Emails

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. ...

TO BE REMOVED FROM FUTURE MAILINGS, SIMPLY REPLY TO THIS MESSAGE AND PUT "REMOVE" IN THE SUBJECT.

99 MILLION EMAIL ADDRESSES FOR ONLY $99

Ok, I know this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.
Classification: 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
- ...
- Features are increasingly induced rather than crafted
Recap: Naïve Bayes Model

- **Random variables in this Bayes net:**
  - $Y = \text{The label}$
  - $F_1, F_2, ..., F_n = \text{The n features}$

- **Probability tables in this Bayes net:**
  - $P(Y) = \text{Probability of each label occurring, given no information about the features. Sometimes called the prior.}$
  - $P(F_i|Y) = \text{One table per feature. Probability distribution over a feature, given the label.}$
Recap: Naïve Bayes Model

- **To perform training:**
  - Use the training dataset to estimate the probability tables.
  - Estimate $P(Y) = \text{how often does each label occur?}$
  - Estimate $P(F_i|Y) = \text{how does the label affect the feature?}$

- **To perform classification:**
  - Instantiate all features. You know the input features, so they’re your evidence.
  - Query for $P(Y|f_1, f_2, ..., f_n)$. Probability of label, given all the input features.
    Use an inference algorithm (e.g. variable elimination) to compute this.
Recap: Naïve Bayes for Spam Filter

- Step 1: Select a ML algorithm. We choose to model the problem with Naïve Bayes.
- Step 2: Choose features to use.

<table>
<thead>
<tr>
<th>Y: The label (spam or ham)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>ham</td>
</tr>
<tr>
<td>spam</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F_1: A feature (do I know the sender?)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_1</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>yes</td>
</tr>
<tr>
<td>no</td>
</tr>
<tr>
<td>yes</td>
</tr>
<tr>
<td>no</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F_2: Another feature (# of occurrences of FREE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_2</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
Example: Overfitting

- Postiors determined by *relative* probabilities (odds ratios):

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

| south-west : inf | screens : inf |
| nation : inf | minute : inf |
| morally : inf | guaranteed : inf |
| nicely : inf | $205.00 : inf |
| extent : inf | delivery : inf |
| seriously : inf | signature : inf |
| ... | ... |

What went wrong here?
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!!
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 cs281a)

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

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

- Laplace’s estimate (extended):
  - Pretend you saw every outcome k extra times
    \[ P_{LAP,k}(x) = \frac{c(x) + k}{N + k|X|} \]
  - What’s Laplace with k = 0?
  - k is the strength of the prior

- Laplace for conditionals:
  - Smooth each condition independently:
    \[ P_{LAP,k}(x|y) = \frac{c(x, y) + k}{c(y) + k|X|} \]
Real Naïve Bayes: Smoothing

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

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

|          | P(W|ham) | P(W|spam) |
|----------|----------|-----------|
| helvetica| 11.4     |           |
| seems    | 10.8     |           |
| group    | 10.2     |           |
| ago      | 8.4      |           |
| areas    | 8.3      |           |
|          |          |           |
|          | 28.8     |           |
| verdana  | 28.8     |           |
| Credit   | 28.4     |           |
| ORDER    | 27.2     |           |
| <FONT>   | 26.9     |           |
| money    | 26.5     |           |
|          |          |           |
|          |          | 26.5      |

*Do these make more sense?*
Tuning
Training and Testing

- Apple = Car

Practice Exam

Final Exam!
**Important Concepts**

- **How do we check that we’re not overfitting during training?**
- **Split training data into 3 different sets:**
  - Training set
  - Held out set (more on this later)
  - Test set

- **Experimentation cycle**
  - Learn parameters (e.g. model probabilities) on training set
  - Compute accuracy of test set
  - Very important: never “peek” at the test set!

- **Evaluation (many metrics possible, e.g. accuracy)**
  - 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
Tuning on Held-Out Data

- 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
Linear Classifiers
Hello,
Do you want free printr cartriges? Why pay more when you can get them ABSOLUTELY FREE! Just

\[
\begin{align*}
\text{# free} & : 2 \\
\text{YOUR NAME} & : 0 \\
\text{MISSPELLED} & : 2 \\
\text{FROM FRIEND} & : 0 \\
\ldots & \\
\end{align*}
\]

SPAM or

\[
\begin{align*}
\text{PIXEL-7,12} & : 1 \\
\text{PIXEL-7,13} & : 0 \\
\ldots & \\
\text{NUM_LOOPS} & : 1 \\
\ldots & \\
\end{align*}
\]

“2”
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

Dot product \( w \cdot f \) positive means the positive class (spam)

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

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

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

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

Do these weights make sense for spam classification?
A tuple like (2,3) can be interpreted two different ways:

- A point on a coordinate grid

- A vector in space. Notice we are not on a coordinate grid.

A tuple with more elements like (2, 7, -3, 6) is a point or vector in higher-dimensional space (hard to visualize)
Definition of dot product:
- \( a \cdot b = \sum_i a_i b_i = |a| |b| \cos(\theta) \)
- \( \theta \) is the angle between the vectors \( a \) and \( b \)

Consequences of this definition:
- Vectors closer together = “similar” vectors
  = smaller angle \( \theta \) between vectors
  = larger (more positive) dot product
- If \( \theta < 90^\circ \), then dot product is positive
- If \( \theta = 90^\circ \), then dot product is zero
- If \( \theta > 90^\circ \), then dot product is negative
Weights

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

\[
\begin{align*}
\text{# free} & : 4 \\
\text{YOUR NAME} & : -1 \\
\text{MISSPELLED} & : 1 \\
\text{FROM_FRIEND} & : -3 \\
\ldots & \\
\end{align*}
\begin{align*}
\text{# free} & : 2 \\
\text{YOUR NAME} & : 0 \\
\text{MISSPELLED} & : 0 \\
\text{FROM_FRIEND} & : 2 \\
\ldots & \\
\end{align*}
\]

\[ w \cdot f(x_1) \]

\[ w \cdot f(x_2) \]

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 (divides space into two sides)
  - One side corresponds to $Y=+1$, the other corresponds to $Y=-1$
- In the example:
  - $f \cdot w > 0$ when $4*\text{free} + 2*\text{money} > 0$
  - $f \cdot w < 0$ when $4*\text{free} + 2*\text{money} < 0$
    - These equations correspond to two halves of the feature space
  - $f \cdot w = 0$ when $4*\text{free} + 2*\text{money} = 0$
    - This equation corresponds to the decision boundary (a line in 2D, a hyperplane in higher dimensions)
Weight Updates

$\omega_n$
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.
    \[ w = w + y^* \cdot f \]
Learning: Binary Perceptron

- **Misclassification, Case I:**
  - $w \cdot f > 0$, so we predict $+1$
  - True class is $-1$
  - We want to modify $w$ to $w'$ such that dot product $w' \cdot f$ is lower
  - **Update if we misclassify a true class $-1$ sample:** $w' = w - f$
    - Proof: $w' \cdot f = (w - f) \cdot f = (w \cdot f) - (f \cdot f) = (w \cdot f) - |f|^2$
      - Note that $|f|^2$ is always positive

- **Misclassification, Case II:**
  - $w \cdot f < 0$, so we predict $-1$
  - True class is $+1$
  - We want to modify $w$ to $w'$ such that dot product $w' \cdot f$ is higher
  - **Update if we misclassify a true class $+1$ sample:** $w' = w + f$
    - Proof: $w' \cdot f = (w + f) \cdot f = (w \cdot f) + (f \cdot f) = (w \cdot f) + |f|^2$
      - Note that $|f|^2$ is always positive

- **Write update compactly as** $w' = w + y^* \cdot f$, where $y^*$ = true class
Examples: Perceptron

- Separable Case
Examples: Perceptron

- Separable Case
Examples: Perceptron

- Separable Case
Examples: Perceptron

- Separable Case
Examples: Perceptron

- Separable Case
Examples: Perceptron

- Separable Case
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:

  \[ \text{mistakes} < \frac{k}{\delta^2} \]
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
Improving the Perceptron
Even the best linear boundary makes at least one mistake
Non-Separable Case: Probabilistic Decision
How to get deterministic decisions?

- Perceptron scoring: \( z = w \cdot f(x) \)
- If \( z = w \cdot f(x) \) positive \( \rightarrow \) classifier says: 1.0 probability this is class +1
- If \( z = w \cdot f(x) \) negative \( \rightarrow \) classifier says: 0.0 probability this is class +1

- Step function

\[
H(z) = \begin{cases} 
1 & z > 0 \\
0 & z \leq 0 
\end{cases}
\]

- \( z = \) output of perceptron
  
  \( H(z) = \) probability the class is +1, according to the classifier
How to get probabilistic decisions?

- **Perceptron scoring**: $z = w \cdot f(x)$

- If $z = w \cdot f(x)$ very positive $\rightarrow$ probability of class +1 should approach 1.0

- If $z = w \cdot f(x)$ very negative $\rightarrow$ probability of class +1 should approach 0.0

- **Sigmoid function**

  $$\phi(z) = \frac{1}{1 + e^{-z}}$$

- $z$ = output of perceptron

  $\phi(z)$ = probability the class is +1, according to the classifier
A 1D Example

\[ P(\text{red}|x) = \frac{1}{1 + e^{-wx}} \]

where \( w \) is some weight constant (1D vector) we have to learn (assume \( w \) is positive in this example)
Recall maximum likelihood estimation: Choose the $w$ value that maximizes the probability of the observed (training) data

$$\text{Likelihood} = P(\text{training data}|w)$$

$$= \prod_i P(\text{training datapoint } i \mid w)$$

$$= \prod_i P(\text{point } x^{(i)} \text{ has label } y^{(i)}|w)$$

$$= \prod_i P(y^{(i)}|x^{(i)}; w)$$

$$\text{Log Likelihood} = \sum_i \log P(y^{(i)}|x^{(i)}; w)$$
Recall maximum likelihood estimation: Choose the $w$ value that maximizes the probability of the observed (training) data

$$P(\text{point } x^{(i)} \text{ has label } y^{(i)} = +1 \mid w)$$

$$= P(y^{(i)} = +1 \mid x^{(i)}; w)$$

$$= \frac{1}{1 + e^{-w \cdot x^{(i)}}}$$

$$P(\text{point } x^{(i)} \text{ has label } y^{(i)} = -1 \mid w)$$

$$= P(y^{(i)} = -1 \mid x^{(i)}; w)$$

$$= 1 - \frac{1}{1 + e^{-w \cdot x^{(i)}}}$$
Best w?

- **Maximum likelihood estimation:**

\[
\max_w \ ll(w) = \max_w \ \sum_i \log P(y^{(i)}|x^{(i)}; w)
\]

with:

\[
P(y^{(i)} = +1|x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}
\]

\[
P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}
\]

= Logistic Regression
Separable Case: Deterministic Decision – Many Options
Separable Case: Probabilistic Decision – Clear Preference
Multiclass Logistic Regression

- **Recall Perceptron:**
  - A weight vector for each class: $w_y$
  - Score (activation) of a class $y$: $w_y \cdot f(x)$
  - Prediction highest score wins: $y = \arg \max_y w_y \cdot f(x)$

- **How to make the scores into probabilities?**

\[
\begin{align*}
z_1, z_2, z_3 &\rightarrow \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \quad \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \quad \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}
\end{align*}
\]

original activations

softmax activations
Recall maximum likelihood estimation: Choose the $w$ value that maximizes the probability of the observed (training) data

\[
\text{Likelihood} = P(\text{training data}|w) \\
= \prod_{i} P(\text{training datapoint } i \mid w) \\
= \prod_{i} P(\text{point } x^{(i)} \text{ has label } y^{(i)}|w) \\
= \prod_{i} P(y^{(i)}|x^{(i)}; w) \\
\text{Log Likelihood} = \sum_{i} \log P(y^{(i)}|x^{(i)}; w)
\]
Best w?

- Maximum likelihood estimation:

\[
\max_w \ ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)
\]

with:

\[
P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y(i)} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}
\]

= Multi-Class Logistic Regression
Softmax with Different Bases

\[ P(\text{red} | x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}} \]

looks like \( \max_y w_y \cdot x \)
Recall: Binary perceptron is a special case of multi-class perceptron

- Multi-class: Compute $w_y \cdot f(x)$ for each class $y$, pick class with the highest activation
- Binary case:
  - Let the weight vector of +1 be $w$ (which we learn).
  - Let the weight vector of -1 always be 0 (constant).
- Binary classification as a multi-class problem:
  - Activation of negative class is always 0.
  - If $w \cdot f$ is positive, then activation of +1 ($w \cdot f$) is higher than -1 (0).
  - If $w \cdot f$ is negative, then activation of -1 (0) is higher than +1 ($w \cdot f$).

\[
P(\text{red}|x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}}
\]

with $w_{\text{red}} = 0$ becomes:

\[
P(\text{red}|x) = \frac{1}{1 + e^{-wx}}
\]
Optimization

i.e., how do we solve:

$$\max_w ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)$$