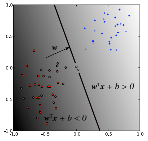
Linear models and the perceptron algorithm

Chapters 1, 3



Preliminaries

Definition: The Euclidean dot product between two vectors is the expression

$$\mathbf{w}^T\mathbf{x} = \sum_{i=1}^{d} w_i x_i$$

The dot product is also referred to as inner product or scalar product.

It is sometimes denoted as $\mathbf{w} \cdot \mathbf{x}$

(hence the name dot product).

Preliminaries

Definition: The Euclidean dot product between two vectors is the expression

$$\mathbf{w}^T\mathbf{x} = \sum_{i=1}^{d} w_i x_i$$

The dot product is also referred to as inner product or scalar product.

Geometric interpretation. The dot product between two unit vectors[1] is the cosine of the angle between them.

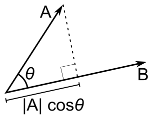
The dot product between a vector and a unit vector is the length of its projection in that direction.

And in general:

$$\mathbf{w}^\mathsf{T}\mathbf{x} = ||\mathbf{w}|| \cdot ||\mathbf{x}|| cos(\theta)$$

The norm of a vector:

$$||\mathbf{x}||^2 = \mathbf{x}^\mathsf{T}\mathbf{x}$$



Labeled data

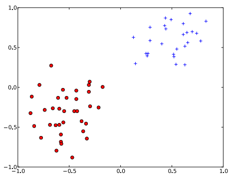
A labeled dataset:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$$

Where $\mathbf{x}_i \in \mathbb{R}^d$ are d-dimensional vectors

The labels:

are discrete for classification problems (e.g. +1, -1) for binary classification



Labeled data

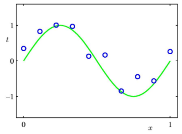
A labeled dataset:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$$

Where $\mathbf{x}_i \in \mathbb{R}^d$ are d-dimensional vectors

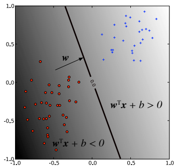
The labels:

are continuous values for a regression problem

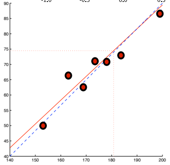


Linear models

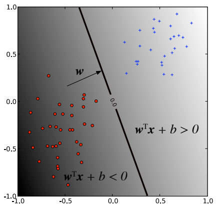
Linear models for classification (linear decision boundaries)



Linear models for regression (estimating a linear function)



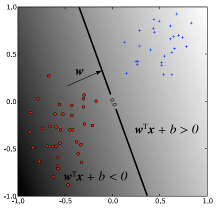
Linear models for classification



Discriminant/scoring function: $\mathbf{w}^\mathsf{T}\mathbf{x} + b$

weight vector bias

Linear models for classification



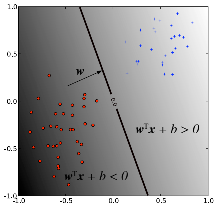
Decision boundary:

all x such that $\mathbf{w}^\mathsf{T}\mathbf{x} + b = 0$

For linear models the the decision boundary is a line in 2-d, a plane in 3-d and a hyperplane in higher dimensions

---

Linear models for classification

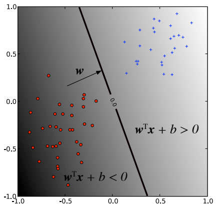


Using the discriminant to make a prediction: $\hat{y} = sign(\mathbf{w}^\mathsf{T}\mathbf{x} + b)$

the sign function equals 1 when its argument is positive and -1 otherwise

---

Linear models for classification



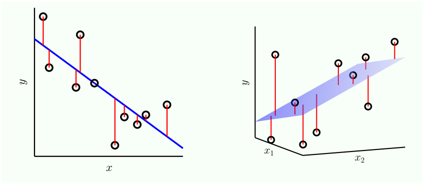
Decision boundary:
all x such that $\mathbf{w}^\mathsf{T}\mathbf{x} + b = 0$

What can you say about the decision boundary when b = 0?

Linear models for regression

When using a linear model for regression the scoring function is the prediction:

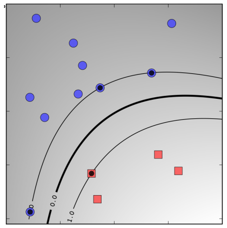
$$\hat{y} = \mathbf{w}^\mathsf{T}\mathbf{x} + b$$



Why linear?

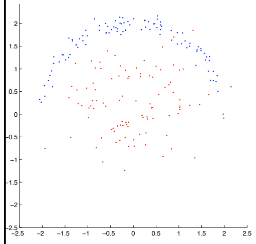
- It's a good baseline: always start simple
- Linear models are stable
- Linear models are less likely to overfit the training data because they have relatively less parameters. Can sometimes underfit. Often all you need when the data is high dimensional.
- Lots of scalable algorithms

From linear to non-linear

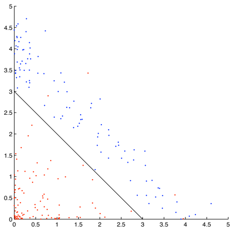


There is a neat mathematical trick that will enable us to use linear classifiers to create non-linear decision boundaries!

From linear to non-linear



Original data: not linearly separable

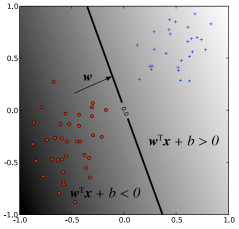


Transformed data: (x',y') = (x², y²)

Linearly separable data

Linearly separable data: there exists a linear decision boundary separating the classes.

Example:



The bias and homogeneous coordinates

In some cases we will use algorithms that learn a discriminant function without a bias term. This does not reduce the expressivity of the model because we can obtain a bias using the following trick:

Add another dimension x_0 to each input and set it to 1.

Learn a weight vector of dimension d+1 in this extended space, and interpret w_0 as the bias term. With the notation

$$\mathbf{w} = (w_1, \ldots, w_d) \quad \tilde{\mathbf{w}} = (w_0, w_1, \ldots, w_d)$$

$$\tilde{\mathbf{x}} = (1, x_1, \ldots, x_d)$$

We have that:

$$\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} = w_0 + \mathbf{w} \cdot \mathbf{x}$$

See page 7 in the book

Finding a good hyperplane

We would like a classifier that fits the data, i.e. we would like to find a vector **w** that minimizes

$$E_{\text{in}}(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\left[h(\mathbf{x}_i) \neq f(\mathbf{x}_i)\right]$$

$$= \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\left[\text{sign}(\mathbf{w}^{\mathsf{T}} \mathbf{x}_i) \neq y_i\right]$$

This is a difficult problem because of the discrete nature of the indicator and sign function (known to be NP-hard).

The perceptron algorithm (Rosenblatt, 1957)

Idea: iterate over the training examples, and update the weight vector **w** in a way that would make $\mathbf{x}_i$ is more likely to be correctly classified.

Let's assume that $\mathbf{x}_i$ is misclassified, and is a positive example i.e.

$$\mathbf{w} \cdot \mathbf{x}_i < 0$$

Note: we're learning a classifier without a bias term

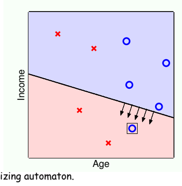
We would like to update **w** to **w'** such that

$$\mathbf{w}' \cdot \mathbf{x}_i > \mathbf{w} \cdot \mathbf{x}_i$$

This can be achieved by choosing

$$\mathbf{w}' = \mathbf{w} + \eta \mathbf{x}_i$$

$0 < \eta \leq 1$ is the learning rate



Rosenblatt, Frank (1957), The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

Section 1.1 in the book

The perceptron algorithm

If x_i is a negative example, the update needs to be opposite.
Overall, we can summarize the two cases as:

$$\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$$

```
Input:  labeled data D in homogeneous coordinates
Output:  weight vector w

w = 0
converged = false
while not converged :
    converged = true
    for i in 1,…,|D| :
        if xi is misclassified update w and set
            converged=false
return w
```

The perceptron algorithm

Since the algorithm is not guaranteed to converge you need to set a limit on the number of iterations:

```
Input:  labeled data D in homogeneous coordinates
Output:  weight vector w

w = 0
converged = false
while (not converged or number of iterations < T) :
    converged = true
    for i in 1,…,|D| :
        if xi is misclassified:
            update w and set converged=false
return w
```

The perceptron algorithm

The algorithm makes sense, but let's try to derive it in a more principled way.

The algorithm is trying to find a vector **w** that separates positive from negative examples.

We can express that as:

$$y_i \mathbf{w}^\intercal \mathbf{x}_i > 0, \quad i = 1, \ldots, n$$

For a given weight vector **w** the degree to which this does not hold can be expressed as:

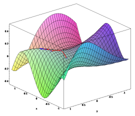
$$E(\mathbf{w}) = -\sum_{i:\ \mathbf{x}_i \text{ is misclassified}} y_i \mathbf{w}^\intercal \mathbf{x}_i$$

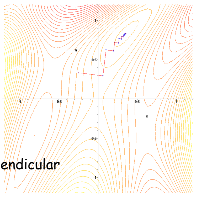
We want to find **w** that minimizes or maximizes this criterion?

Digression: gradient descent

Given a function E(**w**), the gradient is the direction of steepest ascent

Therefore to minimize E(**w**), take a step in the direction of the negative of the gradient





Notice that the gradient is perpendicular to contours of equal E(**w**)

Images from http://en.wikipedia.org/wiki/Gradient_descent

Gradient descent

We can now express gradient descent as:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla E(\mathbf{w})$$

$$\mathbf{w}(t) - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

where

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \left(\frac{\partial E(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial E(\mathbf{w})}{\partial w_d} \right)^\intercal$$

And **w**(t) is the weight vector at iteration t

The constant η is called the step size (learning rate when used in the context of machine learning).

The perceptron algorithm

Let's apply gradient descent to the perceptron criterion:

$$E(\mathbf{w}) = -\sum_{i:\ \mathbf{x}_i \text{ is misclassified}} y_i \mathbf{w}^\intercal \mathbf{x}_i$$

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = -\sum_{i:\ \mathbf{x}_i \text{ is misclassified}} y_i \mathbf{x}_i$$

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

$$= \mathbf{w}(t) + \eta \sum_{i:\ \mathbf{x}_i \text{ is misclassified}} y_i \mathbf{x}_i$$

Which is exactly the perceptron algorithm!

The perceptron algorithm

The algorithm is guaranteed to converge if the data is linearly separable, and does not converge otherwise.

Issues with the algorithm:

- The algorithm chooses an arbitrary hyperplane that separates the two classes. It may not be the best one from the learning perspective.
- Does not converge if the data is not separable (can halt after a fixed number of iterations).

There are variants of the algorithm that address these issues (to some extent).

The pocket algorithm

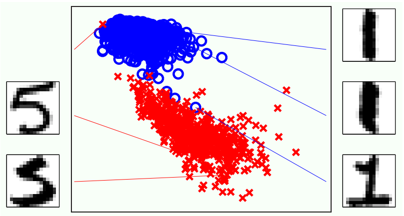
```
Input:  labeled data D in homogeneous coordinates
Output:  a weight vector $\mathbf{w}$

$\mathbf{w}$ = 0, $\mathbf{w}_{pocket}$ = 0
converged = false
while (not converged or number of iterations < T) :
    converged = true
    for i in 1,…,|D| :
        if $\mathbf{x}_i$ is misclassified:
            update $\mathbf{w}$ and set converged=false
            if $\mathbf{w}$ leads to better $E_{in}$ than $\mathbf{w}_{pocket}$ :
                  $\mathbf{w}_{pocket}$ = $\mathbf{w}$
return $\mathbf{w}_{pocket}$
```

Gallant, S. I. (1990). Perceptron-based learning algorithms. IEEE Transactions on Neural Networks, vol. 1, no. 2, pp. 179–191.

Image classification

Features: important properties of the input you think are relevant for classification



In this case we consider the level of symmetry (image – its flipped version) and overall intensity (fraction of pixels that are dark)

Pocket vs perceptron

Comparison on image data: distinguishing between the digits “1” and “5” (see page 83 in the book):