Regularization
+
Perceptron

Perceptron Readings:
Murphy 8.5.4
Bishop 4.1.7
HTF –
Mitchell 4.4.0

Matt Gormley
Lecture 10
February 20, 2016
Reminders

• Homework 3: Linear / Logistic Regression
  – Release: Mon, Feb. 13
  – Due: Wed, Feb. 22 at 11:59pm

• Homework 4: Perceptron / Kernels / SVM
  – Due: Wed, Mar. 01 at 11:59pm

• Midterm Exam (Evening Exam)
  – Tue, Mar. 07 at 7:00pm – 9:30pm
  – See Piazza for details about location
Outline

• Regularization
  – Motivation: Overfitting
  – L2, L1, L0 Regularization
  – Relation between Regularization and MAP Estimation

• Perceptron
  – Online Learning
  – Margin Definitions
  – Perceptron Algorithm
  – Perceptron Mistake Bound

• Generative vs. Discriminative Classifiers
REGULARIZATION
Overfitting

**Definition:** The problem of **overfitting** is when the model captures the noise in the training data instead of the underlying structure.

Overfitting can occur in all the models we’ve seen so far:

- KNN (e.g. when k is small)
- Naïve Bayes (e.g. without a prior)
- Linear Regression (e.g. with basis function)
- Logistic Regression (e.g. with many rare features)
Motivation: Regularization

Example: Stock Prices

• Suppose we wish to predict Google’s stock price at time $t+1$

• **What features should we use?**
  (putting all computational concerns aside)
  – Stock prices of all other stocks at times $t, t-1, t-2, \ldots, t-k$
  – Mentions of Google with positive / negative sentiment words in all newspapers and social media outlets

• Do we believe that **all** of these features are going to be useful?
Motivation: Regularization

• **Occam’s Razor:** prefer the simplest hypothesis

• What does it mean for a hypothesis (or model) to be simple?
  1. small number of features (**model selection**)
  2. small number of “important” features (**shrinkage**)
Regularization

Whiteboard

– L2, L1, L0 Regularization
– Example: Linear Regression
– Probabilistic Interpretation of Regularization
Don’t Regularize the Bias (Intercept) Parameter!

- In our models so far, the bias / intercept parameter is usually denoted by \( \theta_0 \) -- that is, the parameter for which we fixed \( x_0 = 1 \).
- Regularizers always avoid penalizing this bias / intercept parameter.
- Why? Because otherwise the learning algorithms wouldn’t be invariant to a shift in the y-values.

Whitening Data

- It’s common to *whiten* each feature by subtracting its mean and dividing by its variance.
- For regularization, this helps all the features be penalized in the same units (e.g. convert both centimeters and kilometers to z-scores).
Regularization:

\[ \ln \lambda = +.18 \]
## Polynomial Coefficients

<table>
<thead>
<tr>
<th></th>
<th>none</th>
<th>exp(18)</th>
<th>huge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_0^*$</td>
<td>0.35</td>
<td>0.35</td>
<td>0.13</td>
</tr>
<tr>
<td>$w_1^*$</td>
<td>232.37</td>
<td>4.74</td>
<td>-0.05</td>
</tr>
<tr>
<td>$w_2^*$</td>
<td>-5321.83</td>
<td>-0.77</td>
<td>-0.06</td>
</tr>
<tr>
<td>$w_3^*$</td>
<td>48568.31</td>
<td>-31.97</td>
<td>-0.05</td>
</tr>
<tr>
<td>$w_4^*$</td>
<td>-231639.30</td>
<td>-3.89</td>
<td>-0.03</td>
</tr>
<tr>
<td>$w_5^*$</td>
<td>640042.26</td>
<td>55.28</td>
<td>-0.02</td>
</tr>
<tr>
<td>$w_6^*$</td>
<td>-1061800.52</td>
<td>41.32</td>
<td>-0.01</td>
</tr>
<tr>
<td>$w_7^*$</td>
<td>1042400.18</td>
<td>-45.95</td>
<td>-0.00</td>
</tr>
<tr>
<td>$w_8^*$</td>
<td>-557682.99</td>
<td>-91.53</td>
<td>0.00</td>
</tr>
<tr>
<td>$w_9^*$</td>
<td>125201.43</td>
<td>72.68</td>
<td>0.01</td>
</tr>
</tbody>
</table>
Over Regularization:

Slide courtesy of William Cohen
Regularization Exercise

*In-class Exercise*

1. Plot train error vs. # features (cartoon)
2. Plot test error vs. # features (cartoon)
Example: Logistic Regression

Training Data
Example: Logistic Regression

Test Data
Example: Logistic Regression

![Graph showing error vs 1/lambda for training and testing datasets.]

- **Error** on the y-axis.
- **1/lambda** on the x-axis.
- Red line represents the training dataset.
- Blue line represents the testing dataset.
Example: Logistic Regression

Classification with Logistic Regression (\(\lambda=1e-05\))
Example: Logistic Regression
Example: Logistic Regression

Classification with Logistic Regression (\(\lambda=0.001\))
Example: Logistic Regression

Classification with Logistic Regression (\(\lambda=0.01\))
Example: Logistic Regression
Example: Logistic Regression

Classification with Logistic Regression (\lambda = 1)
Example: Logistic Regression
Example: Logistic Regression
Example: Logistic Regression

Classification with Logistic Regression (lambda=1000)
Example: Logistic Regression

Classification with Logistic Regression (\(\lambda = 10000\))
Example: Logistic Regression

Classification with Logistic Regression (λ=100000)
Example: Logistic Regression

Classification with Logistic Regression (lambda=1e+06)
Example: Logistic Regression

Classification with Logistic Regression ($\lambda = 1e+07$)
Example: Logistic Regression
Takeaways

1. **Nonlinear basis functions** allow **linear models** (e.g. Linear Regression, Logistic Regression) to capture **nonlinear** aspects of the original input.

2. Nonlinear features are **require no changes to the model** (i.e. just preprocessing).

3. **Regularization** helps to avoid **overfitting**.

4. **Regularization** and **MAP estimation** are equivalent for appropriately chosen priors.
THE PERCEPTRON ALGORITHM
Why don’t we drop the generative model and try to learn this hyperplane directly?
Background: Hyperplanes

Hyperplane (Definition 1):
\[ \mathcal{H} = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} = b \} \]

Hyperplane (Definition 2):
\[ \mathcal{H} = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} = 0 \text{ and } x_0 = 1 \} \]

Half-spaces:
\[ \mathcal{H}^+ = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} > 0 \text{ and } x_0 = 1 \} \]
\[ \mathcal{H}^- = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} < 0 \text{ and } x_0 = 1 \} \]
Directly modeling the hyperplane would use a decision function:

\[ h(x) = \text{sign}(\theta^T x) \]

for:

\[ y \in \{-1, +1\} \]

Why don’t we drop the generative model and try to learn this hyperplane directly?
Online Learning

For $i = 1, 2, 3, \ldots$:

- **Receive** an unlabeled instance $x^{(i)}$
- **Predict** $y' = h(x^{(i)})$
- **Receive** true label $y^{(i)}$
  - **Check** for correctness ($y' = y^{(i)}$)

Goal:

- **Minimize** the number of mistakes
Online Learning: Motivation

Examples

1. Email classification (distribution of both spam and regular mail changes over time, but the target function stays fixed - last year's spam still looks like spam).
2. Recommendation systems. Recommending movies, etc.
3. Predicting whether a user will be interested in a new news article or not.
4. Ad placement in a new market.
Perceptron Algorithm

Data: Inputs are continuous vectors of length $K$. Outputs are discrete.

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots$$

where $x \in \mathbb{R}^K$ and $y \in \{+1, -1\}$

Prediction: Output determined by hyperplane.

$$\hat{y} = h_\theta(x) = \text{sign}(\theta^T x)$$

Learning: Iterative procedure:

• while not converged
  • receive next example $(x^{(i)}, y^{(i)})$
  • predict $y' = h(x^{(i)})$
  • if positive mistake: add $x^{(i)}$ to parameters
  • if negative mistake: subtract $x^{(i)}$ from parameters

$$\text{sign}(a) = \begin{cases} 1, & \text{if } a \geq 0 \\ -1, & \text{otherwise} \end{cases}$$
Perceptron Algorithm

Data: Inputs are continuous vectors of length K. Outputs are discrete. 
\[(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\]
where \(x \in \mathbb{R}^K\) and \(y \in \{+1, -1\}\)

Prediction: Output determined by hyperplane.
\[\hat{y} = h_\theta(x) = \text{sign}(\theta^T x)\]
\[\text{sign}(a) = \begin{cases} 1, & \text{if } a \geq 0 \\ -1, & \text{otherwise} \end{cases}\]

Learning:

Algorithm 1 Perceptron Learning Algorithm (Online)

1: procedure PERCEPTRON(\(D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\}\))
2: \(\theta \leftarrow 0\)  \(\triangleright\) Initialize parameters
3: for \(i \in \{1, 2, \ldots\}\) do
4: \(\hat{y} \leftarrow \text{sign}(\theta^T x^{(i)})\)  \(\triangleright\) Predict
5: if \(\hat{y} \neq y^{(i)}\) then
6: \(\theta \leftarrow \theta + y^{(i)}x^{(i)}\)  \(\triangleright\) If mistake, Update parameters
7: return \(\theta\)
Perceptron Algorithm: Example

Example: \((-1,2) - \times \)
\((1,0) + \checkmark \)
\((1,1) + \times \)
\((-1,0) - \checkmark \)
\((-1,-2) - \times \)
\((1,-1) + \checkmark \)

Algorithm:
- Set \(t=1\), start with all-zeroes weight vector \(w_1\).
- Given example \(x\), predict positive iff \(\theta_t \cdot x \geq 0\).
- On a mistake, update as follows:
  - Mistake on positive, update \(\theta_{t+1} \leftarrow \theta_t + x\)
  - Mistake on negative, update \(\theta_{t+1} \leftarrow \theta_t - x\)

\(\theta_1 = (0,0)\)
\(\theta_2 = \theta_1 - (-1,2) = (1,-2)\)
\(\theta_3 = \theta_2 + (1,1) = (2,-1)\)
\(\theta_4 = \theta_3 - (-1,-2) = (3,1)\)

Slide adapted from Nina Balcan
**Definition:** The margin of example $x$ w.r.t. a linear sep. $w$ is the distance from $x$ to the plane $w \cdot x = 0$ (or the negative if on wrong side).
**Geometric Margin**

**Definition:** The margin of example $x$ w.r.t. a linear separator $w$ is the distance from $x$ to the plane $w \cdot x = 0$ (or the negative if on wrong side).

**Definition:** The margin $\gamma_w$ of a set of examples $S$ w.r.t. a linear separator $w$ is the smallest margin over points $x \in S$. 

*Slide from Nina Balcan*
**Geometric Margin**

**Definition:** The margin of example $x$ w.r.t. a linear sep. $w$ is the distance from $x$ to the plane $w \cdot x = 0$ (or the negative if on wrong side).

**Definition:** The margin $\gamma_w$ of a set of examples $S$ wrt a linear separator $w$ is the smallest margin over points $x \in S$.

**Definition:** The margin $\gamma$ of a set of examples $S$ is the maximum $\gamma_w$ over all linear separators $w$. 

*Slide from Nina Balcan*
Analysis: Perceptron

Perceptron Mistake Bound

**Guarantee:** If data has margin $\gamma$ and all points inside a ball of radius $R$, then Perceptron makes $\leq \left(\frac{R}{\gamma}\right)^2$ mistakes.

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn’t change the number of mistakes; algo is invariant to scaling.)
Analysis: Perceptron

Perceptron Mistake Bound

**Theorem 0.1** (Block (1962), Novikoff (1962)).

Given dataset: \( \mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N \).

Suppose:

1. **Finite size inputs:** \( ||x^{(i)}|| \leq R \)
2. **Linearly separable data:** \( \exists \theta^* \text{ s.t. } ||\theta^*|| = 1 \) and \( y^{(i)}(\theta^* \cdot x^{(i)}) \geq \gamma, \forall i \)

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

\[
k \leq (R/\gamma)^2
\]
Proof of Perceptron Mistake Bound:

We will show that there exist constants $A$ and $B$ s.t.

$$A k \leq \| \theta^{(k+1)} \| \leq B \sqrt{k}$$
Theorem 0.1 (Block (1962), Novikoff (1962)).
Given dataset: \( \mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N} \).
Suppose:

1. Finite size inputs: \( \|x^{(i)}\| \leq R \)
2. Linearly separable data: \( \exists \theta^* \) s.t. \( \|\theta^*\| = 1 \) and \( y^{(i)}(\theta^* \cdot x^{(i)}) \geq \gamma, \forall i \)

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

\[
k \leq \left(\frac{R}{\gamma}\right)^2
\]

Algorithm 1 Perceptron Learning Algorithm (Online)

\begin{enumerate}
\item \textbf{procedure} PERCEPTRON(\( \mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots \} \))
\item \( \theta \leftarrow 0, k = 1 \) \hspace{1cm} \( \triangleright \) Initialize parameters
\item \textbf{for} \( i \in \{1, 2, \ldots \} \) \textbf{do} \hspace{1cm} \( \triangleright \) For each example
\item \hspace{1cm} \textbf{if} \( y^{(i)}(\theta^{(k)} \cdot x^{(i)}) \leq 0 \) \textbf{then} \hspace{1cm} \( \triangleright \) If mistake
\item \hspace{2cm} \( \theta^{(k+1)} \leftarrow \theta^{(k)} + y^{(i)}x^{(i)} \) \hspace{1cm} \( \triangleright \) Update parameters
\item \hspace{1cm} \( k \leftarrow k + 1 \)
\item \textbf{return} \( \theta \)
\end{enumerate}
Analysis: Perceptron

Whiteboard:
Proof of Perceptron Mistake Bound
Proof of Perceptron Mistake Bound:
Part 1: for some $A$, $Ak \leq ||\theta^{(k+1)}||$

$\theta^{(k+1)} \cdot \theta^* = (\theta^{(k)} + y^{(i)}x^{(i)})\theta^*$

- by Perceptron algorithm update

$= \theta^{(k)} \cdot \theta^* + y^{(i)}(\theta^* \cdot x^{(i)})$

$\geq \theta^{(k)} \cdot \theta^* + \gamma$

- by assumption

$\Rightarrow \theta^{(k+1)} \cdot \theta^* \geq k\gamma$

- by induction on $k$ since $\theta^{(1)} = 0$

$\Rightarrow ||\theta^{(k+1)}|| \geq k\gamma$

since $||w|| \times ||u|| \geq w \cdot u$ and $||\theta^*|| = 1$

Cauchy-Schwartz inequality
Analysis: Perceptron

Proof of Perceptron Mistake Bound:
Part 2: for some B, \( \| \mathbf{\theta}^{(k+1)} \| \leq B \sqrt{k} \)

\[
\| \mathbf{\theta}^{(k+1)} \|^2 = \| \mathbf{\theta}^{(k)} + y^{(i)} \mathbf{x}^{(i)} \|^2
\]

by Perceptron algorithm update

\[
= \| \mathbf{\theta}^{(k)} \|^2 + (y^{(i)})^2 \| \mathbf{x}^{(i)} \|^2 + 2y^{(i)} (\mathbf{\theta}^{(k)} \cdot \mathbf{x}^{(i)})
\]

\[
\leq \| \mathbf{\theta}^{(k)} \|^2 + (y^{(i)})^2 \| \mathbf{x}^{(i)} \|^2
\]

since k-th mistake \( y^{(i)} (\mathbf{\theta}^{(k)} \cdot \mathbf{x}^{(i)}) \leq 0 \)

\[
= \| \mathbf{\theta}^{(k)} \|^2 + R^2
\]

since \( (y^{(i)})^2 \| \mathbf{x}^{(i)} \|^2 = \| \mathbf{x}^{(i)} \|^2 = R^2 \) by assumption and \( (y^{(i)})^2 = 1 \)

\[
\Rightarrow \| \mathbf{\theta}^{(k+1)} \|^2 \leq k R^2
\]

by induction on \( k \) since \( (\mathbf{\theta}^{(1)})^2 = 0 \)

\[
\Rightarrow \| \mathbf{\theta}^{(k+1)} \| \leq \sqrt{k} R
\]
Part 3: Combining the bounds finishes the proof.

\[ k \gamma \leq \| \theta^{(k+1)} \| \leq \sqrt{kR} \]

\[ \Rightarrow k \leq (R/\gamma)^2 \]

The total number of mistakes must be less than this.
Learning for Perceptron also works if we have a fixed training dataset, $D$. We call this the “batch” setting in contrast to the “online” setting that we’ve discussed so far.

**Algorithm 1** Perceptron Learning Algorithm (Batch)

1. **procedure** `PERCEPTRON($D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$)`
2. $\theta \leftarrow 0$ \hspace{1cm} $\triangleright$ Initialize parameters
3. **while** not converged **do**
4. **for** $i \in \{1, 2, \ldots, N\}$ **do** \hspace{1cm} $\triangleright$ For each example
5. $\hat{y} \leftarrow \text{sign}(\theta^T x^{(i)})$ \hspace{1cm} $\triangleright$ Predict
6. **if** $\hat{y} \neq y^{(i)}$ **then** \hspace{1cm} $\triangleright$ If mistake
7. $\theta \leftarrow \theta + y^{(i)} x^{(i)}$ \hspace{1cm} $\triangleright$ Update parameters
8. **return** $\theta
Learning for Perceptron also works if we have a fixed training dataset, D. We call this the “batch” setting in contrast to the “online” setting that we’ve discussed so far.

**Discussion:**
The Batch Perceptron Algorithm can be derived in two ways.
1. By extending the online Perceptron algorithm to the batch setting (as mentioned above)
2. By applying *Stochastic Gradient Descent (SGD)* to minimize a so-called *Hinge Loss* on a linear separator
Extensions of Perceptron

• **Kernel Perceptron**
  – Choose a kernel $K(x', x)$
  – Apply the **kernel trick** to Perceptron
  – Resulting algorithm is **still very simple**

• **Structured Perceptron**
  – Basic idea can also be applied when $y$ ranges over an exponentially large set
  – Mistake bound **does not** depend on the size of that set
Matching Game

Goal: Match the Algorithm to its Update Rule

1. SGD for Logistic Regression
   \[ h_\theta(x) = p(y|x) \]

2. Least Mean Squares
   \[ h_\theta(x) = \theta^T x \]

3. Perceptron
   \[ h_\theta(x) = \text{sign}(\theta^T x) \]

4. \[ \theta_k \leftarrow \theta_k + (h_\theta(x^{(i)}) - y^{(i)}) \]

5. \[ \theta_k \leftarrow \theta_k + \frac{1}{1 + \exp \lambda(h_\theta(x^{(i)}) - y^{(i)})} \]

6. \[ \theta_k \leftarrow \theta_k + \lambda(h_\theta(x^{(i)}) - y^{(i)})x_k^{(i)} \]

A. 1=5, 2=4, 3=6
B. 1=5, 2=6, 3=4
C. 1=6, 2=4, 3=4
D. 1=5, 2=6, 3=6
E. 1=6, 2=6, 3=6
Summary: Perceptron

• Perceptron is a **linear classifier**

• **Simple learning algorithm**: when a mistake is made, add / subtract the features

• For linearly separable and inseparable data, we can **bound the number of mistakes** (geometric argument)

• Extensions support nonlinear separators and structured prediction
DISCRIMINATIVE AND GENERATIVE CLASSIFIERS
Generative vs. Discriminative

• **Generative Classifiers:**
  – Example: Naïve Bayes
  – Define a joint model of the observations \( x \) and the labels \( y \): \( p(x, y) \)
  – Learning maximizes (joint) likelihood
  – Use Bayes’ Rule to classify based on the posterior:
    \[
    p(y|x) = \frac{p(x|y)p(y)}{p(x)}
    \]

• ** Discriminative Classifiers:**
  – Example: Logistic Regression
  – Directly model the conditional: \( p(y|x) \)
  – Learning maximizes conditional likelihood
Generative vs. Discriminative

Whiteboard

– Contrast: To model p(x) or not to model p(x)?
Generative vs. Discriminative

Finite Sample Analysis (Ng & Jordan, 2002)
[Assume that we are learning from a finite training dataset]

**If model assumptions are correct:** Naive Bayes is a more efficient learner (requires fewer samples) than Logistic Regression

**If model assumptions are incorrect:** Logistic Regression has lower asymptotic error, and does better than Naïve Bayes
solid: NB  dashed: LR
Naïve Bayes makes stronger assumptions about the data but needs fewer examples to estimate the parameters.

Generative vs. Discriminative Learning (Parameter Estimation)

Naïve Bayes:
Parameters are decoupled → Closed form solution for MLE

Logistic Regression:
Parameters are coupled → No closed form solution – must use iterative optimization techniques instead
Naïve Bayes vs. Logistic Reg.

Learning (MAP Estimation of Parameters)

**Bernoulli Naïve Bayes:**
Parameters are probabilities $\rightarrow$ Beta prior (usually) pushes probabilities away from zero / one extremes

**Logistic Regression:**
Parameters are not probabilities $\rightarrow$ Gaussian prior encourages parameters to be close to zero

(effectively pushes the probabilities away from zero / one extremes)
Naïve Bayes vs. Logistic Reg.

Features

**Naïve Bayes:**
Features $x$ are assumed to be conditionally independent given $y$. (i.e. Naïve Bayes Assumption)

**Logistic Regression:**
No assumptions are made about the form of the features $x$. They can be dependent and correlated in any fashion.