CS 546
Machine Learning in NLP

Review 1: Supervised Learning, Binary Classifiers

Dan Roth
Department of Computer Science
University of Illinois at Urbana-Champaign

Augmented and modified by Vivek Srikumar
Announcements

• Registration

• Piazza

• Web site
  – Topics
  – Papers

• Information forms will be available
  – Essential for group formation
Supervised learning, Binary classification

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Learning as optimization
5. Support vector machines
6. Logistic Regression
Where are we?

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Learning as optimization
5. Support vector machines
6. Logistic Regression
Supervised learning: General setting

• Given: Training examples of the form $\langle x, f(x) \rangle$
  – The function $f$ is an unknown function
• The input $x$ is represented in a feature space
  – Typically $x \in \{0,1\}^n$ or $x \in \mathbb{R}^n$
• For a training example $x$, $f(x)$ is called the label
• Goal: Find a good approximation for $f$
• The label
  – Binary classification: $f(x) \in \{-1,1\}$
  – Multiclass classification: $f(x) \in \{1, 2, 3, \ldots, K\}$
  – Regression: $f(x) \in \mathbb{R}$
Nature of applications

• There is no human expert
  – Eg: Identify DNA binding sites

• Humans can perform a task, but can’t describe how they do it
  – Eg: Object detection in images

• The desired function is hard to obtain in closed form
  – Eg: Stock market
Binary classification

• Spam filtering
  – Is an email spam or not?

• Recommendation systems
  – Given user’s movie preferences, will she like a new movie?

• Malware detection
  – Is an Android app malicious?

• Time series prediction
  – Will the future value of a stock increase or decrease with respect to its current value?
The fundamental problem: Machine learning is ill-posed!

**x_1** **x_2** **x_3** **x_4**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Can you learn this function?

What is it?
Is learning possible at all?

- There are $2^{16} = 65536$ possible Boolean functions over 4 inputs.
  - Why? There are 16 possible outputs. Each way to fill these 16 slots is a different function, giving $2^{16}$ functions.
- We have seen only 7 outputs.
- We cannot know what the rest are without seeing them.
  - Think of an adversary filling in the labels every time you make a guess at the function.
Solution: Restricted hypothesis space

- A *hypothesis space* is the set of possible functions we consider
  - We were looking at the space of all Boolean functions

- Instead choose a hypothesis space that is smaller than the space of all functions
  - Only simple conjunctions (with four variables, there are only 16 conjunctions without negations)
  - m-of-n rules: Pick a set of n variables. At least m of them must be true
  - Linear functions

- How do we pick a hypothesis space?
  - Using some prior knowledge (or by guessing)

- What if the hypothesis space is so small that nothing in it agrees with the data?
  - We need a hypothesis space that is flexible enough
Where are we?

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Learning as optimization
5. Support vector machines
6. Logistic Regression
Linear Classifiers

• Input is a n dimensional vector $\mathbf{x}$
• Output is a label $y \in \{-1, 1\}$

• Linear threshold units classify an example $\mathbf{x}$ as

• Classification rule: $\text{sgn}(b + \mathbf{w}^T \mathbf{x}) = \text{sgn}(b + \sum w_i x_i)$
  – $b + \mathbf{w}^T \mathbf{x} \geq 0 \Rightarrow$ Predict $y = 1$
  – $b + \mathbf{w}^T \mathbf{x} < 0 \Rightarrow$ Predict $y = -1$
Linear Classifiers

• Input is a n dimensional vector \( x \)
• Output is a label \( y \in \{-1, 1\} \)  

• Linear threshold units classify an example \( x \) as

• Classification rule= \( \text{sgn}(b + w^T x) = \text{sgn}(b + \sum w_i x_i) \)
  
  - \( b + w^T x \geq 0 \) \( \Rightarrow \) Predict \( y = 1 \)
  
  - \( b + w^T x < 0 \) \( \Rightarrow \) Predict \( y = -1 \)

For now

Called the activation
The geometry of a linear classifier

\[ \text{sgn}(b + w_1 x_1 + w_2 x_2) \]

In n dimensions, a linear classifier represents a hyperplane that separates the space into two half-spaces.

We only care about the sign, not the magnitude.

\[ b + w_1 x_1 + w_2 x_2 = 0 \]
Linear classifiers are an expressive hypothesis class

• Many functions are linear
  – Conjunctions
    \[ y = x_1 \land x_2 \land x_3 \quad y = \text{sgn}(-3 + x_1 + x_2 + x_3) \quad W = (1,1,1); b = -3 \]
  – At least m-of-n functions

• We will see later in the class that many structured predictors are linear functions too

• Often a good guess for a hypothesis space

• Some functions are not linear
  – The XOR function
  – Non-trivial Boolean functions
XOR is not linearly separable

No line can be drawn to separate the two classes
Even these functions can be *made* linear

These points are not separable in 1-dimension by a line

What is a one-dimensional line, by the way?

The trick: Change the representation
Even these functions can be *made* linear

The trick: Use feature *conjunctions*

Transform points: Represent each point $x$ in 2 dimensions by $(x, x^2)$

Now the data is linearly separable in this space!

*Exercise*: How do you apply this for the XOR case?
Almost linearly separable data

\[ \text{sgn}(b + w_1 x_1 + w_2 x_2) \]

Training data is almost separable, except for some noise.

How much noise do we allow for?
Training a linear classifier

Three cases to consider:

1. Training data is linearly separable
   - Simple conjunctions, etc

2. Training data is linearly inseparable
   - XOR, etc

3. Training data is almost linearly separable
   - Noise in the data
   - We could allow some mistakes on the training data to account for these
Simplifying notation

We can stop writing $b$ at each step because of the following notational sugar:

- The prediction function is $\text{sgn}(b + \mathbf{w}^T\mathbf{x})$
- Rewrite $\mathbf{x}$ as $[1, \mathbf{x}] \rightarrow \mathbf{x}'$
- Rewrite $\mathbf{w}$ as $[b, \mathbf{w}] \rightarrow \mathbf{w}'$
  - increases dimensionality by one
- But we can write the prediction as $\text{sgn}(\mathbf{w}'^T\mathbf{x}')$

We will not show $b$, and instead fold the bias term into the input by adding an extra feature

But remember that it is there
Where are we?

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Learning as optimization
5. Support vector machines
6. Logistic Regression
The Perceptron algorithm

• Rosenblatt 1958

• The goal is to find a separating hyperplane
  – For separable data, guaranteed to find one

• An online algorithm
  – Processes one example at a time

• Several variants exist (will discuss briefly at towards the end)
The algorithm

Given a training set \( D = \{(x,y)\}, x \in \mathbb{R}^n, y \in \{-1,1\} \)

1. Initialize \( w = 0 \in \mathbb{R}^n \)

2. For epoch = 1 … T:
   1. For each training example \((x, y)\) \(\in D\):
      1. Predict \( y' = \text{sgn}(w^T x) \)
      2. If \( y \neq y' \), update \( w \gets w + y x \)

3. Return \( w \)

**Prediction:** \( \text{sgn}(w^T x) \)
Given a training set $D = \{(x,y)\}$

1. Initialize $w = 0 \in \mathbb{R}^n$
2. For epoch = 1 ... $T$:
   1. For each training example $(x, y) \in D$:
      1. Predict $y' = \text{sgn}(w^T x)$
      2. If $y \neq y'$, update $w \leftarrow w + y x$
3. Return $w$

**Prediction:** $\text{sgn}(w^T x)$
Geometry of the perceptron update

Exercise: Verify for yourself that a similar picture can be drawn for negative examples too.
Convergence

• Convergence theorem
  – If there exist a set of weights that are consistent with the data (i.e. the data is linearly separable), the perceptron algorithm will converge.
  – How long?

• Cycling theorem
  – If the training data is not linearly separable, then the learning algorithm will eventually repeat the same set of weights and enter an infinite loop
Margin

- The **margin** of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.
Margin

- The **margin** of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.
- The **margin of a data set** ($\gamma$) is the maximum margin possible for that dataset using any weight vector.
The mistake bound theorem [Novikoff 1962]

Let \( D = \{(x_i, y_i)\} \) be a labeled dataset that is separable.
Let \( \|x_i\| \leq R \) for all examples.
Let \( \gamma \) be the margin of the dataset \( D \).
Then, the perceptron algorithm will make at most \( \frac{R^2}{\gamma^2} \) mistakes on the data.

**Proof idea:** We know that there is some true weight vector \( \mathbf{w}^* \). Each perceptron update reduces the angle between \( \mathbf{w} \) and \( \mathbf{w}^* \).

**Note:** The theorem doesn’t depend on the number of examples we have.
Beyond the separable case

• The good news
  – Perceptron makes no assumption about data distribution
  – Even adversarial
  – After a fixed number of mistakes, you are done. Don’t even need to see any more data

• The bad news: Real world is not linearly separable
  – Can’t expect to *never* make mistakes again
  – What can we do: more features, try to be linearly separable if you can
Variants of the algorithm

• So far: We return the final weight vector
• Voted perceptron
  – Remember every weight vector in your sequence of updates.
  – At final prediction time, each weight vector gets to vote on the label. The number of votes it gets is the number of iterations it survived before being updated
  – Comes with strong theoretical guarantees about generalization, impractical because of storage issues
• Averaged perceptron
  – Instead of using all weight vectors, use the average weight vector (i.e longer surviving weight vectors get more say)
  – More practical alternative and widely used
Next steps

• What is the perceptron doing?
  – Error-bound exists, but over training data
  – Is it minimizing a loss function? Can we say something about future errors?

• More on different loss functions
• Support vector machine
  – A look at regularization
• Logistic Regression
Announcements

• Registration information
  – Class size has increased. You should be able to register

• The newsgroup for the class is now active
  – Google groups: cs6961-fall14

• Please return the information forms if you have filled them out

• Please go over the material on logistic regression and at least one of the papers on SVM