Applied Machine Learning

Perceptron and Support Vector Machines

Siamak Ravanbakhsh

COMP 551 (winter 2020)
Learning objectives

- geometry of linear classification
- Perceptron learning algorithm
- margin maximization and support vectors
- hinge loss and relation to logistic regression
Perceptron

old implementation (1960's)

historically a significant algorithm
(first neural network, or rather just a neuron)

biologically motivated model
simple learning algorithm
convergence proof
beginning of connectionist AI

it's criticism in the book "Perceptrons" was a factor in AI winter
Perceptron

old implementation (1960's)

historically a significant algorithm
(first neural network, or rather just a neuron)
biologically motivated model
simple learning algorithm
convergence proof
beginning of connectionist AI
it's criticism in the book "Perceptrons" was a factor in AI winter

Model

\[ f(x) = \text{sign}(w^\top x + w_0) \]
Perceptron

historically a significant algorithm
(first neural network, or rather just a neuron)
biologically motivated model
simple learning algorithm
convergence proof
beginning of connectionist AI
it's criticism in the book "Perceptrons" was a factor in AI winter

Model

\[ f(x) = \text{sign}(w^\top x + w_0) \]

note that we're using +1/-1 for labels rather than 0/1.
geometry of the **separating hyperplane**

dis this hyperplane has one dimension lower than \( D \) (number of features)
geometry of the **separating hyperplane**

This hyperplane has one dimension lower than \( D \) (number of features)

For any two points \( \mathbf{a} \) and \( \mathbf{b} \) on the line

\[
\mathbf{w}^\top (\mathbf{a} - \mathbf{b}) + w_0 - w_0 = 0
\]
geometry of the **separating hyperplane**

this hyperplane has one dimension lower than $D$ (number of features)

for any two points $a$ and $b$ on the line $w^\top(a - b) + w_0 - w_0 = 0$

so $\frac{w}{||w||}$ is the unit normal vector to the line

$$y > 0$$

$$y = w^\top x + w_0 = w_2x_2 + w_1x_1 + w_0 = 0$$
geometry of the **separating hyperplane**

this hyperplane has one dimension lower than \( D \) (number of features)

for any two points \( a \) and \( b \) on the line

\[
w^T (a - b) + w_0 - w_0 = 0
\]

so \( \frac{w}{||w||} \) is the unit normal vector to the line

the orthogonal component of any point on the line

\[
\frac{w^T}{||w||} b = - \frac{w_0}{||w||}
\]

\[
y = w^T x + w_0 = w_2 x_2 + w_1 x_1 + w_0 = 0
\]
geometry of the separating hyperplane

the orthogonal component of any point on the line $\frac{w^+}{||w||} b = -\frac{w_0}{||w||}$
geometry of the separating hyperplane

the orthogonal component of any point on the line $\frac{w^+}{\|w\|} b = -\frac{w_0}{\|w\|}$
The orthogonal component of any point on the line \( \frac{w^\top}{\|w\|} b = - \frac{w_0}{\|w\|} \) is the separating hyperplane.

Signed distance of any point \( c \) from the line \( c \perp c \perp \).
geometry of the **separating hyperplane**

the orthogonal component of any point on the line $\frac{w^\top}{\|w\|} b = -\frac{w_0}{\|w\|}$

signed distance of any point $(c)$ from the line
the orthogonal component of any point on the line \( \frac{w}{||w||} b = -\frac{w_0}{||w||} \)

signed distance of any point \((c)\) from the line

geometry of the **separating hyperplane**
geometry of the separating hyperplane

the orthogonal component of any point on the line \( \frac{w^\top}{\|w\|} b = - \frac{w_0}{\|w\|} \)

signed distance of any point \( (c) \) from the line

\[
\frac{w^\top}{\|w\|} c - \frac{w^\top}{\|w\|} c_\perp
\]
the orthogonal component of any point on the line \( \frac{w^T}{||w||} b = -\frac{w_0}{||w||} \)

signed distance of any point \( (c) \) from the line

\[
\frac{w^T}{||w||} c - \frac{w^T}{||w||} c_{\perp} = \frac{1}{||w||} (w^T c + w_0)
\]
Perceptron: objective

if \( y^{(n)} \hat{y}^{(n)} < 0 \) try to make it positive

label and prediction have different signs

distance to the boundary
this is positive for points that are on the wrong side
Perceptron: objective

If \( y^{(n)} \hat{y}^{(n)} < 0 \) try to make it positive

Label and prediction have different signs

equivalent to minimizing

\[-y^{(n)}(w^\top x^{(n)} + w_0)\]

distance to the boundary
this is positive for points that are on the wrong side
Perceptron: **objective**

\[ x_1^2 x_2 \]

\[ (w^T x^{(n)} + w_0) \]

if \( y^{(n)} y^{(n)} < 0 \) try to make it positive

\[ y^{(n)} (w^T x^{(n)} + w_0) \]

equivalent to minimizing distance to the boundary this is positive for points that are on the wrong side
Perceptron: objective

If $y^{(n)} y^{(n)} < 0$ try to make it positive

Label and prediction have different signs

Equivalent to minimizing $-y^{(n)} (w^\top x^{(n)} + w_0)$

distance to the boundary
this is positive for points that are on the wrong side

So perceptron tries to minimize the distance of misclassified points from the decision boundary and push them to the right side
revisiting **Perceptron: optimization**

if $y^{(n)} \hat{y}^{(n)} < 0$ minimize $J_n(w) = -y^{(n)}(w^T x^{(n)})$
revisiting Perceptron: optimization

if \( y^{(n)} \hat{y}^{(n)} < 0 \) minimize \( J_n(w) = -y^{(n)}(w^\top x^{(n)}) \)

now we included bias in \( w \)
revisiting **Perceptron: optimization**

if $y^{(n)} \hat{y}^{(n)} < 0$ minimize $J_n(w) = -y^{(n)} (w^\top x^{(n)})$

otherwise, do nothing
revisiting **Perceptron:** optimization

if $y^{(n)} \hat{y}^{(n)} < 0$ minimize $J_n(w) = -y^{(n)}(w^\top x^{(n)})$

otherwise, do nothing

use stochastic gradient descent $\nabla J_n(w) = -y^{(n)}x^{(n)}$

$$w^{t+1} \leftarrow w^{t} - \alpha \nabla J_n(w) = w^{t} + \alpha y^{(n)}x^{(n)}$$
revisiting **Perceptron: optimization**

if \( y^{(n)} \hat{y}^{(n)} < 0 \) minimize \( J_n(w) = -y^{(n)}(w^\top x^{(n)}) \)

otherwise, do nothing

use stochastic gradient descent \( \nabla J_n(w) = -y^{(n)}x^{(n)} \)

\[
\begin{align*}
    w^{\{t+1\}} & \leftarrow w^{\{t\}} - \alpha \nabla J_n(w) = w^{\{t\}} + \alpha y^{(n)}x^{(n)} \\
\end{align*}
\]

Perceptron uses learning rate of 1
this is okay because scaling \( w \) does not affect prediction

\[
\text{sign}(w^\top x) = \text{sign}(\alpha w^\top x)
\]
revisiting **Perceptron:** optimization

if \( y^{(n)} \hat{y}^{(n)} < 0 \) minimize \( J_n(w) = -y^{(n)}(w^\top x^{(n)}) \)

otherwise, do nothing

use stochastic gradient descent \( \nabla J_n(w) = -y^{(n)}x^{(n)} \)

\[
\begin{align*}
 w^{\{t+1\}} & \leftarrow w^{\{t\}} - \alpha \nabla J_n(w) = w^{\{t\}} + \alpha y^{(n)}x^{(n)} \\
\end{align*}
\]

Perceptron uses learning rate of 1
this is okay because scaling \( w \) does not affect prediction

\[
\text{sign}(w^\top x) = \text{sign}(\alpha w^\top x)
\]

**Perceptron convergence theorem**

the algorithm is guaranteed to converge in finite steps if linearly separable
Perceptron: example

Iris dataset
(linearly separable case)

iteration 1
Perceptron: example

Iris dataset
(linearly separable case)

iteration 1

```
def Perceptron(X, y, max_iters):
    N,D = X.shape
    w = np.random.rand(D)
    for t in range(max_iters):
        n = np.random.randint(N)
        yh = np.sign(np.dot(X[n,:], w))
        if yh != y[n]:
            w = w + y[n]*X[n,:]
    return w
```
Perceptron: example

Iris dataset
(linearly separable case)

iteration 1

```
1 def Perceptron(X, y, max_iters):
2     N,D = X.shape
3     w = np.random.rand(D)
4     for t in range(max_iters):
5         n = np.random.randint(N)
6         yh = np.sign(np.dot(X[n,:], w))
7         if yh != y[n]:
8             w = w + y[n]*X[n,:]  
9     return w
```

note that the code is not checking for convergence
Perceptron: example

Iris dataset (linearly separable case)

iteration 1

\[ w^\top x = 0 \]

```python
import numpy as np

def Perceptron(X, y, max_iters):
    N, D = X.shape
    w = np.random.rand(D)
    for t in range(max_iters):
        n = np.random.randint(N)
        yh = np.sign(np.dot(X[n, :], w))
        if yh != y[n]:
            w = w + y[n] * X[n, :]
    return w
```

note that the code is not checking for convergence
Perceptron: example

Iris dataset (linearly separable case)

iteration 10

```python
1 def Perceptron(X, y, max_iters):
2     N, D = X.shape
3     w = np.random.rand(D)
4     for t in range(max_iters):
5         n = np.random.randint(N)
6         yh = np.sign(np.dot(X[n, :], w))
7         if yh != y[n]:
8             w = w + y[n] * X[n, :]
9     return w
```

Note that the code is not checking for convergence.
Perceptron: example

Iris dataset
(linearly separable case)

iteration 10

```python
def Perceptron(X, y, max_iters):
    N, D = X.shape
    w = np.random.rand(D)
    for t in range(max_iters):
        n = np.random.randint(N)
        yh = np.sign(np.dot(X[n, :], w))
        if yh != y[n]:
            w = w + y[n]*X[n, :]
    return w
```

note that the code is not checking for convergence
Perceptron: example

Iris dataset (linearly separable case)

iteration 10

```python
def Perceptron(X, y, max_iters):
    N, D = X.shape
    w = np.random.rand(D)
    for t in range(max_iters):
        n = np.random.randint(N)
        yh = np.sign(np.dot(X[n, :], w))
        if yh != y[n]:
            w = w + y[n]*X[n, :]
    return w
```

note that the code is not checking for convergence

observations:

after finding a linear separator no further updates happen
the final boundary depends on the order of instances (different from all previous methods)
Perceptron: example

```python
1  def Perceptron(X, y, max_iters):
2      N,D = X.shape
3      w = np.random.rand(D)
4      for t in range(max_iters):
5          n = np.random.randint(N)
6          yh = np.sign(np.dot(X[n,:], w))
7          if yh != y[n]:
8              w = w + y[n]*X[n,:]
9      return w
```

note that the code is not checking for convergence
Perceptron: \textbf{example}

Iris dataset
\textbf{(NOT linearly separable case)}

\begin{verbatim}
def Perceptron(X, y, max_iters):
    N,D = X.shape
    w = np.random.rand(D)
    for t in range(max_iters):
        n = np.random.randint(N)
        yh = np.sign(np.dot(X[n,:], w))
        if yh != y[n]:
            w = w + y[n]*X[n,:]
    return w
\end{verbatim}

note that the code is not checking for convergence
Perceptron: example

Iris dataset (NOT linearly separable case)

The algorithm does not converge
there is always a wrong prediction and the weights will be updated

```python
def Perceptron(X, y, max_iters):
    N, D = X.shape
    w = np.random.rand(D)
    for t in range(max_iters):
        n = np.random.randint(N)
        yh = np.sign(np.dot(X[n, :], w))
        if yh != y[n]:
            w = w + y[n] * X[n, :]
    return w
```

Note that the code is not checking for convergence.
Perceptron: issues

cyclic updates if the data is not linearly separable?

- try make the data separable using additional features?
- data may be inherently noisy
Perceptron: issues

cyclic updates if the data is not linearly separable?
  • try make the data separable using additional features?
  • data may be inherently noisy

even if linearly separable
convergence could take many iterations
Perceptron: issues

cyclic updates if the data is not linearly separable?
- try make the data separable using additional features?
- data may be inherently noisy

even if linearly separable
convergence could take many iterations

the decision boundary may be suboptimal
Perceptron: issues

cyclic updates if the data is not linearly separable?

- try make the data separable using additional features?
- data may be inherently noisy

even if linearly separable
convergence could take many iterations

the decision boundary may be suboptimal

let's fix this problem first
assume linear separability
the **margin** of a classifier (assuming correct classification)
is the distance of the closest point to the decision boundary

this is positive for correctly classified points
the **margin** of a classifier (assuming correct classification) is the distance of the closest point to the decision boundary.

Signed distance is

$$\frac{1}{\|w\|}(w^\top x^{(n)} + w_0)$$

This is positive for correctly classified points.
Margin

the margin of a classifier (assuming correct classification)
is the distance of the closest point to the decision boundary

signed distance is \( \frac{1}{\|w\|}(w^\top x^{(n)} + w_0) \)
correcting for sign (margin) \( \frac{1}{\|w\|}y^{(n)}(w^\top x + w_0) \)
this is positive for correctly classified points
Max margin classifier

find the decision boundary with maximum margin

margin is not maximal

$y = 1$
$y = 0$
$y = -1$
Max margin classifier

find the decision boundary with maximum margin

margin is not maximal

maximum margin
Max margin classifier

find the decision boundary with maximum margin

\[
\begin{align*}
\max_{w,w_0} & \quad M \\
M & \leq \frac{1}{\|w\|_2} y^{(n)} (w^T x^{(n)} + w_0) \quad \forall n
\end{align*}
\]
Max margin classifier

find the decision boundary with maximum margin

\[
\begin{align*}
\max_{w, w_0} M \\
M \leq \frac{1}{\|w\|_2} y^{(n)} (w^\top x^{(n)} + w_0) \quad \forall n
\end{align*}
\]

only the points \(n\) with

\[
M = \frac{1}{\|w\|_2} y^{(n)} (w^\top x^{(n)} + w_0)
\]

matter in finding the boundary
Max margin classifier

find the decision boundary with maximum margin

\[
\begin{align*}
\max_{w, w_0} & \quad M \\
\text{subject to} & \quad M \leq \frac{1}{\|w\|_2} y^{(n)} (w^\top x^{(n)} + w_0) \quad \forall n
\end{align*}
\]

only the points \((n)\) with

\[
M = \frac{1}{\|w\|_2} y^{(n)} (w^\top x^{(n)} + w_0)
\]

matter in finding the boundary

these are called support vectors
Max margin classifier

find the decision boundary with maximum margin

\[
\max_{w, w_0} M \\
M \leq \frac{1}{\|w\|_2} y^{(n)} (w^\top x^{(n)} + w_0) \quad \forall n
\]

only the points \((n)\) with

\[
M = \frac{1}{\|w\|_2} y^{(n)} (w^\top x^{(n)} + w_0)
\]

matter in finding the boundary

these are called support vectors

max-margin classifier is called support vector machine (SVM)
Support Vector Machine

find the decision boundary with maximum margin

\[ \begin{align*}
\max_{w, w_0} & \quad M \\
M & \leq \frac{1}{\|w\|_2} y^{(n)} (w^\top x^{(n)} + w_0) \quad \forall n
\end{align*} \]
Support Vector Machine

find the decision boundary with maximum margin

\[
\begin{align*}
\max_{w, w_0} M &= M \\
M &\leq \frac{1}{\|w\|_2} y^{(n)} (w^\top x^{(n)} + w_0) \quad \forall n
\end{align*}
\]

if \( w^*, w_0^* \) is an optimal solution then
Support Vector Machine

find the decision boundary with maximum margin

\[
\begin{align*}
M &\leq \frac{1}{\|w\|_2} y^{(n)} (w^\top x^{(n)} + w_0) \quad \forall n \\
\{ \max_{w,w_0} M \}
\end{align*}
\]

observation

if \( w^*, w_0^* \) is an optimal solution then

\( cw^*, cw_0^* \) is also optimal (same margin)
Support Vector Machine

find the decision boundary with maximum margin

\[
\begin{align*}
\max_{w, w_0} M \\
M &\leq \frac{1}{||w||_2} y^{(n)}(w^T x^{(n)} + w_0) & \forall n
\end{align*}
\]

observation

if \( w^*, w_0^* \) is an optimal solution then

\( cw^*, cw_0^* \) is also optimal (same margin)

fix the norm of \( w \) to avoid this \( ||w||_2 = \frac{1}{M} \)
Support Vector Machine

find the decision boundary with maximum margin

\[
\max_{w, w_0} M \\
M \leq \frac{1}{|w|^2} y^{(n)} (w^\top x^{(n)} + w_0) \quad \forall n
\]
Support Vector Machine

find the decision boundary with maximum margin

\[ \max_{w, w_0} M \]

\[ M \leq \frac{1}{\|w\|_2} y^{(n)} (w^\top x^{(n)} + w_0) \quad \forall n \]
Support Vector Machine

find the decision boundary with maximum margin

\[
\begin{align*}
\max_{w, w_0} & \quad M \\
M & \leq \frac{1}{||w||_2} y^{(n)} (w^\top x^{(n)} + w_0) \quad \forall n
\end{align*}
\]

fixing \( ||w||_2 = \frac{1}{M} \)

\[
\begin{align*}
\max_{w, w_0} & \quad \frac{1}{||w||_2} \\
\frac{1}{||w||_2} & \leq \frac{1}{||w||_2} y^{(n)} (w^\top x^{(n)} + w_0) \quad \forall n
\end{align*}
\]
Support Vector Machine

find the decision boundary with maximum margin

\[ \max_{w, w_0} M \]
\[ M \leq \frac{1}{||w||^2} y^{(n)} (w^\top x^{(n)} + w_0) \quad \forall n \]

simplifying, we get hard margin SVM objective

\[ \min_{w, w_0} ||w||^2 \]
\[ y^{(n)} (w^\top x^{(n)} + w_0) \geq 1 \quad \forall n \]
Perceptron: issues

cyclic updates if the data is not linearly separable?
  • try make the data separable using additional features?
  • data may be inherently noisy

even if linearly separable
convergence could take many iterations

the decision boundary may be suboptimal
Perceptron: issues

- cyclic updates if the data is not linearly separable?
  - try make the data separable using additional features?
  - data may be inherently noisy

even if linearly separable
convergence could take many iterations

the decision boundary may be suboptimal

maximize the hard margin
Perceptron: issues

cyclic updates if the data is not linearly separable?
  • try make the data separable using additional features?
  • data may be inherently noisy

even if linearly separable
convergence could take many iterations

the decision boundary may be suboptimal

now lets fix this problem
maximize a soft margin

maximize the hard margin
Soft margin constraints

allow points inside the margin and on the wrong side
but penalize them

\[
y = -1 \\
y = 0 \\
y = 1
\]

instead of hard constraint

\[
y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 \quad \forall n
\]
**Soft margin constraints**

allow points inside the margin and on the wrong side but penalize them

\[ y = 1 \]
\[ y = 0 \]
\[ y = -1 \]

Instead of hard constraint \( y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 \ \forall n \)

Use \( y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)} \ \forall n \)
Soft margin constraints

allow points inside the margin and on the wrong side
but penalize them

Instead of hard constraint
use

\[ y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 \quad \forall n \]

\[ y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)} \quad \forall n \]

\[ \xi^{(n)} \geq 0 \text{ slack variables (one for each n)} \]
Soft margin constraints

allow points inside the margin and on the wrong side but penalize them

instead of hard constraint \( y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 \quad \forall n \)
use \( y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)} \quad \forall n \)

\( \xi^{(n)} \geq 0 \) slack variables (one for each \( n \))
\( \xi^{(n)} = 0 \) zero if the point satisfies original margin constraint
**Soft margin constraints**

allow points inside the margin and on the wrong side but penalize them

\[ y(n)(w^\top x^{(n)} + w_0) \geq 1 \quad \forall n \]

instead of hard constraint

use

\[ y(n)(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)} \quad \forall n \]

\[ \xi^{(n)} \geq 0 \quad \text{slack variables (one for each n)} \]

\[ \xi^{(n)} = 0 \quad \text{zero if the point satisfies original margin constraint} \]

\[ 0 < \xi^{(n)} < 1 \quad \text{if correctly classified but inside the margin} \]
**Soft margin constraints**

allow points inside the margin and on the wrong side but penalize them

Instead of hard constraint \( y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 \ \forall n \)

use \( y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)} \ \forall n \)

\( \xi^{(n)} \geq 0 \) slack variables (one for each \( n \))

\( \xi^{(n)} = 0 \) zero if the point satisfies original margin constraint

\( 0 < \xi^{(n)} < 1 \) if correctly classified but inside the margin

\( \xi^{(n)} > 1 \) incorrectly classified
Soft margin constraints allow points inside the margin and on the wrong side but penalize them.

\[
\text{soft-margin objective}
\]

\[
\min_{w, w_0} \frac{1}{2} \|w\|_2^2 + \gamma \sum_n \xi^{(n)}
\]

\[
y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)} \quad \forall n
\]

\[
\xi^{(n)} \geq 0 \quad \forall n
\]
Soft margin constraints

allow points inside the margin and on the wrong side but penalize them

\[
\min_{w, w_0} \frac{1}{2} \|w\|_2^2 + \gamma \sum_n \xi^{(n)} \\
y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)} \quad \forall n \\
\xi^{(n)} \geq 0 \quad \forall n
\]

\(\gamma\) is a hyper-parameter that defines the importance of constraints for very large \(\gamma\) this becomes similar to hard margin SVM
Hinge loss

would be nice to turn this into an unconstrained optimization

\[
\min_{w, w_0} \frac{1}{2} \|w\|_2^2 + \gamma \sum_n \xi^{(n)}
\]

\[
y^{(n)}(w^T x^{(n)} + w_0) \geq 1 - \xi^{(n)}
\]

\[
\xi^{(n)} \geq 0 \quad \forall n
\]
Hinge loss

would be nice to turn this into an unconstrained optimization

\[
\min_{w, w_0} \frac{1}{2} \|w\|_2^2 + \gamma \sum_n \xi^{(n)}
\]

\[
y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)}
\]

\[
\xi^{(n)} \geq 0 \quad \forall n
\]

if point satisfies the margin \( y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 \)

minimum slack is \( \xi^{(n)} = 0 \)
Hinge loss

would be nice to turn this into an unconstrained optimization

\[
\min_{w, w_0} \frac{1}{2} ||w||^2_2 + \gamma \sum_n \xi^{(n)}
\]

\[
y^{(n)}(w^T x^{(n)} + w_0) \geq 1 - \xi^{(n)}
\]

\[
\xi^{(n)} \geq 0 \quad \forall n
\]

if point satisfies the margin \( y^{(n)}(w^T x^{(n)} + w_0) \geq 1 \)

minimum slack is \( \xi^{(n)} = 0 \)

otherwise \( y^{(n)}(w^T x^{(n)} + w_0) < 1 \)

the smallest slack is \( \xi^{(n)} = 1 - y^{(n)}(w^T x^{(n)} + w_0) \)
Hinge loss

would be nice to turn this into an unconstrained optimization

\[
\min_{w, w_0} \frac{1}{2} ||w||_2^2 + \gamma \sum_n \xi^{(n)}
\]

\[
y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)}
\]

\[
\xi^{(n)} \geq 0 \quad \forall n
\]

- if point satisfies the margin \( y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 \) minimum slack is \( \xi^{(n)} = 0 \)
- otherwise \( y^{(n)}(w^\top x^{(n)} + w_0) < 1 \) the smallest slack is \( \xi^{(n)} = 1 - y^{(n)}(w^\top x^{(n)} + w_0) \)

so the optimal slack satisfying both cases

\[
\xi^{(n)} = \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0))
\]
Hinge loss

would be nice to turn this into an unconstrained optimization

\[
\min_{w, w_0} \frac{1}{2} \|w\|^2_2 + \gamma \sum_n \xi^{(n)}
\]

\[
y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)}
\]

\[
\xi^{(n)} \geq 0 \quad \forall n
\]
Hinge loss

would be nice to turn this into an unconstrained optimization

$$\min_{w, w_0} \frac{1}{2} \|w\|_2^2 + \gamma \sum_n \xi^{(n)}$$

$$y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)}$$

$$\xi^{(n)} \geq 0 \quad \forall n$$

replace $$\xi^{(n)} = \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0))$$
Hinge loss

would be nice to turn this into an unconstrained optimization

\[
\min_{w, w_0} \frac{1}{2} ||w||_2^2 + \gamma \sum_n \xi^{(n)}
\]

\[
y^{(n)} (w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)}
\]

\[
\xi^{(n)} \geq 0 \quad \forall n
\]

replace \( \xi^{(n)} = \max(0, 1 - y^{(n)} (w^\top x^{(n)} + w_0)) \)

we get \( \min_{w, w_0} \frac{1}{2} ||w||_2^2 + \gamma \sum_n \max(0, 1 - y^{(n)} (w^\top x^{(n)} + w_0)) \)
Hinge loss

would be nice to turn this into an unconstrained optimization

\[
\min_{w, w_0} \frac{1}{2} ||w||_2^2 + \gamma \sum_n \xi^{(n)}
\]

\[
y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)}
\]

\[
\xi^{(n)} \geq 0 \quad \forall n
\]

replace \( \xi^{(n)} = \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0)) \)

we get

\[
\min_{w, w_0} \frac{1}{2} ||w||_2^2 + \gamma \sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0))
\]

the same as

\[
\min_{w, w_0} \sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0)) + \frac{1}{2\gamma} ||w||_2^2
\]
Hinge loss

would be nice to turn this into an unconstrained optimization

\[
\begin{align*}
\min_{w,w_0} & \frac{1}{2} \|w\|_2^2 + \gamma \sum_n \xi^{(n)} \\
y^{(n)}(w^\top x^{(n)} + w_0) & \geq 1 - \xi^{(n)} \\
\xi^{(n)} & \geq 0 \quad \forall n
\end{align*}
\]

replace \( \xi^{(n)} = \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0)) \)

we get
\[
\min_{w,w_0} \frac{1}{2} \|w\|_2^2 + \gamma \sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0))
\]

the same as
\[
\min_{w,w_0} \sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0)) + \frac{1}{2\gamma} \|w\|_2^2
\]

this is called the hinge loss

\[
L_{hinge}(y, \hat{y}) = \max(0, 1 - y\hat{y})
\]
Hinge loss

would be nice to turn this into an unconstrained optimization

\[
\min_{w, w_0} \frac{1}{2} ||w||_2^2 + \gamma \sum_n \xi^{(n)}
\]

\[
y^{(n)}(w^T x^{(n)} + w_0) \geq 1 - \xi^{(n)}
\]

\[
\xi^{(n)} \geq 0 \quad \forall n
\]

replace \( \xi^{(n)} = \max(0, 1 - y^{(n)}(w^T x^{(n)} + w_0)) \)

we get \( \min_{w, w_0} \frac{1}{2} ||w||_2^2 + \gamma \sum_n \max(0, 1 - y^{(n)}(w^T x^{(n)} + w_0)) \)

the same as \( \min_{w, w_0} \sum_n \max(0, 1 - y^{(n)}(w^T x^{(n)} + w_0)) + \frac{1}{2\gamma} ||w||_2^2 \)

this is called the hinge loss \( L_{\text{hinge}}(y, \hat{y}) = \max(0, 1 - y\hat{y}) \)

soft-margin SVM is doing L2 regularized hinge loss minimization
Perceptron vs. SVM

Perceptron

if correctly classified evaluates to zero
otherwise it is \( \min_{w, w_0} - y(n)(w^\top x^{(n)} + w_0) \)
Perceptron vs. SVM

Perceptron

if correctly classified evaluates to zero
otherwise it is $\min_{w, w_0} -y^{(n)}(w^\top x^{(n)} + w_0)$

can be written as

$\sum_n \max(0, -y^{(n)}(w^\top x^{(n)} + w_0))$
Perceptron vs. SVM

**Perceptron**

if correctly classified evaluates to zero
otherwise it is \( \min_{w, w_0} - y^{(n)}(w^\top x^{(n)} + w_0) \)

can be written as
\[
\sum_n \max(0, -y^{(n)}(w^\top x^{(n)} + w_0))
\]

**SVM**

\[
\sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0)) + \frac{\lambda}{2} ||w||^2
\]
Perceptron vs. SVM

**Perceptron**

if correctly classified evaluates to zero
otherwise it is \( \min_{w,w_0} - y^{(n)}(w^\top x^{(n)} + w_0) \)

can be written as
\[
\sum_n \max(0, -y^{(n)}(w^\top x^{(n)} + w_0))
\]

**SVM**

\[
\sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0)) + \frac{\lambda}{2} ||w||^2
\]

so this is the difference!
(plus regularization)
Perceptron vs. SVM

**Perceptron**

if correctly classified evaluates to zero
otherwise it is \( \min_{w,w_0} -y^{(n)}(w^\top x^{(n)} + w_0) \)

can be written as
\[
\sum_n \max(0, -y^{(n)}(w^\top x^{(n)} + w_0))
\]

finds some linear decision boundary if exists

**SVM**

\[
\sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0)) + \frac{\lambda}{2} ||w||^2
\]

so this is the difference!
(plus regularization)

for small lambda finds the max-marging decision boundary
Perceptron vs. SVM

**Perceptron**

if correctly classified evaluates to zero
otherwise it is $\min_{w,w_0} - y^{(n)}(w^\top x^{(n)} + w_0))$

can be written as

$\sum_n \max(0, -y^{(n)}(w^\top x^{(n)} + w_0))$

finds some linear decision boundary if exists

stochastic gradient descent with fixed learning rate

**SVM**

$\sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0)) + \frac{\lambda}{2}||w||^2$

so this is the difference! (plus regularization)

for small lambda finds the max-marging decision boundary

depending on the formulation we have many choices
Perceptron vs. SVM

\[ J(w) = \sum_n \max(0, 1 - y^{(n)} w^\top x^{(n)}) + \frac{1}{2} ||w||_2^2 \]

now we included bias in \( w \)
Perceptron vs. SVM

\[ J(w) = \sum_n \max(0, 1 - y^{(n)} w^\top x^{(n)}) + \frac{1}{2} ||w||_2^2 \]

now we included bias in \( w \)

check that the cost function is convex in \( w \).
Perceptron vs. SVM

Cost function: \( J(w) = \sum_n \max(0, 1 - y^{(n)} w^T x^{(n)}) + \frac{1}{2} \|w\|_2^2 \)

Now we included bias in \( w \)

Check that the cost function is convex in \( w \)?
Perceptron vs. SVM

\[ J(w) = \sum_n \max(0, 1 - y^{(n)} w^\top x^{(n)}) + \frac{1}{2} \|w\|^2 \]

now we included bias in \( w \)

check that the cost function is convex in \( w \)?

hinge loss is not smooth (piecewise linear)
Perceptron vs. SVM

Cost: \( J(w) = \sum_n \max(0, 1 - y^{(n)}w^\top x^{(n)}) + \frac{\lambda}{2} \|w\|^2 \)

now we included bias in \( w \)

check that the cost function is convex in \( w \)?

hinge loss is not smooth (piecewise linear)

if we use "stochastic" sub-gradient descent

the update will look like Perceptron

if \( y^{(n)}y^{(n)} < 1 \) minimize \(-y^{(n)}(w^\top x^{(n)}) + \frac{\lambda}{2} \|w\|^2\)

otherwise, do nothing

```python
1  def cost(X,y,w, lamb=1e-3):
2      z = np.dot(X, w)
3      J = np.mean(np.maximum(0, 1 - y*z)) + lamb * np.dot(w[:-1],w[:-1])/2
4      return J
```
Perceptron vs. SVM

$J(w) = \sum_n \max(0, 1 - y^{(n)}w^Tx^{(n)}) + \frac{\lambda}{2}||w||_2^2$

- check that the cost function is convex in $w$

- hinge loss is not smooth (piecewise linear)
- if we use "stochastic" sub-gradient descent

  - the update will look like Perceptron
    - if $y^{(n)}y^{(n)} < 1$ minimize $-y^{(n)}(w^Tx^{(n)}) + \frac{\lambda}{2}||w||_2^2$
    - otherwise, do nothing

```python
def cost(X, y, w, lamb=1e-3):
    z = np.dot(X, w)
    J = np.mean(np.maximum(0, 1 - y*z)) + lamb * np.dot(w[:-1],w[:-1])/
    return J
```

```python
def subgradient(X, y, w, lamb):
    N, D = X.shape
    z = np.dot(X, w)
    violations = np.nonzero(z*y < 1)[0]
    grad = -np.dot(X[violations,:].T, y[violations])/N
    grad[:-1] += lamb2 * w[:-1]
    return grad
```
Perceptron vs. SVM

\[ J(w) = \sum_n \max(0, 1 - y^{(n)} w^T x^{(n)}) + \frac{\lambda}{2} \|w\|^2 \]

check that the cost function is convex in \( w \)

hinge loss is not smooth (piecewise linear)

if we use "stochastic" sub-gradient descent

the update will look like Perceptron

\[
\begin{align*}
\text{if } y^{(n)} y^{(n)} < 1 & \quad \text{minimize } -y^{(n)} (w^T x^{(n)}) + \frac{\lambda}{2} \|w\|^2 \\
\text{otherwise, do nothing}
\end{align*}
\]

```python
1 def cost(X,y,w, lamb=1e-3):
2     z = np.dot(X, w)
3     J = np.mean(np.maximum(0, 1 - y*z)) + lamb * np.dot(w[:-1],w[:-1])/2
4     return J
```

```python
1 def subgradient(X, y, w, lamb):
2     N,D = X.shape
3     z = np.dot(X, w)
4     violations = np.nonzero(z*y < 1)[0]
5     grad = -np.dot(X[violations,:].T, y[violations])/N
6     grad[:-1] += lamb2 * w[:-1]
7     return grad
```
Iris dataset (D=2)  
(linearly separable case)
Example

Iris dataset (D=2)  
(linearly separable case)

```python
from scipy.linalg import norm

def subgradient(X, y, w, lamb):
    g = subgradient(X, y, w)  
    w_old = w

    w = w - lr * g / np.sqrt(t + 1)
    t += 1

    return w
```

```python
def SubGradientDescent(X, y, lr=1, eps=1e-18, max_iters=1000, lamb=1e-8):
    N, D = X.shape
    w = np.zeros(D)
    t = 0
    w_old = w + np.inf

    while np.linalg.norm(w - w_old) > eps and t < max_iters:
        g = subgradient(X, y, w, lamb)
        w_old = w

        w = w - lr * g / np.sqrt(t + 1)
        t += 1

    return w
```
Example

Iris dataset (D=2) (linearly separable case)

max-margin boundary (using small lambda \( \lambda = 10^{-8} \))
Example

Iris dataset (D=2) (linearly separable case)

max-margin boundary (using small lambda $\lambda = 10^{-8}$)

compare to Perceptron’s decision boundary
Iris dataset (D=2)  
(NOT linearly separable case)

```python
1 def SubGradientDescent(X, y, lr=1, eps=1e-18, 
2 max_iters=1000, lamb=1e-8):
3     N, D = X.shape
4     w = np.zeros(D)
5     g = np.inf
6    t = 0
7    while np.linalg.norm(g) > eps and t < max_iters:
8        g = subgradient(X, y, w, lamb=lamb)
9        w = w - lr*g/np.sqrt(t+1)
10       t += 1
11    return w
```

\[ \lambda = 10^{-8} \]
Example

Iris dataset (D=2) (NOT linearly separable case)

def SubGradientDescent(X,y,lr=1,eps=1e-18, max_iters=1000, lamb=1e-8):
    N,D = X.shape
    w = np.zeros(D)
    g = np.inf
    t = 0
    while np.linalg.norm(g) > eps and t < max_iters:
        g = subgradient(X, y, w, lamb=lamb)
        w = w - lr*g/np.sqrt(t+1)
        t += 1
    return w

soft margins using small lambda $\lambda = 10^{-8}$
Example

Iris dataset (D=2)
(\textbf{NOT} linearly separable case)

```python
def SubGradientDescent(X,y,lr=1,eps=1e-18,
                         max_iters=1000, lamb=1e-8):
    N,D = X.shape
    w = np.zeros(D)
    g = np.inf
    t = 0
    while np.linalg.norm(g) > eps and t < max_iters:
        g = subgradient(X, y, w, lamb=lamb)
        w = w - lr*g/np.sqrt(t+1)
        t += 1
    return w
```

soft margins using small lambda $\lambda = 10^{-8}$

Perceptron does not converge
SVM vs. logistic regression

recall: **logistic regression** simplified cost for $y \in \{0, 1\}$

$$J(w) = \sum_{n=1}^{N} y^{(n)} \log (1 + e^{-z^{(n)}}) + (1 - y^{(n)}) \log (1 + e^{z^{(n)}})$$

where $z^{(n)} = w^\top x^{(n)}$

includes the bias
SVM vs. logistic regression

recall: **logistic regression** simplified cost for $y \in \{0, 1\}$

$$
J(w) = \sum_{n=1}^{N} y^{(n)} \log (1 + e^{-z^{(n)}}) + (1 - y^{(n)}) \log (1 + e^{z^{(n)}})
$$

where $z^{(n)} = w^\top x^{(n)}$

includes the bias

for $y \in \{-1, +1\}$ we can write this as

$$
J(w) = \sum_{n=1}^{N} \log (1 + e^{-y^{(n)} z^{(n)}}) + \frac{\lambda}{2} ||w||^2
$$

*also added L2 regularization*
SVM vs. logistic regression

recall: \textit{logistic regression} simplified cost for \( y \in \{0, 1\} \)

\[
J(w) = \sum_{n=1}^{N} y^{(n)} \log (1 + e^{-z^{(n)}}) + (1 - y^{(n)}) \log (1 + e^{z^{(n)}})
\]

where \( z^{(n)} = w^\top x^{(n)} \)

includes the bias

for \( y \in \{-1, +1\} \) we can write this as

\[
J(w) = \sum_{n=1}^{N} \log (1 + e^{-y^{(n)}z^{(n)}}) + \frac{1}{2} \|w\|^2
\]

\textit{also added L2 regularization}

compare to \textit{SVM cost} for \( y \in \{-1, +1\} \)

\[
J(w) = \sum_{n} \max(0, 1 - y^{(n)}z^{(n)}) + \frac{1}{2} \|w\|^2
\]
SVM vs. logistic regression

recall: **logistic regression** simplified cost for $y \in \{0, 1\}$

$$J(w) = \sum_{n=1}^{N} y^{(n)} \log (1 + e^{-z^{(n)}}) + (1 - y^{(n)}) \log (1 + e^{z^{(n)}})$$

where $z^{(n)} = w^\top x^{(n)}$

includes the bias

for $y \in \{-1, +1\}$ we can write this as

$$J(w) = \sum_{n=1}^{N} \log (1 + e^{-y^{(n)}z^{(n)}}) + \frac{\lambda}{2} ||w||_2^2$$

*also added L2 regularization*

compare to **SVM cost** for $y \in \{-1, +1\}$

$$J(w) = \sum_n \max(0, 1 - y^{(n)}(z^{(n)})) + \frac{\lambda}{2} ||w||_2^2$$

they both try to approximate 0-1 loss (accuracy)
Multiclass classification

can we use multiple binary classifiers?

- one versus the rest
Multiclass classification

can we use multiple binary classifiers?

one versus the rest

training:
train C different 1-vs-(C-1) classifiers

\[ z_c(x) = w_{[c]}^T x \]
Multiclass classification

can we use multiple binary classifiers?

**one versus the rest**

**training:**

train C different 1-vs-(C-1) classifiers

\[ z_c(x) = w_{[c]}^T x \]
Multiclass classification

can we use multiple binary classifiers?

**one versus the rest**

**training:**
train C different 1-vs-(C-1) classifiers

\[ z_c(x) = w^\top_{[c]} x \]

**test time:**
choose the class with the highest score

\[ z^* = \arg \max_c z_c(x) \]

image credit: Andrew Zisserman
Multiclass classification

can we use multiple binary classifiers?

**one versus the rest**

**training:**
train C different 1-vs-(C-1) classifiers

\[ z_c(x) = w_{[c]}^T x \]

**test time:**
choose the class with the highest score

\[ z^* = \arg \max_c z_c(x) \]

**problems:**
class imbalance
not clear what it means to compare \( z_c(x) \) values

image credit: Andrew Zisserman
Multiclass classification

can we use multiple binary classifiers?

one versus one
Multiclass classification

can we use multiple binary classifiers?

**one versus one**

training:
train \( \frac{C(C-1)}{2} \) classifiers for each class pair
Multiclass classification

can we use multiple binary classifiers?

**one versus one**

**training:**
train $\frac{C(C-1)}{2}$ classifiers for each class pair

**test time:**
choose the class with the highest vote
Multiclass classification

can we use multiple binary classifiers?

**one versus one**

**training:**
train $\frac{C(C-1)}{2}$ classifiers for each class pair

**test time:**
choose the class with the highest vote

**problems:**
computationally more demanding for large $C$
ambiguities in the final classification
Summary

- geometry of linear classification
- Perceptron algorithm
- distance to the decision boundary (margin)
- max-margin classification
- support vectors
- hard vs soft SVM
- relation to perceptron
- hinge loss and its relation to logistic regression
- some ideas for max-margin multi-class classification