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. graphs)
  - Learning hidden concepts (e.g. clustering)

- First: model-based classification
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, WidelyBroadcast
- ...

---

**Example Email:**

Dear Sir.

First, I must solicit your confidence in this transaction, this is by virtue of its nature as being utterly confidential 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.
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
  - ...

0
1
2
??
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!
Linear Classifiers
A 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

---

Dear Sir.
First, I must solicit your confidence in this transaction, this is by virtue of its nature as being utterly confidential 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.
Feature Vectors

Hello,
Do you want free printer cartridges? Why pay more when you can get them ABSOLUTELY FREE! Just

Hello,  
Do you want free printr cartridges? Why pay more when you can get them ABSOLUTELY FREE! Just

SPAM or +

"2"

Hello,  
Do you want free printr cartridges? Why pay more when you can get them ABSOLUTELY FREE! Just

SPAM or +

"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

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

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 $Y=-1$

$$w$$

<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>
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>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 $Y=-1$

<table>
<thead>
<tr>
<th>Feature</th>
<th>Weight</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>
How do we learn weight?
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
  \[ y = \text{arg max}_y \ w_y \cdot f(x) \]
- 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” \[ [1 \ 1 \ 1 \ 0 \ 1] \]

\[ \begin{array}{ccc}
\text{BIAS} & 1 & 0 \\
\text{win} & 0 & -1 \\
\text{game} & 0 & 0 \\
\text{vote} & 0 & -1 \\
\text{the} & 0 & -1 \\
\end{array} \]

\[ \begin{array}{ccc}
\text{BIAS} & 0 & 1 \\
\text{win} & 0 & 1 \\
\text{game} & 0 & 0 \\
\text{vote} & 0 & 1 \\
\text{the} & 0 & 1 \\
\end{array} \]

\[ \begin{array}{ccc}
\text{BIAS} & 0 & 0 \\
\text{win} & 0 & 0 \\
\text{game} & 0 & 0 \\
\text{vote} & 0 & 0 \\
\text{the} & 0 & 0 \\
\end{array} \]
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)
- **Non-separable?**
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
Training and Testing
Underfitting and Overfitting
Overfitting

- Too many features
  - Spam if contains “FREE!”
  - Spam if contains $dd, CAPS
  - ...
  - Spam if contains “Sir”
  - Spam if contains address
  - Spam if contains “OT”
  - ...

---

Dear Sir.

First, I must solicit your confidence in this transaction, this is by virtue of its nature as being utterly confidential 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.
Example: Overfitting

\[ P(\text{features, } C = 2) \]

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

\[ P(\text{features, } C = 3) \]

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

2 wins!!
Overfitting
Overfitting

Degree 15 polynomial
Unseen Events
Generalization and Overfitting

- Relative frequency parameters will overfit the training data!
  - Just because we never saw a non-spam email with an address 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
Regularization

- Limit the number of features
- Limit the norm of the vector \( \mathbf{w} \)
  - If \( \mathbf{w}_1 \) and \( \mathbf{w}_2 \) are equally good, and \( |\mathbf{w}_1| > |\mathbf{w}_2| \), then \( \mathbf{w}_2 \) is likely to better generalize

\[
y = \arg \max_y \left( w_y \cdot f(x) - |w_y| \right)
\]
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 on 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
  - Underfitting: fits the training set poorly
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
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
Improving the Perceptron: Next lecture!