Last Time

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

- Naïve Bayes

\[ P(Y|F_0,0 \ldots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y) \]

- Parameter estimation:
  - MLE, MAP, priors \[ P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}} \]
  - Laplace smoothing \[ P_{\text{LAP,k}}(x) = \frac{c(x) + k}{N + k|X|} \]

- Training set, held-out set, test set
Phase 1: Train model on Training Data. Choice points for “tuning”
- Attributes / Features
- Model types: Naïve Bayes vs. Perceptron vs. Logistic Regression vs. Neural Net etc..
- Model hyperparameters
  - E.g. Naïve Bayes – Laplace k
  - E.g. Logistic Regression – weight regularization
  - E.g. Neural Net – architecture, learning rate, …
- Make sure good performance on training data (why?)

Phase 2: Evaluate on Hold-Out Data
- If Hold-Out performance is close to Train performance
  - We achieved good generalization, onto Phase 3! 😊
- If Hold-Out performance is much worse than Train performance
  - We overfitted to the training data! 😞
  - Take inspiration from the errors and:
    - Either: go back to Phase 1 for tuning (typically: make the model less expressive)
    - Or: if we are out of options for tuning while maintaining high train accuracy, collect more data
      (i.e., let the data drive generalization, rather than the tuning/regularization) and go to Phase 1

Phase 3: Report performance on Test Data

Possible outer-loop: Collect more data 😊
Practical Tip: Baselines

- 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
Linear Classifiers
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

\[ w \cdot f(x) \] 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 $Y=-1$

$$w$$

<table>
<thead>
<tr>
<th>BIAS</th>
<th>-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>free</td>
<td>4</td>
</tr>
<tr>
<td>money</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
In the space of feature vectors
- Examples are points
- Any weight vector is a hyperplane
- One side corresponds to $Y=+1$
- Other corresponds to $Y=-1$

$w$

<table>
<thead>
<tr>
<th>free</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>money</td>
<td>2</td>
</tr>
</tbody>
</table>
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 $Y=-1$

$$w$$

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

$$f \cdot w = 0$$

+1 = SPAM

-1 = HAM
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.
    \[ w = w + y^* \cdot f \]

Before: \( w_f \)
After: \( w_f + y^*f_f \)
\( ff \geq 0 \)
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”  [1 1 0 1 1]
“win the election”  [1 1 0 0 1]
“win the game”  [1 1 0 1]

<table>
<thead>
<tr>
<th>( w_{SPORTS} )</th>
<th>1</th>
<th>-2</th>
<th>-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIAS</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>win</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>game</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>vote</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>the</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( w_{POLITICS} )</th>
<th>0</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIAS</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>win</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>game</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>vote</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( w_{TECH} )</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIAS</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>win</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>game</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>vote</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>the</td>
<td>0</td>
<td></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
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
Non-Separable Case: Deterministic Decision

Even the best linear boundary makes at least one mistake.
Non-Separable Case: Probabilistic Decision
How to get probabilistic decisions?

- Perceptron scoring: \( z = w \cdot f(x) \)
- If \( z = w \cdot f(x) \) very positive \( \rightarrow \) want probability going to 1
- If \( z = w \cdot f(x) \) very negative \( \rightarrow \) want probability going to 0

- Sigmoid function

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

\[\begin{align*}
\phi(z) &= \frac{1}{1 + e^{-z}} \\
&= \frac{1}{1 + e^{-z}}
\end{align*}\]
A 1D Example

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

- definitely blue
- not sure
- definitely red

Probability increases exponentially as we move away from boundary.

(normalizer)
The Soft Max

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

This formula looks like \( \max_y w_y \cdot x \).
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?**

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

original activations

softmax activations
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
Optimization

i.e., how do we solve:

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