Text Classification
+
Machine Learning Review 3

CS 287
Review: Logistic Regression (Murphy, p 268)

Cons

▶ Harder to fit versus naive Bayes.
▶ Must fit all classes together.
▶ Not a good fit for semi-supervised/missing data cases

Pros

▶ Better calibrated probability estimates
▶ Natural handling of feature input
  ▶ Features likely not multinomials
▶ (For us) extend naturally to neural networks
Review: Gradients for Softmax Regression

For multiclass logistic regression:

\[
\frac{\partial L(y, \hat{y})}{\partial z_i} = \sum_j \frac{\partial \hat{y}_j}{\partial z_i} 1(j = c) = \begin{cases} 
-(1 - \hat{y}_i) & i = c \\
\hat{y}_i & \text{ow.}
\end{cases}
\]

Therefore for parameters \( \theta \),

\[
\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial z_i} \quad \frac{\partial L}{\partial W_{f,i}} = x_f \frac{\partial L}{\partial z_i}
\]
Review: Gradients for Softmax Regression

For multiclass logistic regression:

\[
\frac{\partial L(y, \hat{y})}{\partial z_i} = \sum_j \frac{\partial \hat{y}_j}{\partial z_i} \frac{\mathbf{1}(j = c)}{\hat{y}_j} = \begin{cases} 
- (1 - \hat{y}_i) & i = c \\
\hat{y}_i & \text{ow.}
\end{cases}
\]

Therefore for parameters \( \theta \),

\[
\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial z_i} \quad \frac{\partial L}{\partial W_{f,i}} = x_f \frac{\partial L}{\partial z_i}
\]

Intuition:

- Nothing happens on correct classification.
- Weight of true features increases based on prob not given.
- Weight of false features decreases based on prob given.
Gradient-Based Optimization: SGD

procedure SGD
    while training criterion is not met do
        Sample a training example $x_i, y_i$
        Compute the loss $L(\hat{y}_i, y_i; \theta)$
        Compute gradients $\hat{g}$ of $L(\hat{y}_i, y_i; \theta)$ with respect to $\theta$
        $\theta \leftarrow \theta - \eta \hat{g}$
    end while
    return $\theta$
end procedure
Quiz: Softmax Regression

Given bag-of-word features

$$\mathcal{F} = \{\text{The, movie, was, terrible, rocked, A}\}$$

and two training data points:

- Class 1: The movie was terrible
- Class 2: The movie rocked

Assume that we start with parameters $$\mathbf{W} = 0$$ and $$\mathbf{b} = 0$$, and we train with learning rate $$\eta = 1$$ and $$\lambda = 0$$. What is the loss and the parameters after one pass through the data in order?
First iteration,

\[ \hat{y}_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \]

\[ L(y_1, \hat{y}_1) = -\log 0.5 \]

\[ W = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0 & 0 \\ -0.5 & -0.5 & -0.5 & -0.5 & 0 & 0 \end{bmatrix} \]

\[ b = \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix} \]
Answer: Softmax Regression (2)

Second iteration,

\[ \hat{y}_1 = \text{softmax}([1.5 \ - \ 1.5]) \approx [0.95 \ 0.05] \]

\[ L(y_2, \hat{y}_2) = -\log 0.05 \]

\[ W \approx \begin{bmatrix} -0.45 & -0.45 & 0.5 & 0.5 & -0.95 & 0 \\ 0.45 & 0.45 & -0.5 & -0.5 & 0.95 & 0 \end{bmatrix} \]

\[ b = \begin{bmatrix} -0.45 \\ 0.45 \end{bmatrix} \]
Today’s Class

So far

▶ Naive Bayes (Multinomial)
▶ Multiclass Logistic Regression (SGD)

Today

▶ Multiclass Hinge-loss
▶ More about optimization
Contents

Multiclass Hinge-Loss

Gradients

Black-Box Optimization
Other Loss Functions

What if we just try to directly find $\mathbf{W}$ and $\mathbf{b}$?

\[ \hat{y} = \mathbf{xW} + \mathbf{b} \]

- No longer a probabilistic interpretation.
- Just try to find parameters that fit training data.
0/1 Loss

Just count the number of training examples we classify correctly,

\[ \mathcal{L}(\theta) = \sum_{i=1}^{n} L_{0/1}(y, \hat{y}) = 1(\arg \max_{c'} \hat{y}_{c'} \neq c) \]
0/1 Loss

Just count the number of training examples we classify correctly,

\[ \mathcal{L}(\theta) = \sum_{i=1}^{n} L_{0/1}(y, \hat{y}) = 1(\arg \max_{c'} \hat{y}_{c'} \neq c) \]

\[ \frac{\partial L(y, \hat{y})}{\partial \hat{y}_j} = \begin{cases} 
0 & j = c \\
0 & \text{o.w.} 
\end{cases} \]

\[ L_{0/1}([x \ y]) = 1(x > y) \]
Hinge Loss

\[ \mathcal{L}(\theta) = \sum_{i=1}^{n} L_{\text{hinge}}(y, \hat{y}) \]

\[ L_{\text{hinge}}(y, \hat{y}) = \max\{0, 1 - (\hat{y}_c - \hat{y}_{c'})\} \]

Where

- Let \( c \) be defined as true class \( y_{i,c} = 1 \)

\[ c' = \arg \max_{i \in C \setminus \{c\}} \hat{y}_i \]
Hinge Loss

\[ \mathcal{L}(\theta) = \sum_{i=1}^{n} L_{\text{hinge}}(y, \hat{y}) \]

\[ L_{\text{hinge}}(y, \hat{y}) = \max\{0, 1 - (\hat{y}_c - \hat{y}_{c'})\} \]

Where

- Let \( c \) be defined as true class \( y_{i,c} = 1 \)

\[ c' = \arg \max_{i \in C \setminus \{c\}} \hat{y}_i \]

Minimizing hinge loss is an upper-bound for 0/1.

\[ L_{\text{hinge}}(y, \hat{y}) \geq L_{0/1}(y, \hat{y}) \]
Hinge Loss

\[
hinge(\hat{y}) = \mathbf{1}(\max\{0, 1 - (y - x)\})
\]

\[
\arg\max([x \ y]) = \mathbf{1}(x > y)
\]
Important Case: Hinge-loss for Binary

\[ L_{hinge}([0, 1], [x, y]) = \max\{0, 1 - (y - x)\} = \text{ReLU}(1 - (y - x)) \]

Neural network name (Rectified linear unit):

\[ \text{ReLU}(t) = \max\{0, t\} \]
Hinge-Loss Properties

Complete objective:

\[ L_{hinge}(\theta) = \sum_{i=1}^{n} \max\{0, 1 - (\hat{y}_c - \hat{y}_{c'})\} \]

\[ = \sum_{i=1}^{n} \max\{0, 1 - (\hat{y}_c - \max_{c' \in \mathcal{C} \setminus \{c\}} \hat{y}_{c'})\} \]

- Apply convexity rules: Linear \( \hat{y} \) is convex, max of convex functions is convex, linear + convex is convex, sum of convex functions is convex (Boyd and Vandenberghe, 2004 p. 72-74)

- However, non-differentiable because of max.
Piecewise Linear Objective

\[ \mathcal{L}(\theta) = \sum_{i=1}^{n} \max\{0, 1 - (\hat{y}_c - \hat{y}_{c'})\} \]

\[ 10 \times \max\{0, 1 - (y - x)\} + 5 \times \max\{0, 1 - (x - y)\} \]
Objective with Regularization

\[ \mathcal{L}(\theta) = \sum_{i=1}^{n} \max\{0, 1 - (\hat{y}_c - \hat{y}_{c'})\} + \lambda \|\theta\|^2 \]

\[ 10 \max\{0, 1 - (y - x)\} + 5 \max\{0, 1 - (x - y)\} + 5 \|\theta\|^2 \]
Contents

Multiclass Hinge-Loss

Gradients

Black-Box Optimization
(Sub)Gradient Rule

- Technically ReLU is non-differentiable.
- Only an issue at 0, generally for “ties”.
- We informally use subgradients,

\[
\frac{d \text{ReLU}(x)}{dx} = \begin{cases} 
1 & x > 0 \\
0 & x < 0 \\
1 \text{ or } 0 & \text{ o.w}
\end{cases}
\]

Generally,

\[
\frac{d \max_{v'}(f(x, v'))}{dx} = f'(x, \hat{v}) \text{ for any } \hat{v} \in \arg \max_{v'} f(x, v')
\]
Symbolic Gradients

- Let \( c \) be defined as true class
- Let \( c' \) be defined as the highest scoring non-true class

\[
c' = \arg \max_{i \in C \setminus \{c\}} \hat{y}_i
\]

- Partials of \( L(y, \hat{y}) \)

\[
\frac{\partial L(y, k\hat{y})}{\partial \hat{y}_j} = \begin{cases} 
0 & \hat{y}_c - \hat{y}_{c'} > 1 \\
1 & j = c' \\
-1 & j = c \\
0 & o.w.
\end{cases}
\]

Intuition: If wrong or close to wrong, improve correct and lower closest incorrect.
Notes: Hinge Loss: Regularization

- Many different names,
  - Margin Classifier
  - Multiclass Hinge
  - Linear SVM

- Important to use regularization.

\[ \mathcal{L}(\theta) = -\sum_{i=1}^{n} L(\hat{y}, y) + ||\theta||^2_2 \]

- Can be much more efficient to train than LR. (No partition).
Results: Longer Reviews

<table>
<thead>
<tr>
<th>Our results</th>
<th>RT-2k</th>
<th>IMDB</th>
<th>Subj.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNB-uni</td>
<td>83.45</td>
<td>83.55</td>
<td>92.58</td>
</tr>
<tr>
<td>MNB-bi</td>
<td>85.85</td>
<td>86.59</td>
<td>93.56</td>
</tr>
<tr>
<td>SVM-uni</td>
<td>86.25</td>
<td>86.95</td>
<td>90.84</td>
</tr>
<tr>
<td>SVM-bi</td>
<td>87.40</td>
<td>89.16</td>
<td>91.74</td>
</tr>
<tr>
<td>NBSVM-uni</td>
<td>87.80</td>
<td>88.29</td>
<td>92.40</td>
</tr>
<tr>
<td>NBSVM-bi</td>
<td><strong>89.45</strong></td>
<td><strong>91.22</strong></td>
<td><strong>93.18</strong></td>
</tr>
<tr>
<td>BoW (bnc)</td>
<td>85.45</td>
<td>87.8</td>
<td>87.77</td>
</tr>
<tr>
<td>BoW (bΔt‘c)</td>
<td>85.8</td>
<td>88.23</td>
<td>85.65</td>
</tr>
<tr>
<td>LDA</td>
<td>66.7</td>
<td>67.42</td>
<td>66.65</td>
</tr>
<tr>
<td>Full+BoW</td>
<td>87.85</td>
<td>88.33</td>
<td>88.45</td>
</tr>
<tr>
<td>Full+Unlab’d+BoW</td>
<td><strong>88.9</strong></td>
<td><strong>88.89</strong></td>
<td><strong>88.13</strong></td>
</tr>
</tbody>
</table>

IMDB (longer movie review), Subj (longer subjectivity)

- NBSVM is hinge-loss interpolated with Naive Bayes.
Optimization Methods

**Goal:** Minimize function $\mathcal{L} : \mathbb{R}^{|\theta|} \mapsto \mathbb{R}$

First-order Methods

$$\mathcal{L}(\theta + \delta) \approx \mathcal{L}(\theta) + \mathcal{L}'(\theta)^\top \delta$$

▸ Require computing $\mathcal{L}(\theta)$ and gradient $\mathcal{L}'(\theta)$.

Second-order Methods

$$\mathcal{L}(\theta + \delta) \approx \mathcal{L}(\theta) + \mathcal{L}'(\theta)^\top \delta + 1/2\delta^\top \mathbf{H} \delta^\top$$

▸ Require computing $\mathcal{L}(\theta)$ and gradient $\mathcal{L}'(\theta)$ and Hessian $\mathbf{H}$.

Stochastic Methods

▸ Require computing $\mathbb{E}(\mathcal{L}(\theta))$ and expected gradient.
Gradient Descent

while training criterion is not met do
    $k \leftarrow 0$
    $\hat{g} \leftarrow 0$
    for $i = 1$ to $n$ do
        Compute the loss $L(\hat{y}_i, y_i; \theta)$
        Compute gradients $g'$ of $L(\hat{y}_i, y_i; \theta)$ with respect to $\theta$
        $\hat{g} \leftarrow \hat{g} + \frac{1}{n}g'$
    end for
    $\theta_{k+1} \leftarrow \theta_k - \eta_k \hat{g}$
    $k \leftarrow k + 1$
end while

return $\theta$
Choosing the Learning Rate

- Gradient Descent converging
- Gradient Descent diverging (stepsize too large)

\[
f(x + t\Delta x)
\]

\[
f(x) + t\nabla f(x)^T \Delta x
\]

\[
f(x) + \alpha t \nabla f(x)^T \Delta x
\]

\[
t = 0 \quad t \quad t_0
\]
Gradient Descent with Momentum

Standard Gradient Descent (figure from Boyd):

\[ \theta_{k+1} \leftarrow \theta_k - \eta_k \hat{g} \]

Momentum terms:

\[ \theta_{k+1} \leftarrow \theta_k - \eta_k \hat{g} + \mu_k (\theta_k - \theta_{k-1}) \]

- Also known as: Heavy-ball method
Second-Order

Requires compute Hessian $\hat{H}$

Second-order update becomes:

$$\theta_{k+1} \leftarrow \theta_k - \eta_k \hat{H}^{-1}\hat{g}$$

- Used for strictly convex functions (although there are variants)
- Also known as: Newton’s Method
Quasi-Newton Methods

Construct an approximate Hessian from first-order information

- **BFGS**
  - construct approx. Hessian directly
  - $O(|\theta|^2)$ space

- **L-BFGS**;
  - limited-memory BFGS, only save last $m$ gradients
  - can often set $m < 20$ or smaller

Fast implementations available, specific details are beyond scope of course.
Stochastic Methods

- Minimize function $L(\theta)$
- Require computing $\mathbb{E}(L(\theta))$ and gradient
- Typically, we by sampling a subset of the data. computing a gradient, and updating
- Other first-order optimizers (like momentum) can be used.
while training criterion is not met do
  Sample a minibatch of $m$ examples $(x_1, y_1), \ldots, (x_m, y_m)$
  \[ \hat{g} \leftarrow 0 \]
  for $i = 1$ to $m$ do
    Compute the loss $L(\hat{y}_i, y_i; \theta)$
    Compute gradients $g'$ of $L(\hat{y}_i, y_i; \theta)$ with respect to $\theta$
    \[ \hat{g} \leftarrow \hat{g} + \frac{1}{m} g' \]
  end for
  \[ \theta \leftarrow \theta - \eta_k \hat{g} \]
end while
return $\theta$
Tricks On Using SGD (Bottou 2012)

- Can be crucial to experiment with learning rate.
- Often useful to use development set for stopping.
- Shuffle data first and run over it.
- Also describes “averaged” versions which work well in practice.
Optimization in NLP

- For convex batch-methods:
  - L-BFGS is easy to use and effective.
  - Nice for verifying results.
  - Sometimes even $m$-times the parameters is a lot though.

- For both convex, and especially, non-convex problems:
  - SGD and variants are dominant.
  - Trade-off of speed vs. exact optimization.
  - Also see notes on AdaGrad, another popular method.