4. Linear Classification

Kai Yu
Linear Classifiers

- A simplest classification model
- Help to understand nonlinear models
- Arguably the most useful classification method!
Linear Classifiers

- A simplest classification model
- Help to understand nonlinear models
- Arguably the most useful classification method!
Outline

- Perceptron Algorithm
- Support Vector Machines
- Logistic Regression
- Summary
Basic Neuron
Perceptron Node – Threshold Logic Unit

\[
x_1 \rightarrow w_1 \\
x_2 \rightarrow w_2 \\
x_n \rightarrow w_n
\]

\[
\begin{align*}
&z = 1 \quad \text{if} \quad \sum_{i=1}^{n} x_i w_i \geq b \\
&z = 0 \quad \text{if} \quad \sum_{i=1}^{n} x_i w_i < b
\end{align*}
\]
Perceptron Learning Algorithm

\[ x_1 \rightarrow .4 \rightarrow .1 \rightarrow z \]
\[ x_2 \rightarrow -.2 \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>.8</td>
<td>.3</td>
<td>1</td>
</tr>
<tr>
<td>.4</td>
<td>.1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ z = \begin{cases} 
1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq b \\
0 & \text{if } \sum_{i=1}^{n} x_i w_i < b 
\end{cases} \]
First Training Instance

\[ net = .8 \times .4 + .3 \times - .2 = .26 \]

\[
\begin{array}{c|cc|c}
 x_1 & x_2 & t \\
 .8 & .3 & 1 \\
 .4 & .1 & 0 \\
\end{array}
\]

\[
\begin{array}{c}
 z = \begin{cases} 
 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq b \\
 0 & \text{if } \sum_{i=1}^{n} x_i w_i < b 
\end{cases}
\end{array}
\]
Second Training Instance

\[
\begin{align*}
\text{net} &= .4 \times .4 + .1 \times -.2 = .14 \\
\Delta w_i &= (t - z) \times c \times x_i
\end{align*}
\]

\[
\begin{array}{ccc}
x_1 & x_2 & t \\
.8 & .3 & 1 \\
.4 & .1 & 0
\end{array}
\]

\[
z = \begin{cases} 
1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq b \\
0 & \text{if } \sum_{i=1}^{n} x_i w_i < b
\end{cases}
\]
The Perceptron Learning Rule

\[ \Delta w_{ij} = c(t_j - z_j) x_i \]

- **Least perturbation principle**
  - Only change weights if there is an error (learning from mistakes)
  - The change amounts to \( x_i \) scaled by a small \( c \)

- Iteratively apply an example from the training set
- Each iteration through the training set is an *epoch*
- Continue training until total training error is less than *epsilon*
- **Perceptron Convergence Theorem**: Guaranteed to find a solution in finite time if a solution exists
Outline

- Perceptron Algorithm
- **Support Vector Machines**
- Logistic Regression
- Summary
Support Vector Machines: Overview

• A powerful method for 2-class classification
  • Original idea: Vapnik, 1965 for linear classifiers
  • SVM, Cortes and Vapnik, 1995
  • Becomes very hot since late 90’s

• Better generalization (less overfitting)

• Key ideas
  – Use kernel function to transform low dimensional training samples to higher dim (for linear separability problem)
  – Use quadratic programming (QP) to find the best classifier boundary hyperplane (for global optima and)
Linear Classifiers

- \( f(x, w, b) = \text{sign}(w \cdot x - b) \)

\( x \rightarrow \alpha \rightarrow f \rightarrow y_{\text{est}} \)

- denotes +1
- denotes -1

How would you classify this data?
Linear Classifiers

\[ f(x, w, b) = \text{sign}(w \cdot x - b) \]

- denotes +1
- denotes -1

How would you classify this data?
Linear Classifiers

\[ f(x, w, b) = \text{sign}(w \cdot x - b) \]

- \( f \) denotes +1
- Denotes -1

How would you classify this data?
Linear Classifiers

\[ f(x, w, b) = \text{sign}(w \cdot x - b) \]

How would you classify this data?

- denotes +1
- denotes -1
Linear Classifiers

\[
f(x, w, b) = \text{sign}(w \cdot x - b)
\]

- denotes +1
- denotes -1

Any of these would be fine..

..but which is best?
Classifier Margin

- denotes +1
- denotes -1

Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

$$f(x, w, b) = \text{sign}(w \cdot x - b)$$
The maximum margin linear classifier is the linear classifier with maximum margin.

This is the simplest kind of SVM (Called an LSVM)

\[ f(x, w, b) = \text{sign}(w \cdot x - b) \]
The maximum margin linear classifier is the linear classifier with maximum margin.

This is the simplest kind of SVM (Called an LSVM)
Computing the margin width

How do we compute $M$ in terms of $w$ and $b$?

- Plus-plane $= \{ x : w \cdot x + b = +1 \}$
- Minus-plane $= \{ x : w \cdot x + b = -1 \}$
- Margin $M = \frac{2}{\sqrt{w \cdot w}}$
Why Maximum Margin?

- denotes +1
- denotes -1

The maximum margin linear classifier is the simplest kind of SVM (Called an LSVM)

1. Intuitively this feels safest.
2. If we’ve made a small error in the location of the boundary this gives us least chance of causing a misclassification.
3. It also helps generalization
4. There’s some theory that this is a good thing.
5. Empirically it works very very well.

\[ f(x, w, b) = \text{sign}(w \cdot x - b) \]
Another way to understand max margin

- For \( f(x) = w.x + b \), one way to characterize the smoothness of \( f(x) \) is

\[
\left| \frac{\partial f(x)}{\partial x} \right| = |w|
\]

- Therefore, margin measures the smoothness of \( f(x) \).

- As a rule of thumb, machine learning prefers smooth functions: similar \( x \)'s should have similar \( y \)'s.
Learning the Maximum Margin Classifier

Given a guess of $\mathbf{w}$ and $b$ we can

- Compute whether all data points in the correct half-planes
- Compute the width of the margin

So now we just need to write a program to search the space of $\mathbf{w}'$s and $b'$s to find the widest margin that matches all the data points. How?
Learning via Quadratic Programming

- QP is a well-studied class of optimization algorithms to maximize a quadratic function of some real-valued variables subject to linear constraints.

- Minimize both $w \cdot w$ (to maximize $M$) and misclassification error.
Quadratic Programming

Find \( \arg \max_u \ c + d^T u + \frac{u^T R u}{2} \)  

Subject to 
\[
\begin{align*}
    a_{11} u_1 + a_{12} u_2 + \ldots + a_{1m} u_m &\leq b_1 \\
    a_{21} u_1 + a_{22} u_2 + \ldots + a_{2m} u_m &\leq b_2 \\
    \vdots &\quad \vdots \\
    a_{n1} u_1 + a_{n2} u_2 + \ldots + a_{nm} u_m &\leq b_n
\end{align*}
\]

And subject to 
\[
\begin{align*}
    a_{(n+1)1} u_1 + a_{(n+1)2} u_2 + \ldots + a_{(n+1)m} u_m &= b_{(n+1)} \\
    a_{(n+2)1} u_1 + a_{(n+2)2} u_2 + \ldots + a_{(n+2)m} u_m &= b_{(n+2)} \\
    \vdots &\quad \vdots \\
    a_{(n+e)1} u_1 + a_{(n+e)2} u_2 + \ldots + a_{(n+e)m} u_m &= b_{(n+e)}
\end{align*}
\]
What should our quadratic optimization criterion be?

Minimize \( \frac{1}{2} \mathbf{w} \cdot \mathbf{w} \)

What constraints should be?

\[ \mathbf{w} \cdot \mathbf{x}_k + b \geq 1 \text{ if } y_k = 1 \]
\[ \mathbf{w} \cdot \mathbf{x}_k + b \leq -1 \text{ if } y_k = -1 \]
Solving the Optimization Problem

Find $w$ and $b$ such that
$\Phi(w) = w^T w$ is minimized
and for all $(x_i, y_i), i=1..n : y_i (w^T x_i + b) \geq 1$

- Need to optimize a quadratic function subject to linear constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.
- The solution involves constructing a dual problem where a Lagrange multiplier $\alpha_i$ is associated with every inequality constraint in the primal (original) problem:

Find $\alpha_1…\alpha_n$ such that
$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j x_i^T x_j$ is maximized and
(1) $\sum \alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all $\alpha_i$
Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables* $\varepsilon_i$ can be added to allow misclassification of difficult or noisy examples, resulting so-called *soft margin*. 

![Diagram showing soft margin classification with slack variables $\varepsilon_i$.]
Learning Maximum Margin with Noise

Given guess of $w$, $b$ we can:

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume $R$ data points, each $(x_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

Minimize

$$
\frac{1}{2} w \cdot w + C \sum_{k=1}^{R} \varepsilon_k
$$

$\varepsilon_k = \text{distance of error points to their correct place}$

How many constraints will we have? $R$

What should they be?

- $w \cdot x_k + b \geq 1 - \varepsilon_k$ if $y_k = 1$
- $w \cdot x_k + b \leq -1 + \varepsilon_k$ if $y_k = -1$
- $\varepsilon_k \geq 0$ for all $k$
Soft Margin Classification Mathematically

- **The old formulation:**

  \[
  \text{Find } \mathbf{w} \text{ and } b \text{ such that } \\
  \Phi(\mathbf{w}) = \mathbf{w}^T \mathbf{w} \text{ is minimized} \text{ and for all } (x_i, y_i), i=1..n : \\
  y_i (\mathbf{w}^T x_i + b) \geq 1
  \]

- **Modified formulation incorporates slack variables:**

  \[
  \text{Find } \mathbf{w} \text{ and } b \text{ such that } \\
  \Phi(\mathbf{w}) = \mathbf{w}^T \mathbf{w} + C \Sigma \xi_i \text{ is minimized} \text{ and for all } (x_i, y_i), i=1..n : \\
  y_i (\mathbf{w}^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0
  \]

- Parameter $C$ can be viewed as a way to control overfitting: it “trades off” the relative importance of maximizing the margin and fitting the training data.
Hinge loss

- The soft margin SVM is equivalent to applying a hinge loss

\[ L(w, b) := \sum_{i=1}^{n} \max(1 - y_i(w^T x_i + b), 0) \]

- Equivalent unconstrained optimization formulation

\[ \min_{\{w, b\}} L(w, b) + \lambda||w||^2 \quad \lambda=0.5/C \]
Outline

- Perceptron Algorithm
- Support Vector Machines
- Logistic Regression
- Summary
Logistic Regression

- Binary response: \( Y = \{+1, -1\} \)

\[
Y_i | X_i \sim \text{Bernoulli}(p_i)
\]

where \( p_i \) is the probability of \( Y_i = 1 \)

\[
p_i = \frac{1}{1 + \exp(-W^T X_i)}
\]

- Likelihood

\[
\prod_{i=1}^{n} P(Y_i | X_i) = \prod_{i=1}^{n} \left( \frac{1}{1 + \exp(-Y_i X_i^T W)} \right)
\]
Logistic regression

- Maximum likelihood estimator (MLE) becomes logistic regression

\[
\min_W \sum_{i=1}^{n} - \ln p(Y_i \lvert X_i) = \sum_{i=1}^{n} \ln(1 + \exp(Y_i X_i^T W))
\]

- Convex optimization problem in terms of \( W \)

- MAP is regularized logistic regression

\[
\min_W \sum_{i=1}^{n} \ln(1 + \exp(Y_i X_i^T W)) + \lambda \| W \|^2
\]
Outline

- Perceptron Algorithm
- Support Vector Machines
- Logistic Regression
- Summary
General formulation of linear classifiers

\[ \min_{\{w,b\}} L(w,b) + \lambda \|w\|^2 \]

- The objective: empirical loss + regularization

- The regularization term is usually \(L2\) norm, but also often \(L1\) norm for sparse models

- The empirical loss can be hinge loss, logistic loss, smooth hinge loss, … or your own invention
Different loss functions

![Graph showing different loss functions]

- classification error
- SVM
- logistic regression
- least squares
- exponential
- modified huber
Comments on linear classifiers

- Choosing logistic regression or SVM?
  - Not that big different!
  - Logistic regression outputs probabilities.

- Smooth loss functions, e.g. logistic
  - Easier to optimize (LBFGS …)
  - Hinge → Differentiable hinge, then you can easily have your own implementation of SVMs

- Try different loss functions and regularization terms
  - Depend on data, e.g., many outliers? Irrelevant features? structure in output?
Linear classifiers in practice and research

- **Linear classifiers are** simple and scalable
  - Training complexity is linear or sub-linear w.r.t. data size
  - Classification is simple to implement
  - State-of-the-art for texts, images, …

- **Their success depends on** quality of features
  - A useful practice: use a lot of features, learn sparse w

- **Hot topic**: large-scale linear classification
  - Many data, many features, many classes
  - Stochastic optimization
  - Parallel implementation