This Lecture

- Linear classification fundamentals
- Three discriminative models: logistic regression, perceptron, SVM
  - Different motivations but very similar update rules / inference!
- Optimization
- Sentiment analysis
Classification

- Datapoint \( x \) with label \( y \in \{0, 1\} \)
- Embed datapoint in a feature space \( f(x) \in \mathbb{R}^n \)
  but in this lecture \( f(x) \) and \( x \) are interchangeable
- Linear decision rule: \( w^T f(x) + b > 0 \)
  \[ w^T f(x) > 0 \]
- Can delete bias if we augment feature space:
  \( f(x) = [0.5, 1.6, 0.3] \)
  \[ [0.5, 1.6, 0.3, 1] \]

Classification: Sentiment Analysis

- "Kernel trick" does this for "free," but is too expensive to use in NLP applications, training is \( O(n^2) \) instead of \( O(n \cdot (\text{num feats})) \)

Feature Representation

- Convert this example to a vector using bag-of-words features
- [contains the] position 0
- [contains a] position 1
- [contains was] position 2
- [contains movie] position 3
- [contains film] position 4
- \( f(x) = [0, 0, 1, 1, 0, \ldots] \)
- Very large vector space (size of vocabulary), sparse features (how many?)
- Requires indexing the features (mapping them to axes)
- More sophisticated feature mappings possible (tf-idf), as well as lots of other features: n-grams, character n-grams, parts of speech, lemmas, …
Generative vs. Discriminative Modeling

- Data point $x = (x_1, \ldots, x_n)$, label $y \in \{0, 1\}$
- Generative models: probabilistic models of $P(x, y)$
  - Compute $P(y|x)$, predict $\arg\max_y P(y|x)$ to classify
    \[ P(y|x) = \frac{P(y)P(x|y)}{P(x)} \propto P(y)P(x|y) \]
- Discriminative models model $P(y|x)$ directly, compute $\arg\max_y P(y|x)$
  - Examples: Naive Bayes (see textbook), Hidden Markov Models
  - Examples: logistic regression
- Cannot draw samples of $x$, but typically better classifiers

Logistic Regression

- $P(y = +|x) = \text{logistic}(w^T x)$
- $P(y = +|x) = \frac{\exp(\sum_{i=1}^n w_i x_i)}{1 + \exp(\sum_{i=1}^n w_i x_i)}$
- To learn weights: maximize discriminative log likelihood of data ($\log P(y|x)$)
  \[ \mathcal{L}(x, y) = \sum_j \log P(y_j|x_j) \quad \text{corpus-level LL} \]
  \[ \mathcal{L}(x_j, y_j = +) = \log P(y_j = +|x_j) = \sum_{i=1}^n w_i x_{ji} - \log \left( 1 + \exp \left( \sum_{i=1}^n w_i x_{ji} \right) \right) \quad \text{sum over features} \]
- Log likelihood of one (positive) example LL
- To learn weights: maximize $\mathcal{L}(x_j, y_j = +)$
- Derivative of $\mathcal{L}(x_j, y_j = +)$
  \[ \frac{\partial \mathcal{L}(x_j, y_j = +)}{\partial w_i} = x_{ji} - \frac{1}{1 + \exp(\sum_{i=1}^n w_i x_{ji})} \frac{\partial}{\partial w_i} \left( 1 + \exp(\sum_{i=1}^n w_i x_{ji}) \right) \]
  \[ = x_{ji} \frac{1}{1 + \exp(\sum_{i=1}^n w_i x_{ji})} \exp(\sum_{i=1}^n w_i x_{ji}) \]
  \[ = x_{ji} - x_{ji} \frac{\exp(\sum_{i=1}^n w_i x_{ji})}{1 + \exp(\sum_{i=1}^n w_i x_{ji})} = x_{ji} (1 - P(y_j = +|x_j)) \]
Logistic Regression

- Gradient of $w_i$ on positive example $= x_{ji}(1 - P(y_j = + | x_j))$
  If $P(+) \approx 1$, make very little update. Otherwise make $w_i$ look more like $x_{ji}$, which will increase $P(+)$. 
- Gradient of $w_i$ on negative example $= x_{ji}(-P(y_j = + | x_j))$
  If $P(+) \approx 0$, make very little update. Otherwise make $w_i$ look less like $x_{ji}$, which will decrease $P(+)$. 
- Let $y_j = 1$ for positive instances, $y_j = 0$ for negative instances. 
- Can combine these gradients as $x_j(y_j - P(y_j = 1 | x_j))$

Example

- (1) this movie was great! would watch again + $f(x_1) = [1 \ 1]
- (2) I expected a great movie and left happy + $f(x_2) = [1 \ 1]
- (3) great potential but ended up being a flop - $f(x_3) = [1 \ 0]

$[\text{contains great}]$ $[\text{contains movie}]$

$w = [0, 0] \rightarrow P(y = 1 | x_1) = \exp(0)/(1 + \exp(0)) = 0.5 \rightarrow g = [0.5, 0.5]$

$w = [0.5, 0.5] \rightarrow P(y = 1 | x_2) = \text{logistic}(1) = 0.75 \rightarrow g = [0.25, 0.25]$

$w = [0.75, 0.75] \rightarrow P(y = 1 | x_3) = \text{logistic}(0.75) = 0.67 \rightarrow g = [-0.67, 0]$

$w = [0.08, 0.75] \ldots$

$\begin{align*}
  x_j(y_j - P(y_j = 1 | x_j))
\end{align*}$

Regularization

- Regularizing an objective can mean many things, including an L2-norm penalty to the weights:
  $$\sum_{j=1}^{n} L(x_j, y_j) - \lambda \|w\|^2_2$$
- Keeping weights small can prevent overfitting.
- For most of the NLP models we build, explicit regularization isn’t necessary.
- Early stopping.
- Large numbers of sparse features are hard to overfit in a really bad way.
- For neural networks: dropout and gradient clipping.

Logistic Regression: Summary

- Model
  $$P(y = +|x) = \frac{\exp(\sum_{i=1}^{n} w_i x_i)}{1 + \exp(\sum_{i=1}^{n} w_i x_i)}$$
- Inference
  $$\text{argmax}_y P(y|x)$$
  $$P(y = 1|x) \geq 0.5 \iff w^\top x \geq 0$$
- Learning: gradient ascent on the (regularized) discriminative log-likelihood.
Perceptron/SVM

Perceptron

- Simple error-driven learning approach similar to logistic regression
- Decision rule: $w^T x > 0$
  - If incorrect: if positive, $w \leftarrow w + x$
  - if negative, $w \leftarrow w - x$
- Guaranteed to eventually separate the data if the data are separable

Logistic Regression

$w \leftarrow w + x(1 - P(y = 1|x))$
$w \leftarrow w - xP(y = 1|x)$

Support Vector Machines

- Many separating hyperplanes — is there a best one?

Support Vector Machines

- Many separating hyperplanes — is there a best one?
Support Vector Machines

- Constraint formulation: find $w$ via following quadratic program:

\[
\begin{align*}
\text{Minimize} & \quad \|w\|_2^2 \\
\text{s.t.} & \quad \forall j \quad w^T x_j \geq 1 \text{ if } y_j = 1 \\
& \quad w^T x_j \leq -1 \text{ if } y_j = 0
\end{align*}
\]

- As a single constraint:

\[
\forall j \quad (2y_j - 1)(w^T x_j) \geq 1
\]

- Generally no solution (data is generally non-separable) — need slack!

N-Slack SVMs

\[
\begin{align*}
\text{Minimize} & \quad \lambda \|w\|_2^2 + \sum_{j=1}^{m} \xi_j \\
\text{s.t.} & \quad \forall j \quad (2y_j - 1)(w^T x_j) \geq 1 - \xi_j \quad \forall j \quad \xi_j \geq 0
\end{align*}
\]

- The $\xi_j$ are a “fudge factor” to make all constraints satisfied

- Take the gradient of the objective:

\[
\frac{\partial}{\partial w_i} \xi_j = 0 \text{ if } \xi_j = 0 \\
\frac{\partial}{\partial w_i} \xi_j = (2y_j - 1)x_{ji} \text{ if } \xi_j > 0
\]

\[
= x_{ji} \text{ if } y_j = 1, \quad -x_{ji} \text{ if } y_j = 0
\]

- Looks like the perceptron! But updates more frequently

Gradients on Positive Examples

- Logistic regression

\[
x(1 - \text{logistic}(w^T x))
\]

- Perceptron

\[
x \text{ if } w^T x < 0, \quad \text{else } 0
\]

- SVM (ignoring regularizer)

\[
x \text{ if } w^T x < 1, \quad \text{else } 0
\]

*gradients are for maximizing things, which is why they are flipped

Comparing Gradient Updates (Reference)

- Logistic regression (unregularized)

\[
x(y - P(y = 1|x)) = x(y - \text{logistic}(w^T x))
\]

- Perceptron

\[
(2y - 1)x \text{ if classified incorrectly} \\
0 \text{ else}
\]

- SVM

\[
(2y - 1)x \text{ if not classified correctly with margin of } 1 \\
0 \text{ else}
\]

y = 1 for pos, 0 for neg
Structured Prediction

- Four elements of a structured machine learning method:
  - Model: probabilistic, max-margin, deep neural network
  - Objective
    - Inference: just maxes and simple expectations so far, but will get harder
    - Training: gradient descent?

Optimization

- Stochastic gradient ascent
  - $w \leftarrow w + \alpha g$, $g = \frac{\partial}{\partial w} \mathcal{L}$
  - Very simple to code up
  - "First-order" technique: only relies on having gradient
  - Can avg gradient over a few examples and apply update once (minibatch)
  - Setting step size is hard (decrease when held-out performance worsens?)
- Newton’s method
  - Second-order technique
  - Optimizes quadratic instantly
    - Inverse Hessian: $n \times n$ mat, expensive!
- Quasi-Newton methods: L-BFGS, etc. approximate inverse Hessian

AdaGrad

- Optimized for problems with sparse features
- Per-parameter learning rate: smaller updates are made to parameters that get updated frequently
  $$w_i \leftarrow w_i + \alpha \frac{1}{\sqrt{\epsilon + \sum_{t=1}^{t} g_{t,i}^2}} g_{t,i} \quad \text{(smoothed) sum of squared gradients from all updates}$$
  - Generally more robust than SGD, requires less tuning of learning rate
  - Other techniques for optimizing deep models — more later!

Duchi et al. (2011)
Implementation

- Supposing $k$ active features on an instance, gradient is only nonzero on $k$ dimensions
  \[ w \leftarrow w + \alpha g, \quad g = \frac{\partial}{\partial w} L \]
- $k < 100$, total num features = 1M+ on many problems
- Be smart about applying updates!
- In PyTorch: applying sparse gradients only works for certain optimizers and sparse updates are very slow. The code we give you is much faster

---

Sentiment Analysis

**this movie was great! would watch again**

**the movie was gross and overwrought, but I liked it**

**this movie was not really very enjoyable**

- Bag-of-words doesn’t seem sufficient (discourse structure, negation)
- There are some ways around this: extract bigram feature for “not X” for all X following the not

---

Sentiment Analysis

<table>
<thead>
<tr>
<th>Features</th>
<th># of features</th>
<th>frequency or presence?</th>
<th>NB</th>
<th>ME</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) unigrams</td>
<td>16165</td>
<td>freq.</td>
<td>78.7</td>
<td>N/A</td>
<td>72.8</td>
</tr>
<tr>
<td>(2) unigrams</td>
<td></td>
<td>pres.</td>
<td>81.0</td>
<td>80.4</td>
<td>82.9</td>
</tr>
<tr>
<td>(3) unigrams+bigrams</td>
<td>32330</td>
<td>pres.</td>
<td>80.6</td>
<td>80.8</td>
<td>82.7</td>
</tr>
<tr>
<td>(4) bigrams</td>
<td>16165</td>
<td>pres.</td>
<td>77.3</td>
<td>77.4</td>
<td>77.1</td>
</tr>
<tr>
<td>(5) unigrams+POS</td>
<td>16055</td>
<td>pres.</td>
<td>81.5</td>
<td>80.4</td>
<td>81.9</td>
</tr>
<tr>
<td>(6) adjectives</td>
<td>2633</td>
<td>pres.</td>
<td>77.0</td>
<td>77.7</td>
<td>75.1</td>
</tr>
<tr>
<td>(7) top 2633 unigrams</td>
<td>2633</td>
<td>pres.</td>
<td>80.3</td>
<td>81.0</td>
<td>81.4</td>
</tr>
<tr>
<td>(8) unigrams+position</td>
<td>22430</td>
<td>pres.</td>
<td>81.0</td>
<td>80.1</td>
<td>81.6</td>
</tr>
</tbody>
</table>

- Simple feature sets can do pretty well!
### Sentiment Analysis

<table>
<thead>
<tr>
<th>Method</th>
<th>RT-s</th>
<th>MPQA</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNB-uni</td>
<td>77.9</td>
<td>85.3</td>
</tr>
<tr>
<td>MNB-bi</td>
<td><strong>79.0</strong></td>
<td><strong>86.5</strong></td>
</tr>
<tr>
<td>SVM-uni</td>
<td>76.2</td>
<td>86.1</td>
</tr>
<tr>
<td>SVM-bi</td>
<td>77.7</td>
<td>86.7</td>
</tr>
<tr>
<td>NBSVM-uni</td>
<td>78.1</td>
<td>86.3</td>
</tr>
<tr>
<td>NBSVM-bi</td>
<td><strong>79.4</strong></td>
<td><strong>86.3</strong></td>
</tr>
<tr>
<td>RAE</td>
<td>76.8</td>
<td>85.7</td>
</tr>
<tr>
<td>RAE-pretrain</td>
<td><strong>77.7</strong></td>
<td><strong>86.4</strong></td>
</tr>
<tr>
<td>Voting-w/Rev.</td>
<td>63.1</td>
<td>81.7</td>
</tr>
<tr>
<td>Rule</td>
<td>62.9</td>
<td>81.8</td>
</tr>
<tr>
<td>BoF-noDic.</td>
<td>75.7</td>
<td>81.8</td>
</tr>
<tr>
<td>BoF-w/Rev.</td>
<td>76.4</td>
<td>84.1</td>
</tr>
<tr>
<td>Tree-CRF</td>
<td>77.3</td>
<td>86.1</td>
</tr>
<tr>
<td>BoW/SVM</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Kim (2014) CNNs: **81.5 | 89.5**

---

### Recap

- **Logistic regression:**
  \[ P(y = 1|x) = \frac{\exp(\sum_{i=1}^{n} w_i x_i)}{1 + \exp(\sum_{i=1}^{n} w_i x_i)} \]

  - Decision rule: \[ P(y = 1|x) \geq 0.5 \iff w^T x \geq 0 \]

  - Gradient (unregularized): \[ x(y - P(y = 1|x)) \]

- **SVM:**
  Decision rule: \[ w^T x \geq 0 \]

  - (Sub)gradient (unregularized): 0 if correct with margin of 1, else \[ x(2y - 1) \]

---

**Naive Bayes is doing well!**

**Ng and Jordan (2002) — NB can be better for small data**

**Before neural nets had taken off — results weren’t that great**

---

### Sentiment Analysis

- Stanford Sentiment Treebank (SST) binary classification
- Best systems now: large pretrained networks
- 90 -> 97 over the last 2 years

---

**Wang and Manning (2012)**

---

**https://github.com/sebastianruder/NLP-progress/blob/master/english/sentiment_analysis.md**