Machine Learning
ITCS 6156/8156

The Perceptron Algorithm

The Kernel Trick

Razvan C. Bunescu
Department of Computer Science @ CCI

razvan.bunescu@uncc.edu
Linear Discriminants Classification

- Use a linear function of the input vector:
  \[ h(x) = w^T \varphi(x) + w_0 \]

- Decision:
  \[ x \in C_1 \text{ if } h(x) \geq 0, \text{ otherwise } x \in C_2. \]
  \[ \Rightarrow \text{ decision boundary is hyperplane } h(x) = 0. \]

- Properties:
  - \( w \) is orthogonal to vectors lying within the decision surface.
  - \( w_0 \) controls the location of the decision hyperplane.
Geometric Interpretation

\[ h > 0 \]
\[ h = 0 \]
\[ h < 0 \]

\[ \mathcal{R}_1 \]
\[ \mathcal{R}_2 \]

\[ \mathbf{x} \]
\[ \mathbf{w} \]
\[ \mathbf{x}_{\perp} \]

\[ \frac{h(x)}{||\mathbf{w}||} \]

\[ -\frac{w_0}{||\mathbf{w}||} \]
Linear Discriminant Classification: Two Classes (K = 2)

- What algorithms can be used to learn \( y(x) = w^T \varphi(x) + w_0 \)?
  Assume a training dataset of \( N = N_1 + N_2 \) examples in \( C_1 \) and \( C_2 \).

  - Perceptron:
    - Voted/Averaged Perceptron
    - Kernel Perceptron
  - Support Vector Machines:
    - Linear
    - Kernel
  - Fisher’s Linear Discriminant
Linear Discriminant Classification

- Assume classes \( T = \{c_1, c_2\} = \{1, -1\} \).
- Training set is \((x_1, t_1), (x_2, t_2), \ldots (x_n, t_n)\).
\[
x = [1, x_1, x_2, \ldots, x_k]^T
\]
\[
\hat{t}(x) = sgn(w^Tx) = sgn(w_0 + w_1 x_1 + \ldots + w_k x_k)
\]

A linear discriminant function
Linear Discriminant Classification: Objective Function

- Learning = finding the “right” parameters \( \mathbf{w}^T = [w_0, w_1, \ldots, w_k] \)
  - Find \( \mathbf{w} \) that minimizes an error function \( E(\mathbf{w}) \) which measures the misfit between \( h(x_n, \mathbf{w}) \) and \( t_n \).
  - Expect that \( h(x, \mathbf{w}) \) performing well on training examples \( x_n \Rightarrow h(x, \mathbf{w}) \) will perform well on arbitrary test examples \( x \in X \).

- **Least Squares** error function?

\[
E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{h(x_n, \mathbf{w}) - t_n\}^2
\]

2 x # of mistakes
Least Squares vs. Perceptron Criterion

- **Least Squares** => cost is # of misclassified patterns:
  - Piecewise constant function of \( \mathbf{w} \) with discontinuities.
  - Cannot find closed form solution for \( \mathbf{w} \) that minimizes cost.
  - Cannot use gradient methods (gradient zero almost everywhere).

- **Perceptron Criterion**:
  - Set labels to be +1 and \(-1\). Want \( \mathbf{w}^T \mathbf{x}_n > 0 \) for \( t_n = 1 \), and \( \mathbf{w}^T \mathbf{x}_n < 0 \) for \( t_n = -1 \).
    \[ \Rightarrow \] would like to have \( \mathbf{w}^T \mathbf{x}_n t_n > 0 \) for all patterns.
    \[ \Rightarrow \] want to minimize \(-\mathbf{w}^T \mathbf{x}_n t_n\) for all misclassified patterns \( M \).

\[ \Rightarrow \text{minimize } E_p(\mathbf{w}) = -\sum_{n \in M} \mathbf{w}^T \mathbf{x}_n t_n \]
Stochastic Gradient Descent

- **Perceptron Criterion:**
  \[
  \text{minimize } E_P(w) = -\sum_{n\in M} w^T x_n t_n
  \]

- Update parameters \( w \) sequentially after each mistake:
  \[
  w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E_P(w^{(\tau)}, x_n)
  = w^{(\tau)} + \eta x_n t_n
  \]

- The magnitude of \( w \) is inconsequential \( \Rightarrow \) set \( \eta = 1 \).
  \[
  w^{(\tau+1)} = w^{(\tau)} + x_n t_n
  \]
The Perceptron Algorithm: Two Classes

1. **initialize** parameters $\mathbf{w} = 0$
2. **for** $n = 1 \ldots N$
3.   \[ h_n = \text{sgn}(\mathbf{w}^T \mathbf{x}_n) \]
4.   **if** $h_n \neq t_n$ **then**
5.   \[ \mathbf{w} = \mathbf{w} + t_n \mathbf{x}_n \]

Repeat:
   a) until convergence.
   b) for a number of epochs $E$.

Theorem [Rosenblatt, 1962]:
If the training dataset is linearly separable, the perceptron learning algorithm is guaranteed to find a solution in a finite number of steps.
• see Theorem 1 (Block, Novikoff) in [Freund & Schapire, 1999].

$\text{sgn}(z) = +1$ if $z > 0$,
     $0$ if $z = 0$,
    $-1$ if $z < 0$
The Perceptron Algorithm: Two Classes

1. initialize parameters $w = 0$
2. for $n = 1 \ldots N$
3. $h_n = w^T x_n$
4. if $h_n t_n \leq 0$ then
5. $w = w + t_n x_n$

Repeat:
- a) until convergence.
- b) for a number of epochs $E$.

Theorem [Rosenblatt, 1962]:
If the training dataset is linearly separable, the perceptron learning algorithm is guaranteed to find a solution in a finite number of steps.
- see Theorem 1 (Block, Novikoff) in [Freund & Schapire, 1999].
The Perceptron Algorithm: Two Classes

1. **initialize** parameters $w = 0$
2. **for** $n = 1 \ldots N$
   
   3. $h_n = w^T x_n$
   4. if $h_n \geq 0$ and $t_n = -1$
   5. $w = w - x_n$
   6. if $h_n \leq 0$ and $t_n = +1$
   7. $w = w + x_n$

Repeat:

a) until convergence.
b) for a number of epochs $E$.

$s_{gn}(z) = +1$ if $z > 0$,
     0 if $z = 0$,
     $-1$ if $z < 0$

What is the impact of the perceptron update on the score $w^T x_n$ of the misclassified example $x_n$?
Linear vs. Non-linear Decision Boundaries

And

Or

Xor

\[ \varphi(x) = [1, x_1, x_2]^T \]
\[ w = [w_0, w_1, w_2]^T \]  \[ \Rightarrow w^T \varphi(x) = [w_1, w_2]^T [x_1, x_2] + w_0 \]
How to Find Non-linear Decision Boundaries

1) Perceptron with manually engineered features:
   – Quadratic features.

2) Kernel methods (e.g. SVMs) with non-linear kernels:
   – Quadratic kernels, Gaussian kernels.

3) Self-supervised feature learning (e.g. auto-encoders):
   – Plug learned features in any linear classifier.

4) Neural Networks with one or more hidden layers:
   – Automatically learned features.
Non-Linear Classification: XOR Dataset

\[ x = [x_1, x_2] \]
1) Manually Engineered Features: Add $x_1 x_2$

$x = [x_1, x_2, x_1 x_2]$
Logistic Regression with Manually Engineered Features

\[ \mathbf{x} = [x_1, x_2, x_1x_2] \]
Perceptron with Manually Engineered Features

Project $\mathbf{x} = [x_1, x_2, x_1 x_2]$ and decision hyperplane back to $\mathbf{x} = [x_1, x_2]$
Averaged Perceptron: Two Classes

1. **initialize** parameters $\mathbf{w} = 0$, $\tau = 1$, $\overline{\mathbf{w}} = 0$
2. **for** $n = 1 \ldots N$
3. $h_n = \text{sgn} (\mathbf{w}^T \mathbf{x}_n)$
4. **if** $h_n \neq t_n$ **then**
5. $\mathbf{w} = \mathbf{w} + t_n \mathbf{x}_n$
6. $\overline{\mathbf{w}} = \overline{\mathbf{w}} + \mathbf{w}$
7. $\tau = \tau + 1$
8. **return** $\overline{\mathbf{w}} / \tau$

During testing: $h(\mathbf{x}) = \text{sgn}(\overline{\mathbf{w}}^T \mathbf{x})$

$s\text{gn}(z) = +1$ if $z > 0$,
$0$ if $z = 0$,
$-1$ if $z < 0$

Repeat:
- a) until convergence.
- b) for a number of epochs $E$. 
2) Kernel Methods with Non-Linear Kernels

- Perceptrons, SVMs can be ‘kernelized’:
  1. Re-write the algorithm such that during training and testing feature vectors \( x, y \) appear only in dot-products \( x^T y \).
  2. Replace dot-products \( x^T y \) with non-linear kernels \( K(x, y) \):
     - \( K \) is a kernel if and only if \( \exists \phi \) such that \( K(x, y) = \phi(x)^T \phi(y) \)
       - \( \phi \) can be in a much higher dimensional space.
         » e.g. combinations of up to \( k \) original features
       - \( \phi(x)^T \phi(y) \) can be computed efficiently without enumerating \( \phi(x) \) or \( \phi(y) \).
The Perceptron Representer Theorem

1. **initialize** parameters $w = 0$
2. for $n = 1 \ldots N$
3. $h_n = sgn(w^T x_n)$
4. if $h_n \neq t_n$ then
5. $w = w + t_n x_n$

Repeat:
- a) until convergence.
- b) for a number of epochs $E$.

Loop invariant: $w$ is a weighted sum of training vectors:

$$w = \sum_{n=1..N} \alpha_n t_n x_n \Rightarrow w^T x = \sum_{n=1..N} \alpha_n t_n x_n x^T$$
Kernel Perceptron: Two Classes

1. define \( f(x) = \mathbf{w}^T \mathbf{x} = \sum_{j=1..N} \alpha_j t_j \mathbf{x}_j^T \mathbf{x} = \sum_{j=1..N} \alpha_j t_j K(\mathbf{x}_j, \mathbf{x}) \)

2. initialize dual parameters \( \alpha_n = 0 \)

3. for \( n = 1 \ldots N \)

4. \( h_n = \text{sgn} \, f(\mathbf{x}_n) \)

5. if \( h_n \neq t_n \) then

6. \( \alpha_n = \alpha_n + 1 \)

Repeat:
- a) until convergence.
- b) for a number of epochs \( E \).

During testing: \( h(\mathbf{x}) = \text{sgn} \, f(\mathbf{x}) \)
Kernel Perceptron: Two Classes

1. define \( f(x) = w^T x = \sum_{j=1}^{N} \alpha_j t_j x_j^T x = \sum_{j=1}^{N} \alpha_j t_j K(x_j, x) \)
2. initialize dual parameters \( \alpha_n = 0 \)
3. for \( n = 1 \ldots N \)
4. \( h_n = sgn f(x_n) \)
5. if \( h_n \neq t_n \) then
6. \( \alpha_n = \alpha_n + 1 \)

Let \( S = \{ j | \alpha_j \neq 0 \} \) be the set of support vectors. Then \( f(x) = \sum_{j \in S} \alpha_j t_j K(x_j, x) \)

During testing: \( h(x) = sgn f(x) \)
Kernel Perceptron: Equivalent Formulation

1. define \( f(x) = w^T x = \sum_j \alpha_j x_j^T x = \sum_j \alpha_j K(x_j, x) \)
2. initialize dual parameters \( \alpha_n = 0 \)
3. for \( n = 1 \ldots N \)
4. \( h_n = \text{sgn} f(x_n) \)
5. if \( h_n \neq t_n \) then
6. \( \alpha_n = \alpha_n + t_n \)

Repeat:
- a) until convergence.
- b) for a number of epochs E.

During testing: \( h(x) = \text{sgn} f(x) \)
The Perceptron vs. Boolean Functions

\[ \varphi(x) = [1, x_1, x_2]^T \]
\[ w = [w_0, w_1, w_2]^T \]
\[ \Rightarrow w^T \varphi(x) = [w_1, w_2]^T [x_1, x_2] + w_0 \]
Perceptron with Quadratic Kernel

• Discriminant function:

\[ f(x) = \sum_{i} \alpha_i t_i \varphi(x_i)^T \varphi(x) = \sum_{i} \alpha_i t_i K(x_i, x) \]

• Quadratic kernel:

\[ K(x, y) = (x^T y)^2 = (x_1 y_1 + x_2 y_2)^2 \]

\[ \Rightarrow \] corresponding feature space \( \varphi(x) = ? \)

conjunctions of two atomic features
Perceptron with Quadratic Kernel

Linear kernel \( K(x, y) = x^T y \)

Quadratic kernel \( K(x, y) = (x^T y)^2 \)
Quadratic Kernels

- Circles, hyperbolas, and ellipses as separating surfaces:

\[ K(x, y) = (1 + x^T y)^2 = \varphi(x)^T \varphi(y) \]

\[ \varphi(x) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2]^T \]
Quadratic Kernels

\[ K(x, y) = (x^T y)^2 = \varphi(x)^T \varphi(y) \]
Explicit Features vs. Kernels

- Explicitly enumerating features can be prohibitive:
  - 1,000 basic features for $\mathbf{x}^T\mathbf{y} \Rightarrow 500,500$ quadratic features for $(\mathbf{x}^T\mathbf{y})^2$
  - Much worse for higher order features.

- **Solution:**
  - Do not compute the feature vectors, compute kernels instead (i.e. compute dot products between implicit feature vectors).
    - $(\mathbf{x}^T\mathbf{y})^2$ takes 1001 multiplications.
    - $\varphi(\mathbf{x})^T \varphi(\mathbf{y})$ in feature space takes 500,500 multiplications.
Kernel Functions

• **Definition:**
  
  A function \( k : X \times X \to \mathbb{R} \) is a kernel function if there exists a feature mapping \( \varphi : X \to \mathbb{R}^n \) such that:
  
  \[
  k(x, y) = \varphi(x)^T \varphi(y)
  \]

• **Theorem:**
  
  \( k : X \times X \to \mathbb{R} \) is a valid kernel \( \iff \) the Gram matrix \( K \) whose elements are given by \( k(x_n, x_m) \) is positive semidefinite for all possible choices of the set \( \{x_n\} \).
Kernel Examples

- **Linear kernel**: $K(x, y) = x^T y$

- **Quadratic kernel**: $K(x, y) = (c + x^T y)^2$
  - contains constant, linear terms and terms of order two ($c > 0$).

- **Polynomial kernel**: $K(x, y) = (c + x^T y)^M$
  - contains all terms up to degree $M$ ($c > 0$).

- **Gaussian kernel**: $K(x, y) = \exp(-\|x - y\|^2 / 2\sigma^2)$
  - Corresponding feature space has infinite dimensionality.
  - Prove using Taylor expansion of exponential.

  $\phi(x) = e^{-\gamma x^2} [1, \sqrt{2\gamma x}, \sqrt{2\gamma x^2}, ...]$
Techniques for Constructing Kernels

Given valid kernels $k_1(x, x')$ and $k_2(x, x')$, the following new kernels will also be valid:

\[
\begin{align*}
    k(x, x') & = c k_1(x, x') \\
    k(x, x') & = f(x) k_1(x, x') f(x') \\
    k(x, x') & = q(k_1(x, x')) \\
    k(x, x') & = \exp(k_1(x, x')) \\
    k(x, x') & = k_1(x, x') + k_2(x, x') \\
    k(x, x') & = k_1(x, x') k_2(x, x') \\
    k(x, x') & = k_3(\phi(x), \phi(x')) \\
    k(x, x') & = x^T A x' \\
    k(x, x') & = k_a(x, x') + k_b(x, x') \\
    k(x, x') & = k_a(x, x') k_b(x, x')
\end{align*}
\]

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\phi(x)$ is a function from $x$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $A$ is a symmetric positive semidefinite matrix, $x_a$ and $x_b$ are variables (not necessarily disjoint) with $x = (x_a, x_b)$, and $k_a$ and $k_b$ are valid kernel functions over their respective spaces.
Kernels over Discrete Structures

- **Subsequence Kernels** [Lodhi et al., JMLR 2002]:
  - \( \Sigma \) is a finite alphabet (set of symbols).
  - \( x, y \in \Sigma^* \) are two sequences of symbols with lengths \( |x| \) and \( |y| \).
  - \( k(x, y) \) is defined as the number of common substrings of length \( n \).
  - \( k(x, y) \) can be computed in \( O(n|x||y|) \) time complexity.

- **Tree Kernels** [Collins and Duffy, NIPS 2001]:
  - \( T_1 \) and \( T_2 \) are two trees with \( N_1 \) and \( N_2 \) nodes respectively.
  - \( k(T_1, T_2) \) is defined as the number of common subtrees.
  - \( k(T_1, T_2) \) can be computed in \( O(N_1N_2) \) time complexity.
  - in practice, time is linear in the size of the trees.
Supplementary Reading

• PRML Chapter 6:
  – Section 6.1 on dual representations for linear regression models.
  – Section 6.2 on techniques for constructing new kernels.
Linear Discriminant Functions:
Multiple Classes (K > 2)

1) Train K or K−1 one-versus-the-rest binary classifiers.
2) Train K(K−1)/2 one-versus-one binary classifiers.

3) Train K linear functions:
\[ y_k(x) = w_k^T \varphi(x) + w_{k0} \]

• Decision:
\[ x \in C_k \text{ if } y_k(x) > y_j(x), \text{ for all } j \neq k. \]
⇒ decision boundary between classes \( C_k \) and \( C_j \) is hyperplane defined by \( y_k(x) = y_j(x) \) i.e. \( (w_k - w_j)^T \varphi(x) + (w_{k0} - w_{j0}) = 0 \)
⇒ same geometrical properties as in binary case.
Linear Discriminant Functions: Multiple Classes (K > 2)

4) More general ranking approach:

\[ y(x) = \arg \max_{t \in T} w^T \varphi(x, t) \quad \text{where} \quad T = \{c_1, c_2, \ldots, c_K\} \]

- It subsumes the approach with K separate linear functions.
- Useful when T is very large (e.g. exponential in the size of input x), assuming inference can be done efficiently.
The Perceptron Algorithm: K classes

1. **initialize** parameters $w = 0$
2. **for** $i = 1 \ldots n$
3. $y_i = \arg \max_{t \in T} w^T \varphi(x_i, t)$
4. **if** $y_i \neq t_i$ **then**
5. $w = w + \varphi(x_i, t_i) - \varphi(x_i, y_i)$

During testing:
\[ t^* = \arg \max_{t \in T} w^T \varphi(x, t) \]
Averaged Perceptron: K classes

1. **initialize** parameters $\mathbf{w} = 0, \tau = 1, \overline{\mathbf{w}} = 0$
2. **for** $i = 1 \ldots n$
3. \quad $y_i = \arg \max_{t \in T} \mathbf{w}^T \varphi(\mathbf{x}_i, t)$
4. \quad **if** $y_i \neq t_i$ **then**
5. \quad \quad $\mathbf{w} = \mathbf{w} + \varphi(\mathbf{x}_i, t_i) - \varphi(\mathbf{x}_i, y_i)$
6. \quad $\overline{\mathbf{w}} = \overline{\mathbf{w}} + \mathbf{w}$
7. \quad $\tau = \tau + 1$
8. **return** $\overline{\mathbf{w}} / \tau$

Repeat:
\[
\text{a) until convergence.} \\
\text{b) for a number of epochs E.}
\]

During testing: \( t^* = \arg \max_{t \in T} \overline{\mathbf{w}}^T \varphi(\mathbf{x}, t) \)
The Perceptron Algorithm: K classes

1. **initialize** parameters $w = 0$
2. for $i = 1 \ldots n$
3. \[ c_j = \arg \max_{t \in T} w^T \phi(x_i, t) \]
4. if $c_j \neq t_i$ then
5. \[ w = w + \phi(x_i, t_i) - \phi(x_i, c_j) \]

Repeat:
- a) until convergence.
- b) for a number of epochs E.

Loop invariant: $w$ is a weighted sum of training vectors:
\[
\begin{align*}
    w &= \sum_{i,j} \alpha_{ij} (\phi(x_i, t_i) - \phi(x_i, c_j)) \\
    \Rightarrow w^T \phi(x, t) &= \sum_{i,j} \alpha_{ij} (\phi(x_i, t_i)^T \phi(x, t) - \phi(x_i, c_j)^T \phi(x, t))
\end{align*}
\]
Kernel Perceptron: K classes

1. define \( f(x, t) = \sum_{i,j} \alpha_{ij} (\phi(x_i, t_i)^T \phi(x, t) - \phi(x_i, c_j)^T \phi(x, t)) \)
2. initialize dual parameters \( \alpha_{ij} = 0 \)
3. for \( i = 1 \ldots n \)
4. \( c_j = \text{arg max}_{t \in T} f(x_i, t) \)
5. if \( y_i \neq t_i \) then
6. \( \alpha_{ij} = \alpha_{ij} + 1 \)

Repeat:
- a) until convergence.
- b) for a number of epochs \( E \).

During testing:
\[ t^* = \text{arg max}_{t \in T} f(x, t) \]
Kernel Perceptron: K classes

- Discriminant function:

\[ f(x,t) = \sum_{i,j} \alpha_{i,j} (\phi(x_i, t_i)^T \phi(x,t) - \phi(x_i, c_j)^T \phi(x,t)) \]

\[ = \sum_{i,j} \alpha_{ij} (K(x_i, t_i, x, t) - K(x_i, c_j, x, t)) \]

where:

\[ K(x_i, t_i, x, t) = \varphi^T(x_i, t_i) \varphi(x, t) \]

\[ K(x_i, y_i, x, t) = \varphi^T(x_i, y_i) \varphi(x, t) \]