Generalized Linear Classifiers

Joakim Nivre
Linear Models in Theory

- The linear model for binary classification:

\[
\hat{y} = \begin{cases} 
1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\
-1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b < 0 
\end{cases}
\]

- Learning as optimization (loss + regularization):

\[
\hat{\mathbf{w}}, \hat{b} = \arg\min_{\mathbf{w}, b} \mathcal{L}(\mathbf{w}, b; \mathcal{T}) + \lambda R(\mathbf{w}, b)
\]

- Gradient descent for convex optimization:

\[
\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f(\mathbf{w}, b; \mathcal{T})
\]
Linear Models in Practice

- Binary classification is sometimes useful
  - Spam filtering, spell checking, ...
- But most NLP problems involve more than two classes
  - Text categorization: news, business, culture, sports, ...
  - Word sense disambiguation: one class per sense
- And many involve structured prediction
  - Part-of-speech tagging: sequence-to-sequence
  - Dependency parsing: sequence-to-tree
Today’s Lecture

- Multiclass classification
  - Using binary classifiers (OVA, AVA)
  - Multiclass perceptron (and generalizations)
- Structured prediction
  - Using simple (unstructured) classifiers
  - Structured perceptron (and generalizations)
- But first a little more on learning strategies . . .
Minimize Error

\[ L(w, b; T) = \begin{cases} 
1 & \text{if } y\hat{y} \leq 0 \\
0 & \text{otherwise}
\end{cases} \]

The perceptron (implicitly) minimizes 0-1 loss
Minimize Error

The perceptron (or 0-1 loss) does not care about margin
Maximize Margin

\[ \mathcal{L}(\mathbf{w}, b; \mathcal{T}) = \max(0, 1 - y\hat{y}) \]

Hinge loss goes to 0 with a margin of (at least) 1

Typical of (min-error) max-margin methods: SVM, MIRA, ...
Maximize Likelihood

\[ \mathcal{L}(w, b; T) = \frac{1}{\log 2} \log(1 + \exp(-y\hat{y})) \]

Log loss improves beyond a margin of 1
Minimizing log loss means maximizing likelihood
Min Error $\neq$ Max Likelihood

- Consider a training set $\mathcal{T}$ with 100 instances
  - 99 negative instances: $\langle\langle 2, 1 \rangle, -1 \rangle$
  - 1 positive instance: $\langle\langle 2, 3 \rangle, 1 \rangle$

- Consider the weight vector $\mathbf{w} = \langle -1, 1 \rangle$
  - $\langle -1, 1 \rangle \cdot \langle 2, 1 \rangle = -1$
  - $\langle -1, 1 \rangle \cdot \langle 2, 3 \rangle = 1$

- Loss functions:
  - 0/1 loss = 0
  - Hinge loss = 0
  - Log loss = $0.452 \times 100 = 45.2$
Min Error $\neq$ Max Likelihood

Consider a training set $\mathcal{T}$ with 100 instances
- 99 negative instances: $\langle\langle 2, 1 \rangle, -1 \rangle$
- 1 positive instance: $\langle\langle 2, 3 \rangle, 1 \rangle$

Consider the weight vector $\mathbf{w} = \langle -2, 1 \rangle$
- $\langle -2, 1 \rangle \cdot \langle 2, 1 \rangle = -3$
- $\langle -2, 1 \rangle \cdot \langle 2, 3 \rangle = -1$

Loss functions:
- $0/1$ loss = 1
- Hinge loss = 2
- Log loss = $0.07 \times 99 + 1.895 \times 1 = 8.82$
Remember

- Different loss functions represent different learning strategies
- Regularization is important to prevent overfitting
- Training loss is not what we really care about
Multiclass Classification

- Can we do multiclass classification with binary classifiers?
Multiclass Classification

- Can we do multiclass classification with binary classifiers?
  - Yes, but we need more than one classifier
    - One-Versus-All (OVA): one classifier for every class \( y_j \)
    - All-Versus-All (AVA): one classifier for every pair \( y_j, y_k \)
One-Versus-All

- Given multiclass training data:
  \[
  \mathcal{T} = \{ (x^{(i)}, y^{(i)}) \}_{i=1}^N \quad y^{(i)} \in \{y_1, \ldots, y_n\}
  \]

- Create training set for each class \(y_j\):
  \[
  \mathcal{T}_j = \{ (x^{(i)}, z^{(i)}) \}_{i=1}^N \quad z^{(i)} = \begin{cases} 
  1 & \text{if } y^{(i)} = y_j \\
  -1 & \text{otherwise}
  \end{cases}
  \]

- Train one classifier (weight vector) \(w_{y_j}\) for each class \(y_j\)

- Decision rule:
  \[
  f(x) = \arg\max_y w_y \cdot x
  \]
All-Versus-All

- Given multiclass training data:
  \[ \mathcal{T} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N \quad y^{(i)} \in \{y_1, \ldots, y_n\} \]

- Create training set for each pair \(y_j, y_k\):
  \[ \mathcal{T}_{jk} = \{(x^{(i)}, z^{(i)})\}_{i=1}^{N_{j,k}} \quad z^{(i)} = \begin{cases} 
  1 & \text{if } y^{(i)} = y_j \\
  -1 & \text{if } y^{(i)} = y_k
  \end{cases} \]

- Train one classifier (weight vector) \(w_{jk}\) for each pair \(y_j, y_k\)

- Score for \(y_j\) combines all classifiers involving \(y_j\)
OVA or AVA?

- Both methods come with guarantees (see textbook)
- Both methods can work well in practice
- OVA is more efficient both at training and classification time
- Given $n$ classes:
  - OVA only needs to train and run $n$ classifiers
  - AVA requires $\frac{n(n-1)}{2}$ classifiers
Generalized Linear Models

▷ In binary classification, we use feature vectors over inputs:

\[ f(x) : \mathcal{X} \rightarrow \mathbb{R}^m \]

▷ For multiple classes, we need to represent input-output pairs:

\[ f(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m \]

▷ This can be generalized to structured outputs (more later)
Examples

- $x$ is a document and $y$ is a label

$$f_j(x, y) = \begin{cases} 
1 & \text{if } x \text{ contains the word "interest"} \\
& \text{and } y = "financial" \\
0 & \text{otherwise}
\end{cases}$$

$$f_j(x, y) = \% \text{ of words in } x \text{ with punctuation and } y = "scientific"$$

- $x$ is a word and $y$ is a part-of-speech tag

$$f_j(x, y) = \begin{cases} 
1 & \text{if } x = "bank" \text{ and } y = \text{Verb} \\
0 & \text{otherwise}
\end{cases}$$
Examples

- $x$ is a name, $y$ is a label classifying the name

\begin{align*}
f_0(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "George"} \\
& \text{and } y = \text{"Person"} \\
0 & \text{otherwise}
\end{cases} \\
f_4(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "George"} \\
& \text{and } y = \text{"Object"} \\
0 & \text{otherwise}
\end{cases}
\end{align*}

\begin{align*}
f_1(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Washington"} \\
& \text{and } y = \text{"Person"} \\
0 & \text{otherwise}
\end{cases} \\
f_5(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Washington"} \\
& \text{and } y = \text{"Object"} \\
0 & \text{otherwise}
\end{cases}
\end{align*}

\begin{align*}
f_2(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Bridge"} \\
& \text{and } y = \text{"Person"} \\
0 & \text{otherwise}
\end{cases} \\
f_6(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Bridge"} \\
& \text{and } y = \text{"Object"} \\
0 & \text{otherwise}
\end{cases}
\end{align*}

\begin{align*}
f_3(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "General"} \\
& \text{and } y = \text{"Person"} \\
0 & \text{otherwise}
\end{cases} \\
f_7(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "General"} \\
& \text{and } y = \text{"Object"} \\
0 & \text{otherwise}
\end{cases}
\end{align*}

- $x=\text{General George Washington}, y=\text{Person} \quad \rightarrow \quad f(x, y) = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$
- $x=\text{George Washington Bridge}, y=\text{Object} \quad \rightarrow \quad f(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0]$
- $x=\text{George Washington George}, y=\text{Object} \quad \rightarrow \quad f(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0]$
Block Feature Vectors

- $x=$ General George Washington, $y=$ Person $\rightarrow f(x, y) = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$
- $x=$ George Washington Bridge, $y=$ Object $\rightarrow f(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0]$
- $x=$ George Washington George, $y=$ Object $\rightarrow f(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0]$

- One equal-size block of the feature vector for each label
- Input features duplicated in each block
- Non-zero values allowed only in one block
Multiclass Linear Classification

- Let $\mathbf{w} \in \mathbb{R}^m$ be a weight vector.
- If we assume that $\mathbf{w}$ is known, then we define our classifier as:

$$
\hat{y} = \arg\max_y \mathbf{w} \cdot f(x, y) \\
= \arg\max_y \sum_{j=0}^{m} w_j \times f_j(x, y)
$$
Bias Terms

- Often linear classifiers presented as

$$\hat{y} = \arg\max_y \sum_{j=0}^{m} w_j \times f_j(x, y) + b_y$$

- Where $b$ is a bias or offset term
- But this can be folded into $f$

$x=$General George Washington, $y=$Person $\rightarrow f(x, y) = [1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0]$

$x=$General George Washington, $y=$Object $\rightarrow f(x, y) = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1]$

$f_4(x, y) = \begin{cases} 1 & y = \text{“Person”} \\ 0 & \text{otherwise} \end{cases}$

$f_9(x, y) = \begin{cases} 1 & y = \text{“Object”} \\ 0 & \text{otherwise} \end{cases}$

- $w_4$ and $w_9$ are now the bias terms for the labels
Binary Linear Classifier

Divides all points:
Multiclass Linear Classifier

Defines regions of space:

- i.e., + are all points \((x, y)\) where
  \[ + = \arg\max_y \ w \cdot f(x, y) \]
Supervised Learning

- Input: Training examples $\mathcal{T} = \{(x^{(i)}, y_t^{(i)})\}_{i=1}^N$
- Feature representation $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$
- Output: A vector $\mathbf{w}$ that optimizes some important function of the training set:
  - minimize error (Perceptron, SVMs, Boosting)
  - maximize likelihood of data (Logistic Regression, Naive Bayes)
- NB: Same as binary case except for feature representation
Perceptron Learning Algorithm

1: \( w \leftarrow 0 \)
2: for a fixed number of iterations do
3: for all \((x, y) \in T\) do
4: \( \hat{y} = \arg\max_y w \cdot f(x, y) \)
5: if \( \hat{y} \neq y \)
6: \( w = w + f(x, y) - f(x, \hat{y}) \)
7: end if
8: end for
9: end for
Perceptron Learning Algorithm

1: \( w \leftarrow 0 \)
2: for a fixed number of iterations do
3: for all \((x, y) \in T\) do
4: \( \hat{y} = \arg\max_y w \cdot f(x, y) \) [binary: \( \hat{y} = \text{sign}(w \cdot x) \)]
5: if \( \hat{y} \neq y \)
6: \( w = w + f(x, y) - f(x, \hat{y}) \)
7: end if
8: end for
9: end for
Perceptron Learning Algorithm

1: \( w \leftarrow 0 \)
2: for a fixed number of iterations do
3: for all \((x, y) \in \mathcal{T}\) do
4: \( \hat{y} = \arg\max_y w \cdot f(x, y) \)
5: if \( \hat{y} \neq y \) [binary: if \( y\hat{y} < 0 \)]
6: \( w = w + f(x, y) - f(x, \hat{y}) \)
7: end if
8: end for
9: end for
Perceptron Learning Algorithm

1: \( w \leftarrow 0 \)
2: for a fixed number of iterations do
3: for all \((x, y) \in \mathcal{T}\) do
4: \( \hat{y} = \arg\max_y w \cdot f(x, y) \)
5: if \( \hat{y} \neq y \)
6: \( w = w + f(x, y) - f(x, \hat{y}) \) [binary: \( w = w + x \)]
7: end if
8: end for
9: end for
More Generalized Classifiers

- The multiclass version with block vectors can be generalized
- Loss functions need to be adapted to new setup
- Example hinge loss:

  Binary: \[ \max(0, 1 - y\hat{y}) \]
  Multiclass: \[ \max(0, 1 - (\mathbf{w} \cdot \mathbf{f}(x, y) - \max_{y' \neq y} \mathbf{w} \cdot \mathbf{f}(x, y')))) \]
Structured Prediction

- Sometimes $\mathcal{Y}$ does not consist of simple atomic classes
- Examples:
  - **Parsing**: for a sentence $x$, $\mathcal{Y}$ is the set of possible parse trees
  - **Sequence tagging**: for a sentence $x$, $\mathcal{Y}$ is the set of possible tag sequences, e.g., part-of-speech tags, named-entity tags
  - **Machine translation**: for a source sentence $x$, $\mathcal{Y}$ is the set of possible target language sentences
- Can’t we just use our multiclass learning algorithms?
  - The size of $\mathcal{Y}$ is exponential in the length of the input $x$
  - It is non-trivial to apply our learning algorithms in such cases
Perceptron

1: \( w \leftarrow 0 \)
2: for a fixed number of iterations do
3: for all \((x, y) \in T\) do
4: \( \hat{y} = \arg \max_y w \cdot f(x, y) \)
5: if \( \hat{y} \neq y \)
6: \( w = w + f(x, y) - f(x, \hat{y}) \)
7: end if
8: end for
9: end for

Solving the argmax requires searching an exponential output space!
Factored Feature Representations

- Solution: Factor feature representations relative to the output
  - Context-free parsing:
    \[ f(x, y) = \sum_{A \rightarrow BC \in y} f(x, A \rightarrow BC) \]
  - Sequence analysis – Markov assumptions:
    \[ f(x, y) = \sum_{i=1}^{\left| y \right|} f(x, y_{i-1}, y_i) \]

- These kinds of factorizations allow us to run algorithms like CKY and Viterbi to compute the argmax function
**Example – Sequence Labeling**

- Many NLP problems can be cast in this light
  - Part-of-speech tagging
  - Named-entity extraction
  - Semantic role labeling
- Input: \( x = x_0 x_1 \ldots x_n \)
- Output: \( y = y_0 y_1 \ldots y_n \)
  - Each \( y_i \in \mathcal{Y}_{\text{atom}} \) – which is small
  - Each \( y \in \mathcal{Y} = \mathcal{Y}_{\text{atom}}^n \) – which is large
- Example: part-of-speech tagging – \( \mathcal{Y}_{\text{atom}} \) is set of tags

\[ x = \text{John} \quad \text{saw} \quad \text{Mary} \quad \text{with} \quad \text{the} \quad \text{telescope} \]
\[ y = \text{PROP} \quad \text{VERB} \quad \text{PROP} \quad \text{ADP} \quad \text{DET} \quad \text{NOUN} \]
Sequence Labeling – Output Interaction

\[ x = \text{John} \quad \text{saw} \quad \text{Mary} \quad \text{with} \quad \text{the} \quad \text{telescope} \]

\[ y = \text{PROPN} \quad \text{VERB} \quad \text{PROPN} \quad \text{ADP} \quad \text{DET} \quad \text{NOUN} \]

- Why not just make a sequence of multi-class predictions?
Sequence Labeling – Output Interaction

\[ x = \text{John} \quad \text{saw} \quad \text{Mary} \quad \text{with} \quad \text{the} \quad \text{telescope} \]

\[ y = \text{PROPN} \quad \text{VERB} \quad \text{PROPN} \quad \text{ADP} \quad \text{DET} \quad \text{NOUN} \]

- Why not just make a sequence of multi-class predictions?

- Because there are interactions between neighbouring tags
  - What tag does “saw” have?
  - What if I told you the previous tag was DET?
  - What if it was PROPN?
Sequence Labeling – Markov Factorization

\[ x = \text{John saw Mary with the telescope} \]
\[ y = \text{PROPN VERB PROPN ADP DET NOUN} \]

- Markov factorization – factor by adjacent labels
- First-order (like HMMs)

\[
\begin{align*}
  f(x, y) &= \sum_{i=1}^{\vert y \vert} f(x, y_{i-1}, y_i) \\
  f(x, y) &= \sum_{i=k}^{\vert y \vert} f(x, y_{i-k}, \ldots, y_{i-1}, y_i)
\end{align*}
\]
Sequence Labeling – Features

\[ x = \text{John} \quad \text{saw} \quad \text{Mary} \quad \text{with} \quad \text{the} \quad \text{telescope} \]

\[ y = \text{PROPN} \quad \text{VERB} \quad \text{PROPN} \quad \text{ADP} \quad \text{DET} \quad \text{NOUN} \]

▶ First-order

\[ f(x, y) = \sum_{i=1}^{\|y\|} f(x, y_{i-1}, y_i) \]

▶ \( f(x, y_{i-1}, y_i) \) is any feature of the input & two adjacent labels

\[
f_j(x, y_{i-1}, y_i) = \begin{cases} 
1 & \text{if } x_i = \text{"saw"} \\
\text{and } y_{i-1} = \text{PROPN} \text{ and } y_i = \text{VERB} \\
0 & \text{otherwise}
\end{cases}
\]

\[
f_{j'}(x, y_{i-1}, y_i) = \begin{cases} 
1 & \text{if } x_i = \text{"saw"} \\
\text{and } y_{i-1} = \text{DET} \text{ and } y_i = \text{VERB} \\
0 & \text{otherwise}
\end{cases}
\]

▶ \( w_j \) should get high weight and \( w_{j'} \) should get low weight
Sequence Labeling - Inference

How does factorization effect inference?

\[
y = \arg\max_y w \cdot f(x, y)
\]

\[
= \arg\max_y w \cdot \sum_{i=1}^{|y|} f(x, y_{i-1}, y_i)
\]

\[
= \arg\max_y \sum_{i=1}^{|y|} w \cdot f(x, y_{i-1}, y_i)
\]

\[
= \arg\max_y \sum_{i=1}^{|y|} \sum_{j=1}^m w_j \cdot f_j(x, y_{i-1}, y_i)
\]

We can use the Viterbi algorithm!
Structured Learning

- Efficient inference for factored representations
  - Viterbi for sequence labeling
  - CKY for constituency parsing
  - Spanning tree algorithms for dependency parsing

- But what about learning?
Structured Perceptron

1: \( w \leftarrow 0 \)
2: for a fixed number of iterations do
3: for all \((x, y) \in T\) do
4: \( \hat{y} = \arg\max_y w \cdot f(x, y) \)
5: if \( \hat{y} \neq y \)
6: \( w = w + f(x, y) - f(x, \hat{y}) \)
7: end if
8: end for
9: end for

Solving the argmax is tractable with factored representations!
More Structured Classifiers

- Just as for multiclass classification, we can reuse the same trick for other learning strategies and loss functions
- Minimum error:
  - Structured SVM and MIRA
- Maximum likelihood:
  - Conditional Random Fields (CRF) – structured log regression
  - Hidden Markov Models (HMM) – structured Naive Bayes
Summing Up

- **Linear models**
  - Simple yet powerful learning framework
  - Generalize to multiclass and structured prediction

- **Limitations:**
  - Every classifier defines a hyperplane (linearity)
  - Complex features have to be engineered

- **Next lecture (with Yan): May 3**