Machine Learning
Supervised Learning

dsai.asia

Asian Data Science and Artificial Intelligence Master's Program

DS&AI

Co-funded by the
Erasmus+ Programme
of the European Union

Readings

Readings for these lecture notes:

- Bishop, C. (2006), *Pattern Recognition and Machine Learning*, Springer, Chapters 3, 4, 6, 7.
- Hastie, T., Tibshirani, R., and Friedman, J. (2016), *Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, Chapters 2, 3, 4, 12.
- Ng, A. (2017), *Supervised Learning*, Lecture note set 1 for CS229, Stanford University.
- Ng, A. (2017), *Generative Learning Algorithms*, Lecture note set 2 for CS229, Stanford University.
- Ng, A. (2017), *Support Vector Machines*, Lecture note set 3 for CS229, Stanford University.

These notes contain material © Bishop (2006), Hastie et al. (2016), and Ng (2017).

Outline

1. Linear regression
2. Classification
3. Logistic regression
4. Generalized linear models
5. Generative learning algorithms
6. Support vector machines

Linear regression

Preliminaries

To see how a supervised learning model is formulated in detail, we begin with the simple case of linear regression.

Consider the simple example in which we would like to predict a person's weight given his or her height and age.

To set things up we need to specify

- The training set
- The stucture of the model
- A cost function
- A procedure for minimizing the cost function

We'll need some notation to get started.

Linear regression
Notation

Notation:

- A vector of input variables or features for a given instance will be written x or $\mathrm{x}^{(i)}$, with $i \in \{1, \ldots, m\}$.
- The output or target variable will be written y or $y^{(i)}$.
- For a supervised learning problem, a training set is a set of pairs $\left\{(\mathrm{x}^{(i)}, y^{(i)})\right\}_{i\in\{1,\ldots,m\}}$.
- The input space will be written $\mathcal{X}$, i.e., $\mathrm{x}^{(i)} \in \mathcal{X}$.
- The target space will be written $\mathcal{Y}$, i.e., $y^{(i)} \in \mathcal{Y}$.

Linear regression

Training set

We need a collection of pairs $\left\{(\mathrm{x}^{(i)}, y^{(i)})\right\}$.

Somehow we have to collect data that look like

Height	**Age**	**Weight**
180	30	70
190	25	80
...	...	...

To the extent possible, the training set population we sample from should be representative of the population we will use the trained model on later.

This is an important point that will come up later.

Linear regression

Structure of the model

If we have a simple problem where we're predicting a single real value (weight) given a single real value (height), then $\mathcal{X} = \mathcal{Y} = \mathbb{R}$.

If we have two inputs (height and age) then $\mathcal{X} = \mathbb{R}^2$.

In general, we may have n features representing each input instance, so $\mathcal{X} = \mathbb{R}^n$.

The model we learn will be a function $h : \mathcal{X} \mapsto \mathcal{Y}$.

Linear regression

Supervised learning problem

The supervised learning problem

Given a training set, learn a function $h : \mathcal{X} \mapsto \mathcal{Y}$ so that $h(\mathrm{x})$ is a “good” predictor of the corresponding value of y.

The function h is called a hypothesis.

If $\mathcal{Y}$ is continuous, we have a regression problem.

If $\mathcal{Y}$ is a discrete set, we have a classification problem.

Linear regression

Hypothesis space

Supervised machine learning algorithms can be characterized by the space of hypotheses $h : \mathcal{X} \mapsto \mathcal{Y}$ they search.

When $\mathcal{X} = \mathbb{R}^n$ and $\mathcal{Y} = \mathbb{R}$ the simplest choice is to approximate y as a linear function of x:

$$h_{\boldsymbol{\theta}}(\mathrm{x}) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n.$$

The θ_i's are parameters or weights parameterizing the space of linear mappings from $\mathcal{X}$ to $\mathcal{Y}$.

It is convenient to introduce a dummy input variable $x_0 = 1$ so that we can write

$$h_{\boldsymbol{\theta}}(\mathrm{x}) = \sum_{i=0}^{n} \theta_i x_i = \boldsymbol{\theta}^\top \mathrm{x}$$

Linear regression

Cost function

How to find a good h, or in the case of linear regression, a good $\boldsymbol{\theta}$?

In supervised learning, we require a cost function that assigns a large cost to "bad" predictions $h(\mathrm{x})$ and a small cost to "good" predictions $h(\mathrm{x})$.

For regression problems, we often used the least squares cost function

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m}(h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)}) - y^{(i)})^2.$$

Convince yourself that this choice of J gives small values for good h and large values for bad h.

Beyond this intuition, we will see later that least squares makes a lot of sense when we formulate a generative probabilistic model for the learning problem.

Linear regression

Minimizing the cost function

How to get a good $\boldsymbol{\theta}$?

We need to find

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}).$$

This will come up again and again in the course!

How to find the $\boldsymbol{\theta}$ minimizing $J(\boldsymbol{\theta})$?

- From your undergraduate probability and statistics course, perhaps, you know there is an analytical solution based on taking the gradient, setting the elements of the gradient to 0, and solving the resulting system of linear equations.
- More interesting for later in the course would be to begin with an initial guess about $\boldsymbol{\theta}$ and find the minimum using gradient descent.

Exercise: find the gradient of $J(\boldsymbol{\theta})$, i.e., $\nabla_J(\boldsymbol{\theta})$.

Linear regression

Minimizing the cost function

We have

$$\nabla_J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} & \frac{\partial J}{\partial \theta_1} & \cdots & \frac{\partial J}{\partial \theta_n} \end{bmatrix}$$

You should be able to derive

$$\begin{aligned}
\frac{\partial J}{\partial \theta_0} &= \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) \\
\frac{\partial J}{\partial \theta_1} &= \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) x_1^{(i)} \\
\ldots & \quad \ldots \\
\frac{\partial J}{\partial \theta_n} &= \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) x_n^{(i)}
\end{aligned}$$

Linear regression

Minimizing the cost function

As we said, for an analytical solution we want to find the point where the gradient is 0.

Setting the partial derivatives to 0 we obtain $n+1$ linear equations

$$
\begin{aligned}
\sum_{i=1}^{m}(\theta_0 + \theta_1 x_1^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)}) &= 0 \\
\sum_{i=1}^{m}(\theta_0 x_1^{(i)} + \theta_1 x_1^{(i)} x_1^{(i)} + \cdots + \theta_n x_n^{(i)} x_1^{(i)} - y^{(i)} x_1^{(i)}) &= 0 \\
\cdots & \quad \cdots \\
\sum_{i=1}^{m}(\theta_0 x_n^{(i)} + \theta_1 x_1^{(i)} x_n^{(i)} + \cdots + \theta_n x_n^{(i)} x_n^{(i)} - y^{(i)} x_n^{(i)}) &= 0
\end{aligned}
$$

Linear regression

Minimizing the cost function

With a bit more manipulation, we get the normal equations

$$\begin{aligned}
\theta_0 m + \theta_1 \sum_{i=1}^{m} x_1^{(i)} + \cdots + \theta_n \sum_{i=1}^{m} x_n^{(i)} &= \sum_{i=1}^{m} y^{(i)} \\
\theta_0 \sum_{i=1}^{m} x_1^{(i)} + \theta_1 \sum_{i=1}^{m} x_1^{(i)} x_1^{(i)} + \cdots + \theta_n \sum_{i=1}^{m} x_n^{(i)} x_1^{(i)} &= \sum_{i=1}^{m} y^{(i)} x_1^{(i)} \\
\cdots & \cdots \\
\theta_0 \sum_{i=1}^{m} x_n^{(i)} + \theta_1 \sum_{i=1}^{m} x_1^{(i)} x_n^{(i)} + \cdots + \theta_n \sum_{i=1}^{m} x_n^{(i)} x_n^{(i)} &= \sum_{i=1}^{m} y^{(i)} x_n^{(i)}
\end{aligned}$$

Take care to remember that the $x_j^{(i)}$ and $y(i)$ are all constant (they are given in the training set).

This means we have $n+1$ linear equations in the $n+1$ unknown elements of $\boldsymbol{\theta}$.

Linear regression

Minimizing the cost function

Next, let's try to simplify our system of equations. If we write our training set in matrix and vector form as $\mathtt{X}, \mathrm{y}$ with

$$\mathtt{X} = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \quad \mathrm{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix},$$

You should be able to verify that our system of linear equations becomes

$$\mathtt{X}^\top \mathtt{X} \boldsymbol{\theta} = \mathtt{X}^\top \mathrm{y}.$$

This is a compact form of the normal equations. $\mathtt{X}$ is called the design matrix. The parameters minimizing $J(\boldsymbol{\theta})$ must be

$$\boldsymbol{\theta} = (\mathtt{X}^\top \mathtt{X})^{-1} \mathtt{X}^\top \mathrm{y}.$$

[Be careful about conditions under which $\mathtt{X}^\top \mathtt{X}$ has no inverse.]

Now suppose we could find the gradient $\nabla_J(\boldsymbol{\theta})$ but were for some reason unable to solve for $\nabla_J(\boldsymbol{\theta}) = 0$.

Whenever we are stuck without a closed form solution to an optimization problem, we can try to use iterative numerical methods.

The simplest method is gradient descent.

The gradient $\nabla_f(\mathrm{x})$ of a function $f(\mathrm{x})$ evaluated at an arbitrary point x is a vector with two useful properties:

- The direction of $\nabla_f(\mathrm{x})$ is the direction in which $f(\mathrm{x})$ is increasing the fastest.
- The magnitude of $\nabla_f(\mathrm{x})$ is the slope of $f(\mathrm{x})$ in that direction of maximum increase.

Linear regression

Gradient descent

If we start with a guess $\boldsymbol{\theta}^{(0)}$ and calculate the gradient $\nabla_J(\boldsymbol{\theta}^{(0)})$ at that guessed point, we know that the resulting vector points in the direction in which the cost function is increasing the most.

Gradient descent simply repeats the process of moving $\boldsymbol{\theta}$ in the direction opposite $\boldsymbol{\theta}$ until we find the minimum of $J(\boldsymbol{\theta})$:

```
$\boldsymbol{\theta} \leftarrow$ some initial guess
While not converged
    $\boldsymbol{\theta}^{(n+1)} \leftarrow \boldsymbol{\theta}^{(n)} - \alpha \sum_{i=1}^{m} (h_{\boldsymbol{\theta}^{(\mathrm{n})}}(\mathrm{x}^{(i)}) - y^{(i)})\mathrm{x}^{(i)}$
```

α (a small positive real number) is called the learning rate.

Linear regression

Batch vs. stochastic gradient descent

When we update our parameter vector $\boldsymbol{\theta}$ using the gradient calculated over the entire training set, we are doing what is called batch gradient descent.

One alternative frequently used in deep learning is stochastic gradient descent, in which we repeatedly update using the gradient calculated for each training element:

```
$\boldsymbol{\theta} \leftarrow$ some initial guess
While not converged
    For $i \in 1..m$
        $\boldsymbol{\theta}^{(n+1)} \leftarrow \boldsymbol{\theta}^{(n)} - \alpha(h_{\boldsymbol{\theta}^{(\mathrm{n})}}(\mathrm{x}^{(i)}) - y^{(i)})\mathrm{x}^{(i)}$
```

Stochastic gradient descent tends to get close to the minimum faster than batch gradient descent but may oscillate around the minimum.

Slowly decreasing the learning rate α over time can improve convergence.

Linear regression

Summary

OK! Now we have three methods for minimizing $J(\boldsymbol{\theta})$:

- Solve the normal equations:

  $\boldsymbol{\theta} \leftarrow (\mathtt{X}^\top \mathtt{X})^{-1} \mathtt{X}^\top \mathrm{y}$

- Batch gradient descent:

```
$\boldsymbol{\theta} \leftarrow$ some initial guess
While not converged
    $\boldsymbol{\theta}^{(n+1)} \leftarrow \boldsymbol{\theta}^{(n)} - \alpha \sum_{i=1}^{m} (h_{\boldsymbol{\theta}^{(\mathrm{n})}}(\mathrm{x}^{(i)}) - y^{(i)}) \mathrm{x}^{(i)}$
```

- Stochastic gradient descent:

```
$\boldsymbol{\theta} \leftarrow$ some initial guess
While not converged
    For $i \in 1..m$
        $\boldsymbol{\theta}^{(n+1)} \leftarrow \boldsymbol{\theta}^{(n)} - \alpha (h_{\boldsymbol{\theta}^{(\mathrm{n})}}(\mathrm{x}^{(i)}) - y^{(i)}) \mathrm{x}^{(i)}$
```

Linear regression

First Python tutorial

In Lab 01, you'll see how we can translate the mathematics presented here into working Python code.

Linear regression

Probabilistic interpretation

Last question about linear regression: why least squares? Why not some other cost function?

The least squares method comes naturally from a probabilistic interpretation of the problem.

We suppose that our measurements $(\mathrm{x}^{(i)}, y^{(i)})$ were generated according to

$$y^{(i)} = \boldsymbol{\theta}^\top \mathrm{x}^{(i)} + \epsilon^{(i)},$$

where $\epsilon^{(i)}$ is an error term representing noise and whatever effects our linear model doesn't capture.

Linear regression

Probabilistic interpretation

We might assume that the noise terms $\epsilon^{(i)}$ were sampled i.i.d.[1] from some distribution $p(\epsilon^{(i)})$.

[Is i.i.d. a reasonable assumption?]

In many applications, a natural choice of $p(\epsilon^{(i)})$ is a Gaussian (normal distribution), which we write

$$\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2).$$

The univariate zero-mean Gaussian distribution is written

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)}.$$

[1] Independently and identically distributed

Linear regression

Probabilistic interpretation

We would like to know $p(y^{(i)} \mid \mathrm{x}^{(i)}; \boldsymbol{\theta})$.

[This is the density over possible $y^{(i)}$ (weight measurements) we would obtain once we've identified $\mathrm{x}^{(i)}$ (the individual's height and age).]

Recall that the density over $\epsilon^{(i)}$ is

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left(\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)}.$$

Since $\epsilon^{(i)} = y^{(i)} - \boldsymbol{\theta}^\top \mathrm{x}^{(i)}$, the density over $y^{(i)}$ is just a shifted version of the density over $\epsilon^{(i)}$:

$$p(y^{(i)} \mid \mathrm{x}^{(i)}; \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(y^{(i)} - \boldsymbol{\theta}^\top \mathrm{x}^{(i)})^2}{2\sigma^2}\right)}.$$

Note: the semicolon (;) in the expression $p(y^{(i)} \mid \mathrm{x}^{(i)}; \boldsymbol{\theta})$ means that the distribution is parameterized by constant $\boldsymbol{\theta}$. $\boldsymbol{\theta}$ is not a random variable.

Linear regression

Probabilistic interpretation

Now consider the distribution of the vector of targets y given the design matrix $\mathtt{X}$. We write this

$$p(\mathrm{y} \mid \mathtt{X}; \boldsymbol{\theta}).$$

Since the elements of y are independent, we can write

$$p(\mathrm{y} \mid \mathtt{X}; \boldsymbol{\theta}) = \prod_{i=1}^{m} p(y^{(i)} \mid \mathrm{x}^{(i)}; \boldsymbol{\theta})$$

Since y, $\mathtt{X}$, and $\boldsymbol{\theta}$ are all constant, we are simply multiplying the "heights" of m Gaussians at the $y^{(i)}$'s.

The Gaussians are centered at our predictions of the $y^{(i)}$'s, so the conditional probability $p(\mathrm{y} \mid \mathtt{X}; \boldsymbol{\theta})$ will be maximized when our predictions of the $y^{(i)}$'s are perfect.

This means choosing $\boldsymbol{\theta}$ to maximize $p(\mathrm{y} \mid \mathtt{X}; \boldsymbol{\theta})$ would be good!

Linear regression

Probabilistic interpretation

Now we think we would like to choose $\boldsymbol{\theta}$ to maximize $p(\mathrm{y} \mid \mathtt{X}; \boldsymbol{\theta})$.

First, let's write this distribution as a function of the parameter vector $\boldsymbol{\theta}$:

$$L(\boldsymbol{\theta}) = L(\boldsymbol{\theta}; \mathtt{X}, \mathrm{y}) = p(\mathrm{y} \mid \mathtt{X}; \boldsymbol{\theta})$$

$L(\boldsymbol{\theta})$ is called the likelihood function.

Maximum likelihood

The maximum likelihood principle states that we should choose the $\boldsymbol{\theta}$ that makes the likelihood $L(\boldsymbol{\theta})$ as large as possible:

$$\boldsymbol{\theta}^* = \operatorname*{argmax}_{\boldsymbol{\theta}} L(\boldsymbol{\theta}).$$

OK, let's try to maximize $L(\boldsymbol{\theta})$.

Linear regression

Probabilistic interpretation

As we already noted, since by assumption the $\epsilon^{(i)}$'s are independent,

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} p(y^{(i)} \mid \mathrm{x}^{(i)}; \boldsymbol{\theta}) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y^{(i)} - \boldsymbol{\theta}^\top \mathrm{x}^{(i)})^2}{2\sigma^2}\right). \end{aligned}$$

This looks difficult to maximize directly (try to evalute $\frac{\partial L}{\partial \boldsymbol{\theta}}$).

However, we can equivalenently maximize any function that is strictly increasing in $L(\boldsymbol{\theta})$.

Linear regression

Probabilistic interpretation

It will be more convenient to maximize the log likelihood $\ell(\boldsymbol{\theta})$:

$$\begin{aligned}
\ell(\boldsymbol{\theta}) &= \log L(\boldsymbol{\theta}) \\
&= \log \prod_{i=1}^{m} p(y^{(i)} \mid \mathrm{x}^{(i)}; \boldsymbol{\theta}) \\
&= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y^{(i)} - \boldsymbol{\theta}^\top \mathrm{x}^{(i)})^2}{2\sigma^2}\right) \\
&= m \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - \boldsymbol{\theta}^\top \mathrm{x}^{(i)})^2.
\end{aligned} \tag{1}$$

Linear regression

Probabilistic interpretation

Clearly, any $\boldsymbol{\theta}$ maximizing

$$m \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - \boldsymbol{\theta}^\top \mathrm{x}^{(i)})^2$$

will also maximize

$$-\frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \boldsymbol{\theta}^\top \mathrm{x}^{(i)})^2.$$

Amazingly, maximizing $L(\boldsymbol{\theta})$ or $\ell(\boldsymbol{\theta})$ gives us the same $\boldsymbol{\theta}$ we get by minimizing

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^\top \mathrm{x}^{(i)})^2.$$

Linear regression

Probabilistic interpretation

We thus see that for linear regression, least squares and maximum likelihood are equivalent.

The equivalence comes from the assumption of i.i.d. Gaussian errors in our samples $y^{(i)}$.

For other problems, we may not be able to formulate a least squares cost function, but we will be able to use the more general principle of maximum likelihood.

Linear regression

Summary of terms

Term	Definition / Symbol	Example
Training set	$(\mathrm{x}^{(i)}, y^{(i)})_{i \in 1..m}$	Table of heights, ages, and weights
Features	$\mathrm{x}, \mathrm{x}^{(i)} \in \mathcal{X}$	Height, age ($\mathbb{R}^2$)
Target	$y, y^{(i)} \in \mathcal{Y}$	Weight ($\mathbb{R}$)
Hypothesis	$h_{\boldsymbol{\theta}} : \mathcal{X} \mapsto \mathcal{Y}$	$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
Parameters or weights	$\boldsymbol{\theta}$	θ_0 (intercept), θ_1 (kgs per cm), θ_2 (kgs per year)
Cost function	$J(\boldsymbol{\theta})$	$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m}(h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)}) - y^{(i)})^2$ or negative log likelihood $\ell(\boldsymbol{\theta})$

Linear regression

Summary of terms

Term	Definition / Symbol	Example
Design matrix	$\mathtt{X}$	$\mathtt{X} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ \vdots & \vdots & \vdots \end{bmatrix}$
Normal equations	$\mathtt{X}^\top \mathtt{X}\theta = \mathtt{X}^\top \mathtt{y}$	
Batch gradient descent	$\boldsymbol{\theta}^{n+1} \leftarrow \boldsymbol{\theta}^{n} - \alpha \nabla_J(\boldsymbol{\theta}^{(n)})$	$\boldsymbol{\theta}^{(n+1)} \leftarrow \boldsymbol{\theta}^{(n)} - \alpha \sum_{i=1}^{m} (h_{\boldsymbol{\theta}^{(\mathrm{n})}}(\mathrm{x}^{(i)}) - y^{(i)})\mathrm{x}^{(i)}$
Stochastic gradient descent	$\boldsymbol{\theta}^{n+1} \leftarrow \boldsymbol{\theta}^{n} - \alpha \frac{\partial (h_{\boldsymbol{\theta}^n}(\mathrm{x}^{(i)}) - y^{(i)})^2}{\partial \boldsymbol{\theta}^n}$	$\boldsymbol{\theta}^{(n+1)} \leftarrow \boldsymbol{\theta}^{(n)} - \alpha (h_{\boldsymbol{\theta}^{(\mathrm{n})}}(\mathrm{x}^{(i)}) - y^{(i)})\mathrm{x}^{(i)}$
Learning rate	α	0.000001
Gaussian/normal distribution	$\mathcal{N}(\mu, \sigma^2)$	$\mathcal{N}(y^{(i)} - h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)}), \sigma^2)$

Linear regression

Overfitting and underfitting

So far we have only considered the linear model

$$h_{\boldsymbol{\theta}}(\mathrm{x}) = \boldsymbol{\theta}^\top \mathrm{x}.$$

What if y is not actually linearly related to x?

A simple trick is to add features that are nonlinear in x. For example, if we have just one feature x, we add a new feature x^2:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2.$$

The model is still linear in $\boldsymbol{\theta}$.

Linear regression

Overfitting and underfitting

We can take this further, for example to degree 5:

$$y = \sum_{j=0}^{5} \theta_j x^j .$$

Linear regression

Overfitting and underfitting

Increasing the degree of $h_{\boldsymbol{\theta}}(\mathrm{x})$ works even if we have more than one feature ($n > 1$).

With two features and polynomial degree 2:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2.$$

The question is how far we should take this?

It turns out that we can usually improve prediction performance on the training set by increasing the complexity of our model.

What is the right complexity for a model?

We will return to this question later in the course, but for now, we introduce a simple technique:

Instead of acquiring a single training set $X = (\mathrm{x}^{(i)}, y^{(i)})_{i \in 1..m}$, acquire two data sets:

- The training set $X^{(train)}$ consisting of m_1 pairs, used to estimate the optimal parameters $\boldsymbol{\theta}$ for $h_{\boldsymbol{\theta}}(\mathrm{x})$.
- A separate test set $X^{(test)}$ consisting of m_2 pairs, used to check the generalization ability of the resulting $h_{\boldsymbol{\theta}}(\mathrm{x})$.

Linear regression

Overfitting and underfitting

As we increase the complexity of $h_{\boldsymbol{\theta}}(\mathrm{x})$, the cost $J(\boldsymbol{\theta})$ evaluated on the training set will usually decrease monotonically.

However, we will also take the optimal $\boldsymbol{\theta}$ for the training set and calculate the same cost function $J(\boldsymbol{\theta})$ over the test set.

Good heuristics for finding the right model complexity given separate training and test sets:

- When both training and test cost is high, we are underfitting the data. Increase complexity.
- When training error is decreasing but test error is increasing, we are overfitting the training data. Decrease complexity.

Linear regression

Overfitting and underfitting

Exercise: generate a height/weight dataset in which there is a quadratic relationship between height and weight. Show that the linear model underfits the data and that higher order polynomial fits overfit the data.

For example, you might use $y = \theta_0 + \theta_1 x + \theta_2 x^2 + \epsilon$ and $\boldsymbol{\theta} = \begin{bmatrix} -426 & 5.31 & -0.0139 \end{bmatrix}$.

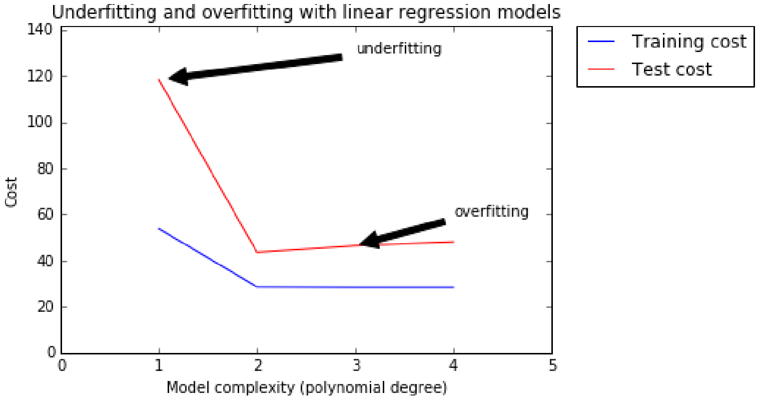
Check that the overfitting problem gets worse as the size of the training set gets smaller or the complexity of the model increases.

Linear regression

Overfitting and underfitting

See the handout on the course Web page for detailed code and results.

Here is the gist:



If you play with the overfitting and underfitting tutorial notebook, you'll find that the test set heuristic is not always perfectly reliable.

Even when we know that the "ground truth" model is quadratic, the test set might happen to be best fit by a higher-complexity model, depending on the training and test set distributions.

Is there any other (hopefully more reliable) way to avoid overfitting?

One general approach that we'll see more of later is a non-paramteric approach.

The particular non-paramteric approach we'll explore today is locally weighted linear regression.

The basic approach is to define

$$h(\mathrm{x}) = \boldsymbol{\theta}^\top \mathrm{x},$$

where

$$\boldsymbol{\theta} = \underset{\boldsymbol{\theta}_l}{\operatorname{argmin}} \sum_{i=1}^{m} w^{(i)} (y^{(i)} - \boldsymbol{\theta}_l^\top \mathrm{x}^{(i)})^2$$

and $w^{(i)}$ is a non-negative weight on training example i.

$w^{(i)}$ controls how much training example i influences $\boldsymbol{\theta}$.

How to set $w^{(i)}$?

One choice of $w^{(i)}$ uses a Gaussian kernel around $\mathrm{x}^{(i)}$:

$$w^{(i)} = e^{-\frac{(\mathrm{x}^{(i)}-\mathrm{x})^\top(\mathrm{x}^{(i)}-\mathrm{x})}{2\tau^2}}$$

LWLR is a non-paramteric method because it doesn't derive a complete fixed vector of parameters from the training set.

Instead, we keep the entire training set around, and when we want a prediction for a previously unseen x, we do a local fit given the nearby training set items.

The Gaussian kernel width τ is called the bandwidth parameter, which affects how local LWLR is.

The bandwidth parameter τ is our first example of a free parameter or a hyperparameter.

Exercise: derive the calculation of $\boldsymbol{\theta}$ given a set of weights $w^{(i)}$ over the training set.

Hint: factor the weights into a diagonal matrix `W` so that you end up with `W` in the right place on the left and right side of the normal equations.

Exercise: write a Python function that implements $h_{\mathtt{X},\mathrm{y},\tau}(\mathrm{x})$.

Outline

1. Linear regression
2. Classification
3. Logistic regression
4. Generalized linear models
5. Generative learning algorithms
6. Support vector machines

Classification

Next we move to classification.

We still have a training set $(\mathrm{x}^{(i)}, y^{(i)})_{i \in 1..m}$ and a hypothesis $h_{\boldsymbol{\theta}}(\mathrm{x})$, but now the values $y^{(i)}$ can take on are a small number of discrete values.

These discrete values could be unordered (cat, dog, mouse) or ordered (small, medium, large).

We'll first consider binary classification in which $y^{(i)} \in \{0, 1\}$.

Think of some examples of binary classification:

- Classifying an image as a cat or a dog.
- Classifying an email as spam or non-spam.
- Classifying a university degree as fake or legitimate.
- And on and on ...

Sometimes class 1 will be called positive and class 0 will be called negative.

Classification

Example

A great example of classification is deciding whether an email is spam or not spam.

Suppose we read thousands of email messages, manually divvyed them up into a `spam` and a `email` folder, and counted the frequency of each word and punctuation mark in each message.

We might get data like this for the words and characters with the largest average difference between `spam` and `email` (Hastie et al., 2016):

	`george`	`you`	`your`	`hp`	`free`	`hp1`	`!`	`our`	`re`	`edu`	`remove`
spam	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13	0.01	0.28
email	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42	0.29	0.01

Each training vector $\mathrm{x}^{(i)}$ would contain the frequency of the tokens `george`, `you`, and so on in one document. The corresponding $y^{(i)}$ might be 0 for email and 1 for spam.

Outline

1. Linear regression

2. Classification

3. Logistic regression

4. Generalized linear models

5. Generative learning algorithms

6. Support vector machines

Logistic regression

Hypothesis space

Given what we know so far, the simplest thing to do might be to ignore the fact that the $y^{(i)}$'s only come in two values and use linear regression with the values 0 and 1.

Given a query value for x, then, we might answer "1" if $h_{\boldsymbol{\theta}}(\mathrm{x}) \geq 0.5$ and "0" otherwise.

This won't work very well!

Improvement: let's change the form of h to constrain it to the range [0..1]:

$$h_{\boldsymbol{\theta}}(\mathrm{x}) = g(\boldsymbol{\theta}^\top \mathrm{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \mathrm{x}}}.$$

Here the function

$$g(z) = \frac{1}{1 + e^{-z}}$$

is called the logistic sigmoid function or just the sigmoid function.

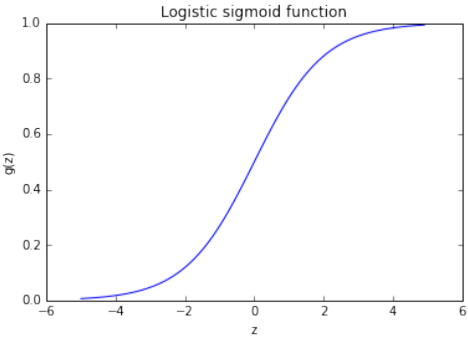
Logistic regression

Logistic sigmoid

```
import numpy
import matplotlib.pyplot as plt

def f(z):
    return 1/(1+numpy.exp(-z))

z = numpy.arange(-5, 5, 0.1)
plt.plot(z, f(z), 'b')
plt.xlabel('z')
plt.ylabel('g(z)')
plt.title('Logistic sigmoid function')
```



Logistic regression

Derivative of the sigmoid function

To optimize a cost function involving the logistic sigmoid, we'll need to know its derivative. Here's a nice trick:

$$\begin{aligned}
g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\
&= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\
&= \frac{1}{1+e^{-z}} \cdot \left(\frac{1+e^{-z}-1}{1+e^{-z}}\right) \\
&= g(z)(1-g(z))
\end{aligned}$$

Logistic regression

Cost function

So far we have seen

- Training set. We are currently limiting ourselves to binary classification.
- The form of the hypothesis $h_{\boldsymbol{\theta}}(\mathrm{x}) = g(\boldsymbol{\theta}^\top \mathrm{x})$.

What else do we need? A cost function and a method to minimize that cost.

Since least squares doesn't make sense for classification, we'll instead maximize the likelihood (actually the log likelihood) of $\boldsymbol{\theta}$.

(Our cost function, then, will be negative log likelihood.)

Logistic regression

Likelihood function

The likelihood of $\boldsymbol{\theta}$ is

$$\begin{aligned} L(\boldsymbol{\theta}) &= p(\mathrm{y} \mid \mathtt{X}; \boldsymbol{\theta}) \\ &= \prod_{i=1}^{m} p(y^{(i)} \mid \mathrm{x}^{(i)}; \boldsymbol{\theta}). \end{aligned}$$

How should we model $p(y^{(i)} \mid \mathrm{x}^{(i)}; \boldsymbol{\theta})$?

It should be maximal when our prediction $h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)})$ is correct and minimal when the prediction is incorrect.

Logistic regression

Likelihood function

In logistic regression we assume

$$p(y \mid \mathrm{x}; \boldsymbol{\theta}) = \begin{cases} h_{\boldsymbol{\theta}}(\mathrm{x}) & y = 1 \\ 1 - h_{\boldsymbol{\theta}}(\mathrm{x}) & y = 0 \end{cases}$$

That is, $h_{\boldsymbol{\theta}}(\mathrm{x})$ models $p(y = 1 \mid \mathrm{x})$.

[Discuss why this is reasonable.]

A compact representation of the equation:

$$p(y \mid \mathrm{x}; \boldsymbol{\theta}) = (h_{\boldsymbol{\theta}}(\mathrm{x}))^y (1 - h_{\boldsymbol{\theta}}(\mathrm{x}))^{1-y}$$

Logistic regression

Likelihood function

We thus obtain

$$\begin{aligned} L(\boldsymbol{\theta}) &= \prod_{i=1}^{m} p(y^{(i)} \mid \mathrm{x}^{(i)}; \boldsymbol{\theta}) \\ &= \prod_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)}))^{y^{(i)}} (1 - h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)}))^{1-y^{(i)}} \end{aligned}$$

and the log likelihood is

$$\begin{aligned} \ell(\boldsymbol{\theta}) &= \log L(\boldsymbol{\theta}) \\ &= \sum_{i=1}^{m} y^{(i)} \log h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)})), \end{aligned}$$

Logistic regression

Maximizing the likelihood function

To find the maximum of $\ell(\boldsymbol{\theta})$, we first need to find $\nabla_{\ell}(\boldsymbol{\theta})$.

We won't have a closed form solution for $\nabla_{\ell}(\boldsymbol{\theta}) = 0$ so instead we'll need to use gradient ascent.

Let's first try the stochastic version of gradient ascent in which we assume the training set is just a single pair (x, y).

Logistic regression

Maximizing the likelihood function

$$\frac{\partial}{\partial \theta_j}\ell(\boldsymbol{\theta}) = \left(y \frac{1}{g(\boldsymbol{\theta}^\top \mathrm{x})} - (1-y)\frac{1}{1-g(\boldsymbol{\theta}^\top \mathrm{x})} \right) \frac{\partial}{\partial \theta_j} g(\boldsymbol{\theta}^\top \mathrm{x}).$$

Since

$$\frac{\partial}{\partial \theta_j} g(\boldsymbol{\theta}^\top \mathrm{x}) = g(\boldsymbol{\theta}^\top \mathrm{x})(1-g(\boldsymbol{\theta}^\top \mathrm{x}))\frac{\partial}{\partial \theta_j}(\boldsymbol{\theta}^\top \mathrm{x}),$$

we obtain

$$\begin{aligned} \frac{\partial}{\partial \theta_j}\ell(\boldsymbol{\theta}) &= \Big(y(1-g(\boldsymbol{\theta}^\top \mathrm{x})) - (1-y)g(\boldsymbol{\theta}^\top \mathrm{x}) \Big) x_j \\ &= (y - h_{\boldsymbol{\theta}}(\mathrm{x}))x_j. \end{aligned}$$

Therefore

$$\nabla_\ell(\boldsymbol{\theta}) = (y - h_{\boldsymbol{\theta}}(\mathrm{x}))\mathrm{x}.$$

Logistic regression

Stochastic gradient ascent rule

We thus get the stochastic gradient ascent rule for logistic regression:

$$\boldsymbol{\theta}^{(n+1)} \leftarrow \boldsymbol{\theta}^{(n)} + \alpha(y^{(i)} - h_{\boldsymbol{\theta}}(\mathsf{x}^{(i)}))\mathsf{x}^{(i)}.$$

Does this look familiar? Take a look at page 23!

Note that the update is not exactly the same, as $h_{\boldsymbol{\theta}}(\mathsf{x})$ is not the same.

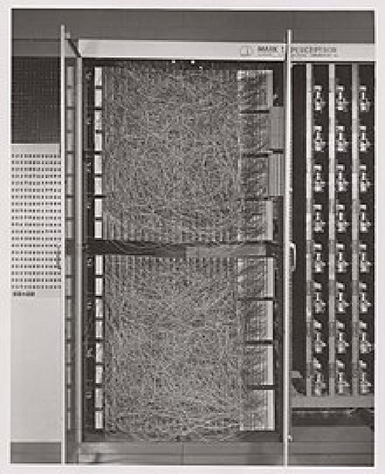
Still, it is amazing that we get similar update rules for

- Linear regression with least squares
- Linear regression with maximum likelihood
- Logistic regression with maximum likelihood

We will see how generalized linear models unify these seemingly different methods.

Logistic regression

Relationship to the perceptron



Mark I Perceptron Machine (Wikipedia)

In 1957, Rosenblatt conceived of the perceptron, a physical machine implementing the classification function

$$h_{\boldsymbol{\theta}}(\mathrm{x}) = g(\boldsymbol{\theta}^\top \mathrm{x})$$

with

$$g(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}.$$

Logistic regression

Relationship to the perceptron

The perceptron learning algorithm also used the update rule

$$\boldsymbol{\theta}^{(n+1)} \leftarrow \boldsymbol{\theta}^{(n)} + \alpha(y^{(i)} - h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)}))\mathrm{x}^{(i)}.$$

Note that this is also different from the logistic and linear regression rules since $h_{\boldsymbol{\theta}}(\mathrm{x})$ is in this case a hard threshold classifier without any probabilistic interpretation.

Logistic regression

Relationship to the perceptron

Rosenblatt was extremely optimistic about the perceptron (the first "neural network"), predicting it may in the future learn and translate languages.

Minsky and Papert's (1969) *Perceptrons* critiqued the power of the perceptron and helped kill neural network research for many years.

Neural network research later went through two "rebirths:"

- Rumelhart, Hinton, and Williams (1986): backpropagation
- Krizhevsky, Sutskever, and Hinton (2012): AlexNet

We're thus in the third cycle of possibly over-hyped expectations about neural networks!

Logistic regression

Netwon's method

We are now familiar with solving supervised learning problems through

- Direct solution of $\nabla_J(\boldsymbol{\theta}) = 0$ or $\nabla_\ell(\boldsymbol{\theta}) = 0$
- Gradient descent on a cost function $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_J(\boldsymbol{\theta})$
- Gradient ascent on a log likelihood function $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_\ell(\boldsymbol{\theta})$

We've seen that the gradient methods are not very efficient. Our jump will usually either undershoot or overshoot depending on the local gradient and our distance from the minimum.

Newton's method is a faster iterative method for finding the zero of an aribtrary nonlinear function.

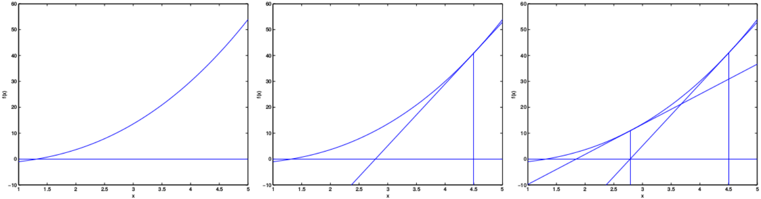
Logistic regression

Netwon's method

The update rule in Netwon's method (for a 1D function) is

$$\theta^{(n+1)} \leftarrow \theta^{(n)} - \frac{f(\theta)}{f'(\theta)}.$$

This corresponds to finding the intersection of the tangent to $f(\theta)$ with the line $y = 0$:



Ng, CS229 lecture note set #1

See animated Newton's method examples online, e.g. at `http://www.cs.uleth.ca/~holzmann/notes/NewtonsMethod/`

Logistic regression

Netwon's method

Note, however, that in our optimization problems, we don't want to find a zero of the objective function; we want to find the minimum or maximum.

Luckily the problem is easily solved: find the minimum or maximum of $J(\boldsymbol{\theta})$ by finding the zero of $\nabla_J(\boldsymbol{\theta})$.

Exercise: use Newton's method to find the minimum of

$$f(x) = (x-2)^2 + 1$$

beginning from $x_0 = 0$. How many iterations are required? Check by inspecting the function or solving $f'(x) = 0$.

Logistic regression

Netwon's method

For the logistic regression problem, then, instead of slowly climbing the gradient of $\ell(\boldsymbol{\theta})$, we can use Newton's method to find the zero of $\nabla_{\ell}(\boldsymbol{\theta})$.

In 1D, we would have

$$\theta \leftarrow \theta - \frac{\ell'(\theta)}{\ell''(\theta)}.$$

In multiple dimensions, the Newton-Raphson generalization of Newton's rule is

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - (\mathtt{H}_{\ell}(\boldsymbol{\theta}))^{-1}\nabla_{\ell}(\boldsymbol{\theta}).$$

The matrix

$$\mathtt{H}_{\ell}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial^2}{\partial\theta_0\theta_0}\ell(\boldsymbol{\theta}) & \frac{\partial^2}{\partial\theta_0\theta_1}\ell(\boldsymbol{\theta}) & \cdots \\ \frac{\partial^2}{\partial\theta_1\theta_0}\ell(\boldsymbol{\theta}) & \frac{\partial^2}{\partial\theta_1\theta_1}\ell(\boldsymbol{\theta}) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

of second derivatives is called the Hessian of $\ell(\boldsymbol{\theta})$.

Logistic regression

Netwon's method

Exercise: recalling that for a one-pair training set,

$$\frac{\partial}{\partial \theta_j}\ell(\boldsymbol{\theta}) = (y - h_{\boldsymbol{\theta}}(\mathrm{x}))x_j,$$

find $\mathtt{H}_\ell(\boldsymbol{\theta})$.

In PS2, you will implement the batch Newton's method for logistic regression in Python.

Outline

1. Linear regression
2. Classification
3. Logistic regression
4. Generalized linear models
5. Generative learning algorithms
6. Support vector machines

Generalized linear models

Introduction

In linear regression, we observe a random variable y assumed to be drawn from a Gaussian distribution depending linearly on a random vector x drawn from some population with conditional density

$$p(y \mid \mathrm{x}; \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}^\top \mathrm{x}, \sigma^2).$$

In logistic regression, we observe a random variable y assumed to be drawn from a Bernoulli distribution[2] depending on a random vector x drawn from some population with conditional density

$$p(y \mid \mathrm{x}; \boldsymbol{\theta}) = \mathrm{Bernoulli}(g(\boldsymbol{\theta}^\top \mathrm{x})).$$

We'll see that these are both generalized linear models (GLMs).

[2] Bernoulli just means y is sampled so that $y = 1$ with probability p and $y = 0$ with probability $1 - p$.

To understand GLMs we need to understand the exponential family of distributions. Following Ng, a class of distributions is in the exponential family if it can be written in the form

$$p(y; \boldsymbol{\eta}) = b(y)e^{(\boldsymbol{\eta}^\top T(y) - a(\boldsymbol{\eta}))},$$

where

- $\boldsymbol{\eta}$ is the natural parameter or canonical parameter of the distribution,
- $T(y)$ is the sufficient statistic (we normally use $T(y) = y$), and
- $a(\boldsymbol{\eta})$ is the log partition function. We use $e^{-a(\boldsymbol{\eta})}$ just to normalize the distribution to have a sum or integral of 1.
- $b(y)$ is an arbitrary scalar function of y.

Each choice of T, a, and b defines a family (set) of distributions parameterized by $\boldsymbol{\eta}$.

Generalized linear models

Exponential family: Bernoulli

The Gaussian and Bernoulli distributions are both exponential family distributions.

If y is $\mathrm{Bernoulli}(\phi)$, then $p(y = 1; \phi) = \phi$ and $p(y = 0; \phi) = 1 - \phi$.

We can rewrite (using the substitution $z = e^{\log z}$)

$$\begin{aligned} p(y; \phi) &= \phi^y (1 - \phi)^{1-y} \\ &= e^{(y \log \phi + (1-y) \log(1-\phi))} \\ &= e^{(\log \frac{\phi}{1-\phi}) y + \log (1-\phi)} \end{aligned}$$

assigning $\eta = [\log \frac{\phi}{1-\phi}]$, $T(y) = y$, $a(\eta) = \log(1 + e^\eta)$, and $b(y) = 1$, we see that $p(y; \phi)$ is in the exponential family.

For the 1D Gaussian, assuming for now that $\sigma^2 = 1$, we obtain

$$\begin{aligned} p(y;\mu) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y-\mu)^2} \\ &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y^2} e^{\mu y - \frac{1}{2}\mu^2} \end{aligned}$$

If we assign $\eta = \mu$, $T(y) = y$, $a(\eta) = \mu^2/2 = \eta^2/2$, and $b(y) = (1/\sqrt{2\pi})e^{-y^2/2}$, we see that the Gaussian is in the exponential family as well.

Generalized linear models

Exponential family

Other members of the exponential family useful for machine learning:

- Multinomial, for *n*-class classification problems
- Poisson, for modeling count data
- Gamma and exponential, for continuous non-negative random variables like time intervals
- Beta and Dirichlet for distributions over probabilities
- and more...

If y is in the exponential family given x and $\boldsymbol{\theta}$, we can apply the same procedure (the GLM recipe) to come up with a model.

Generalized linear models

Assumptions

The GLM makes three assumptions:

1. $p(y \mid \mathrm{x}; \boldsymbol{\theta}) = \mathrm{ExponentialFamily}(\boldsymbol{\eta})$.
2. Given x, we would like to predict an expected value of $T(y)$ given x.
3. $\boldsymbol{\eta}$ is linear in x, i.e., $\eta_i = \boldsymbol{\theta}_i^\top \mathrm{x}$.

Assumption 2 means that we want to learn a hypothesis function $h(\mathrm{x}) = E[y \mid \mathrm{x}]$.

For logistic regression, this would be

$$\begin{aligned} h_{\boldsymbol{\theta}}(\mathrm{x}) &= E[y \mid \mathrm{x}; \boldsymbol{\theta}] \\ &= 0 \cdot p(y = 0 \mid \mathrm{x}; \boldsymbol{\theta}) + 1 \cdot p(y = 1 \mid \mathrm{x}; \boldsymbol{\theta}) \\ &= p(y = 1 \mid \mathrm{x}; \boldsymbol{\theta}) \end{aligned}$$

Generalized linear models

Linear regression as a GLM

In the linear regression setting, if we apply the GLM assumptions with the Gaussian distribution, we obtain

$$y \sim \mathcal{N}(\mu, \sigma^2),$$

and $h_{\boldsymbol{\theta}}(\mathrm{x})$ needs to be a prediction of $T(y)$ given x and $\boldsymbol{\theta}$.

We already found that the Gaussian is an exponential family distribution with natural parameter $\eta = \mu$.

Since $\eta = \boldsymbol{\theta}^\top \mathrm{x}$, letting $T(y) = y$, we obtain

$$\begin{aligned} h_{\boldsymbol{\theta}}(\mathrm{x}) &= E[y \mid \mathrm{x}; \boldsymbol{\theta}] \\ &= \mu \\ &= \eta \\ &= \boldsymbol{\theta}^\top \mathrm{x}. \end{aligned}$$

Generalized linear models

Linear regression as a GLM

So now we have a new justification for the least squares approach to estimate a relationship between a continuous variable y and a feature vector x.

If we assume y given x is Gaussian with some mean μ, the GLM approach states that we should use

$$h_{\boldsymbol{\theta}}(\mathrm{x}) = \boldsymbol{\theta}^\top \mathrm{x}.$$

Then, given a training set $(\mathtt{X}, \mathrm{y})$, we find that the $\boldsymbol{\theta}$ maximizing $p(\mathrm{y} \mid \mathtt{X}; \theta)$ is $(\mathtt{X}^\top \mathtt{X})^{-1} \mathtt{X}^\top \mathrm{y}$.

Generalized linear models

Logistic regression as a GLM

Now consider the logistic regression setting. We have a two classes, 0 and 1.

If we assume $y \sim \mathrm{Bernoulli}(\phi)$ and take the GLM approach, we cast the Bernoulli distribution as a member of the exponential family, obtaining

$$\phi = \frac{1}{1+e^{-\eta}}.$$

We then try to predict the expectation of $T(y) = y$ given x:

$$\begin{aligned} h_{\boldsymbol{\theta}}(\mathrm{x}) &= E[y \mid \mathrm{x}; \boldsymbol{\theta}] \\ &= \phi \\ &= \frac{1}{1+e^{-\eta}} \\ &= \frac{1}{1+e^{-\boldsymbol{\theta}^\top \mathrm{x}}}. \end{aligned}$$

Try to justify each step here!

Generalized linear models

Logistic regression as a GLM

Now we understand why we take the logistic sigmoid

$$\frac{1}{1+e^{-\boldsymbol{\theta}^\top \mathrm{x}}}$$

as a model for $p(y=1 \mid \mathrm{x}; \boldsymbol{\theta})$ in logistic regression: it is the natural consequence of choosing a GLM to model y as a Bernoulli random variable depending on x.

Why should we care about all this? We already know how to do linear and logistic regression!!

The reason is that the GLM approach applies to any distribution and usually leads to elegant learning rules.

So when faced with a new learning problem, your baseline approach should be

1. Come up with a model for the conditional distribution of y given x.
2. Cast that conditional distribution as a member of the exponential family to determine what $\boldsymbol{\eta}$ is.
3. Replace $\boldsymbol{\eta}_i$ with $\boldsymbol{\theta}_i^\top \mathrm{x}$.
4. Come up with a procedure to maximize $\ell(\boldsymbol{\theta})$ for a training set.

Generalized linear models

Example: multinomial distribution

As an example, consider the generalization of the logistic regression problem to k classes, i.e., $\mathcal{Y} = \{1, 2, \ldots, k\}$.

The natural generalization of the Bernoulli distribution is the multinomial distribution with parameters $\phi_1, \ldots \phi_{k-1}$.

[We leave out the redundant $\phi_k = 1 - \sum_{i=1}^{k-1} \phi_i$.]

To model the multinomial as a member of the exponential family, there is a rather involved derivation. See Bishop (2006) or Ng's lecture notes for details.

Generalized linear models

Example: multinomial distribution

The upshot: we can obtain $\eta_i = \log \frac{\phi_i}{\phi_k}$, which can be inverted to obtain

$$\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_j}},$$

which is called the softmax function and is the multi-class generalization of the logistic sigmoid.

Generalized linear models

Example: multinomial distribution

Our prediction is thus

$$\begin{aligned} p(y = i \mid \mathrm{x}; \Theta) &= \phi_i \\ &= \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_j}} \\ &= \frac{e^{\boldsymbol{\theta}_i^\top \mathrm{x}}}{\sum_{j=1}^{k} e^{\boldsymbol{\theta}_j^\top \mathrm{x}}} \end{aligned}$$

Exercise: play with the softmax function to see how it transforms log odds ratios into a probability mass distribution.

See Bishop (2006) or Ng for the log likelihood function. As before, we can use gradient descent or Newton's method to find the optimal Θ.

Generalized linear models

Summary

To summarize the GLM approach:

- **Assumption 1**: the distribution $p(y \mid \mathrm{x}; \boldsymbol{\theta})$ is a member of the exponential family with natural parameter(s) $\boldsymbol{\eta}$.
- **Assumption 2**: our goal is to predict, given an input x, the expectation $E[T(y) \mid \mathrm{x}; \boldsymbol{\theta}]$. T is a transformation of y that comes from modeling $p(y \mid \mathrm{x}; \boldsymbol{\theta})$ as a member of the exponential family.
- **Assumption 3**: the natural parameter(s) of the distribution $\boldsymbol{\eta}$ are linear in x, i.e., $\eta_i = \boldsymbol{\theta}_i^\top \mathrm{x}$.

If you are willing to make these assumptions, you end up with a "concave" log likelihood function, which means that any local maximum is a global maximum.

A GLM should thus be a good first thing to try when you are faced with a machine learning problem you don't already have an algorithm for.

Outline

1. Linear regression
2. Classification
3. Logistic regression
4. Generalized linear models
5. Generative learning algorithms
6. Support vector machines

Generative learning algorithms

Generative vs. discriminative

So far, the methods we have tried attempt to learn $p(y \mid \mathrm{x})$ directly.

Given an input x, we try to map directly to the output y.

Any algorithm that does this is called a discriminative learning algorithm.

Another class of algorithms instead tries to model $p(\mathrm{x} \mid y)$ and $p(y)$.

Such methods are called generative learning algorithms.

Generative learning algorithms

Using Bayes' rule

If we can build a model of $p(\mathrm{x} \mid y)$ and $p(y)$, then Bayes' rule tells us that

$$p(y \mid \mathrm{x}) = \frac{p(\mathrm{x} \mid y)p(y)}{p(\mathrm{x})}.$$

Why don't we care about $p(\mathrm{x})$? Let's see...

First, generative models are most often used for classification, not regression.

In classification, y is discrete, so we have

$$p(\mathrm{x}) = \sum_i p(\mathrm{x} \mid y = y_i)p(y = y_i).$$

If y is continuous, we just have an integral instead of a sum.

This shows that if we can model $p(\mathrm{x} \mid y)$ and $p(y)$, we can obtain $p(y \mid \mathrm{x})$ without explicitly calculating $p(\mathrm{x})$.

Generative learning algorithms

Using Bayes' rule

We could calculate $p(\mathrm{x})$ directly. But usually, we want to know which y maximizes $p(y \mid \mathrm{x})$.

In this case, all we need to do is find

$$\begin{aligned} y^* &= \operatorname*{argmax}_y p(y \mid \mathrm{x}) \\ &= \operatorname*{argmax}_y \frac{p(\mathrm{x} \mid y)p(y)}{p(\mathrm{x})} \\ &= \operatorname*{argmax}_y p(\mathrm{x} \mid y)p(y) \end{aligned}$$

So to perform classification in the generative approach, all we need is models for $p(\mathrm{x} \mid y)$ and $p(y)$.

Let's consider the case where $p(\mathrm{x} \mid y)$ is a multivariate Gaussian.

A possible example would be $\mathcal{Y} = \{\mathrm{male}, \mathrm{female}\}$ and $\mathcal{X} = \mathbb{R}^2$ where the features are a person's height and weight.

The *n*-dimensional multivariate Gaussian distribution has

- a mean vector $\boldsymbol{\mu} \in \mathbb{R}^n$
- a covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$. We require $\Sigma \geq 0$ is symmetric and positive semidefinite.

The distribution is written $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and the density is

$$p(\mathrm{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathrm{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathrm{x}-\boldsymbol{\mu})}.$$

If $\mathrm{X} \sim \mathcal{N}(\boldsymbol{\mu}; \Sigma)$, then we can write

$$E[\mathrm{X}] = \int_{\mathrm{x}} \mathrm{x} p(\mathrm{x}; \boldsymbol{\mu}, \Sigma) d\mathrm{x} = \boldsymbol{\mu}$$

and

$$\mathrm{Cov}(\mathrm{X}) = E[(\mathrm{X} - E[\mathrm{X}])(\mathrm{X} - E[\mathrm{X}])^\top] = \Sigma.$$

See the handout on the multivariate Gaussian on the course Web site to get an intuition about interpreting the covariance matrix.

Generative learning algorithms

Gaussian discriminant analysis

Now, the GDA model for a 2-class problem:

$$\begin{aligned} y &\sim \mathrm{Bernoulli}(\phi) \\ \mathrm{x} \mid y = 0 &\sim \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0) \\ \mathrm{x} \mid y = 1 &\sim \mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1) \end{aligned}$$

We can then write the distributions:

$$\begin{aligned} p(y) &= \phi^y (1-\phi)^{1-y} \\ P(x \mid y = 0) &= \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathrm{x}-\boldsymbol{\mu}_0)^\top \Sigma_0^{-1} (\mathrm{x}-\boldsymbol{\mu}_0)} \\ P(x \mid y = 1) &= \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathrm{x}-\boldsymbol{\mu}_1)^\top \Sigma_1^{-1} (\mathrm{x}-\boldsymbol{\mu}_1)} \end{aligned}$$

Time to optimize!

$$
\begin{aligned}
\ell(\phi, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma_0, \Sigma_1) &= \log \prod_{i=1}^{m} p(\mathrm{x}^{(i)}, y^{(i)}; \phi, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma_0, \Sigma_1) \\
&= \log \prod_{i=1}^{m} p(\mathrm{x}^{(i)} \mid y^{(i)}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma_0, \Sigma_1) p(y^{(i)}; \phi)
\end{aligned}
$$

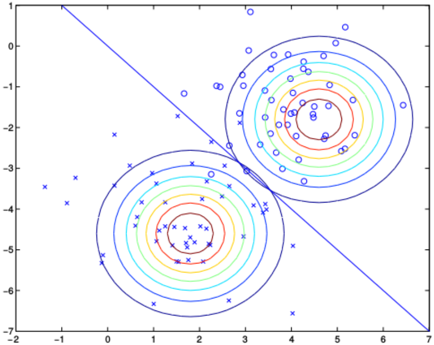
It turns out that if we assume $\Sigma_0 = \Sigma_1 = \Sigma$, we obtain the maximum likelihood estimates

$$\begin{aligned}
\phi &= \frac{1}{m} \sum_{i=1}^{m} 1\{y^{(i)} = 1\} \\
\boldsymbol{\mu}_0 &= \frac{\sum_{i=1}^{m} \delta[y^{(i)} = 0]\mathrm{x}^{(i)}}{\sum_{i=1}^{m} \delta[y^{(i)} = 0]} \\
\boldsymbol{\mu}_1 &= \frac{\sum_{i=1}^{m} \delta[y^{(i)} = 1]\mathrm{x}^{(i)}}{\sum_{i=1}^{m} \delta[y^{(i)} = 1]} \\
\Sigma &= \frac{1}{m} \sum_{i=1}^{m} (\mathrm{x}^{(i)} - \boldsymbol{\mu}_{y^{(i)}})(\mathrm{x}^{(i)} - \boldsymbol{\mu}_{y^{(i)}})^\top \qquad (2)
\end{aligned}$$

Note: I use $\delta[\psi]$ for a version of the Kronecker delta function that is 1 if ψ is true and 0 elsewhere.

Generative learning algorithms

Gaussian discriminant analysis



Ng, CS229 lecture note set #2

The assumption

$$\Sigma_0 = \Sigma_1 = \Sigma$$

turns out to be very convenient.

Exercise: Given the maximum likelihood estimates previously shown, try to find the set of points x for which

$$p(y = 1 \mid \mathrm{x}) = 0.5.$$

(It may help to note that this is where $p(\mathrm{x} \mid y = 1)p(y = 1) = p(\mathrm{x} \mid y = 0)p(y = 0)$).

Generative learning algorithms

Gaussian discriminant analysis

The GDA model is related to the logistic regression model.

Exercise: show that

$$p(y = 1 \mid \mathrm{x}; \phi, \Sigma, \mu_0, \mu_1) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \mathrm{x}}},$$

with $\boldsymbol{\theta}$ as a function of ϕ, Σ, $\boldsymbol{\mu}_0$, $\boldsymbol{\mu}_1$.

GDA and logistic regression have the same form but give different decision boundaries:

- GDA will be better (will require less training data to provide accurate predictions) if $p(\mathrm{x} \mid y)$ is in fact multivariate Gaussian or almost multivariate Gaussian.
- Logistic regression will probably be better if $p(\mathrm{x} \mid y)$ is definitely non-Gaussian or unknown.

Generative learning algorithms

Naive Bayes

GDA was an example of a generative classification method for problems in which $\mathcal{X} = \mathbb{R}^n$.

What if our features have discrete values?

Examples:

- Car buying behavior: $\mathcal{Y} = \{\text{buy}, \neg\text{buy}\}$, $\mathcal{X} = \left[\text{Size}, \text{Color}, \text{Gender}, \text{Age}\right]$, $\text{Size} = \{\text{small}, \text{medium}, \text{large}\}$; $\text{Color} = \{\text{red}, \text{blue}, \text{black}, \text{white}\}$; $\text{Gender} = \{\text{male}, \text{female}\}$; $\text{Age} = \{\text{0-18}, \text{19-39}, \text{30-39}, \text{40-59}, 60+\}$.
- Email spam filter: $\mathcal{Y} = \{\text{spam}, \neg\text{spam}\}$. $\mathcal{X} = \{0, 1\}^K$, with each variable $x_1, \ldots, x_k$ representing presence/absence of a particular word in a vocabulary.

Generative learning algorithms

Naive Bayes

Whatever $\mathcal{Y}$ and $\mathcal{X}$ are, a generative model needs a form for $p(y \mid \mathrm{x})$.

The simplest approach to these problems is to use the multinomial distribution over the set of possible outcomes for x.

Exercise: How many outcomes for x are there for the car buying and spam filter examples?

The full multinomial model will thus have $2L - 1$ parameters for $p(\mathrm{x} \mid y)$, where L is the number of outcomes for x, and 1 parameter for $p(y)$, assuming y is binary.

Is this practical for the car buying example? For the spam example?

Generative learning algorithms

Naive Bayes

Naive Bayes attempts to reduce the number of parameters required using the (very strong but very useful) assumption that the features are conditionally independent given y:

$$\begin{aligned} p(\mathrm{x} \mid y) &= p(x_1, x_2, \ldots, x_n \mid y) \\ &= p(x_1 \mid y) p(x_2 \mid x_1, y) \cdots p(x_n \mid x_1, x_2, \ldots, x_{n-1}, y) \\ &\approx \prod_{i=1}^{n} p(x_i \mid y) \end{aligned}$$

This model requires a set of parameters $\phi_{ij|y=1} = p(x_i = j \mid y = 1)$ and a set of parameters $\phi_{ij|y=0} = p(x_i = j \mid y = 0)$.

Note that if the variables x_i are binary, we only need two parameters $\phi_{i|y=1} = p(x_i = 1 \mid y = 1)$ and $\phi_{i|y=0} = p(x_i = 1 \mid y = 0)$.

Generative learning algorithms

Naive Bayes

For the case where x_i are binary, we get the joint likelihood

$$\mathcal{L}(\phi_y, \phi_{i|y=0}, \phi_{i|y=1}) = \prod_{i=1}^{m} p(\mathrm{x}^{(i)}, y^{(i)}).$$

Exercise: verify that maximizing $\mathcal{L}$ gives maximum likelihood estimates

$$\begin{aligned}
\phi_{j|y=1} &= \frac{\sum_{i=1}^{m} \delta[x_j^{(i)} = 1 \wedge y^{(i)} = 1]}{\sum_{i=1}^{m} \delta[y^{(i)} = 1]} \\
\phi_{j|y=0} &= \frac{\sum_{i=1}^{m} \delta[x_j^{(i)} = 1 \wedge y^{(i)} = 0]}{\sum_{i=1}^{m} \delta[y^{(i)} = 0]} \\
\phi_y &= \frac{\sum_{i=1}^{m} \delta[y^{(i)} = 1]}{m}
\end{aligned}$$

If n is large, there may be some features that do not appear in all combinations with every class.

Examples:

- The term “viagra” might occur only in spam emails.
- In a given period of time, there might not be any customers 18 years or younger who bought a car.

Suppose we are then faced with an email x with the term "viagra" or a customer x whose age is 0-18.

Do we predict $p(\text{spam} \mid \text{x}) = 1$ and $p(\text{buy} \mid \text{x}) = 0$?

Generative learning algorithms

Naive Bayes

Laplace smoothing avoids 0-probability issues by adding one pseudo-example to the dataset:

$$\begin{aligned}
\phi_{j|y=1} &= \frac{\sum_{i=1}^{m} \delta[x_j^{(i)} = 1 \wedge y^{(i)} = 1] + 1}{\sum_{i=1}^{m} \delta[y^{(i)} = 1] + 2} \\
\phi_{j|y=0} &= \frac{\sum_{i=1}^{m} \delta[x_j^{(i)} = 1 \wedge y^{(i)} = 0] + 1}{\sum_{i=1}^{m} \delta[y^{(i)} = 0] + 2}
\end{aligned}$$

This is for binary features (a multi-variate Bernoulli event model).

See Ng's lecture notes for more information on the multinomial event model.

Generative learning algorithms

Naive Bayes

So! Reminder:

- If you have continuous $\mathcal{X}$ and continuous $\mathcal{Y}$ your first go-to model is linear regression. Consider non-linear transformations of the inputs.
- If you have continuous $\mathcal{X}$ and discrete $\mathcal{Y}$ but don't know much about $p(\mathrm{x} \mid y)$, your first go-to model is logistic or softmax regression, or you can come up with a new GLM from scratch.
- If you have continuous $\mathcal{X}$ and discrete $\mathcal{Y}$ and know something about $p(\mathrm{x} \mid y)$, model the distribution accurately, as a Gaussian (GDA) or build a new generative model from scratch.
- If you have discrete $\mathcal{X}$ and discrete $\mathcal{Y}$ you should probably start with naive Bayes then build up from there.

That's it for generative models. Next we'll return to discriminative models with kernel methods and SVMs.

Outline

1. Linear regression
2. Classification
3. Logistic regression
4. Generalized linear models
5. Generative learning algorithms
6. Support vector machines

Support vector machines

Introduction

Thus far, we have studied machine learning models that are relatively easy to analyze and are optimal, when the assumptions they make are satisfied.

For example, GDA assumes the conditional distribution $p(\mathrm{x} \mid y)$ is a multivariate Gaussian.

What happens when the assumptions are violated?

Next we look at support vector machines (SVMs), which are more flexible and widely applicable than the method's we've looked at thus far.

Although deep neural networks have received the most attention very recently, many still believe that the SVM is the best "off-the-shelf" supervised classifier.

SVMs are based on the idea of maximum margin classification.

Consider logistic regression, in which we model $p(y = 1 \mid \mathrm{x}; \boldsymbol{\theta})$ by $h_{\boldsymbol{\theta}}(\mathrm{x}) = g(\boldsymbol{\theta}^\top \mathrm{x})$.

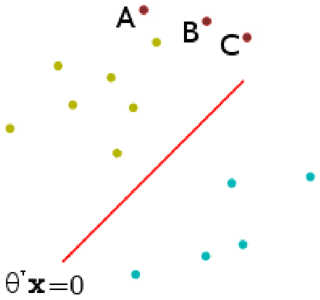
The logistic regression classification rule is

$$y^{pred}(\mathrm{x}) = \begin{cases} 1 & h_{\boldsymbol{\theta}}(\mathrm{x}) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Informally, our goal should be to find $\boldsymbol{\theta}$ such that $\boldsymbol{\theta}^\top \mathrm{x}^{(i)} \gg 0$ for all i with $y^{(i)} = 1$ and $\boldsymbol{\theta}^\top \mathrm{x}^{(i)} \ll 0$ for all i with $y^{(i)} = 0$.

Support vector machines

Decision boundaries



Suppose the yellow points are training data from class 1 and the cyan points are training data from class 0.

$\boldsymbol{\theta}^\top \mathrm{x} = 0$ is a separating hyperplane or decision boundary between the two classes.

Point A is furthest from the decision boundary. We should be confident to predict class 1 for point A.

Point C is more ambiguous.

This observation will lead to the principle of maximizing the margin.

Support vector machines

Model and notation

Next, we'll modify our model a little.

Rather than classes 0 and 1, we'll make them -1 and $+1$.

Rather than $\boldsymbol{\theta}^\top \mathrm{x}$ with implicit $x_0 = 1$, we write $\mathrm{w}^\top \mathrm{x} + b$ replacing θ_0 by b, making the intercept term explicit.

The classifier will be

$$h_{\mathrm{w},b}(\mathrm{x}) = g(\mathrm{w}^\top \mathrm{x} + b).$$

We use the perceptron rule

$$g(z) = \begin{cases} 1 & z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

We have the structure of the model. What about the cost/objective function?

First, we write the functional margin of the parameters w, b wih respect to training example i:

$$\hat{\gamma}^{(i)} = y^{(i)}(\mathrm{w}^\top \mathrm{x} + b).$$

Convince yourself that $\hat{\gamma}^{(i)}$ is

- 0 for training points on the decision boundary,
- negative for incorrectly classified points,
- positive for correctly classified points, and
- larger for correctly classified points further from the boundary.

Support vector machines

Should we maximize the margin?

So $\hat{\gamma}^{(i)}$ seems like a good thing to maximize.

However, there is one problem with $\hat{\gamma}^{(i)}$ as defined so far.

Scaling w and b by an arbitrary factor α does not change the decision boundary but does scale the resulting margin by α.

That means $\hat{\gamma}^{(i)}$ could be made artificially large by scaling with a large α.

We will avoid this by constraining w to be length 1. Instead of (w, b), we will use $(\mathrm{w}/\|\mathrm{w}\|, b/\|\mathrm{w}\|)$ when calculating the margin.

This gives us the geometric margin

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{\mathrm{w}}{\|\mathrm{w}\|} \right)^{\top} \mathrm{x}^{(i)} + \frac{b}{\|\mathrm{w}\|} \right)$$

Support vector machines

Maximize the minimum margin

What should our goal actually be?

We could think of logistic regression as attempting to maximize a kind of average margin.

However, SVMs attempt to maximize the minimum margin.

So we will define the functional margin with respect to an entire training set $S = \{(\mathrm{x}^{(i)}, y^{(i)})\}_{i \in 1..m}$ as

$$\gamma = \min_{i \in 1..m} \gamma^{(i)},$$

i.e., the smallest of the margins of the individual training examples.

Support vector machines

The optimization problem

OK! Now we know that we'd like to find w, b according to

$$\begin{array}{rl} \max_{\gamma,\mathrm{w},b} & \gamma \\ \text{subject to} & y^{(i)}(\mathrm{w}^\top \mathrm{x}^{(i)} + b) \geq \gamma, i \in 1..m \\ & \|\mathrm{w}\| = 1. \end{array}$$

Only problem: the constraint $\|\mathrm{w}\| = 1$ is not linear, as would be required by most optimizers.

Think of the constraint as saying we must find a point on the unit sphere that maximizes γ. Tricky!

Support vector machines

The optimization problem

First step to improve the formulation of the optimization problem: move the constraint into the objective:

$$\begin{array}{rl} \max_{\gamma,\mathrm{w},b} & \frac{\hat{\gamma}}{\|\mathrm{w}\|} \\ \text{subject to} & y^{(i)}(\mathrm{w}^\top \mathrm{x}^{(i)} + b) \geq \hat{\gamma}, i \in 1..m. \end{array}$$

The good: the constraints are now all linear. That is good for standard optimizers.

New problem: now the objective function $\hat{\gamma}/\|\mathrm{w}\|$ is itself non-convex.

Can we fix it? Well, first let's note that the constraint $\|\mathrm{w}\| = 1$ is fairly arbitrary. Actually, any constraint on w that prevents the optimizer from scaling w to get a bigger margin is enough.

Other constraints such as $w_1 = 10$ or $\sqrt{w_1^2 + w_2^2} = 3$ would be equally good.

Support vector machines

The optimization problem

The particular constraint on w, b that we will use is a little weird:

$$\hat{\gamma} = 1.$$

That is, we will scale w, b so that the distance of the training point(s) closest to the hyperplane $\mathrm{w}^\top \mathrm{x} + b = 0$ is 1.

Replacing $\hat{\gamma}$ with 1 in our optimization problem, we can obtain

$$\begin{array}{rl} \min_{\gamma, \mathrm{w}, b} & \frac{1}{2}\|\mathrm{w}\|^2 \\ \text{subject to} & y^{(i)}(\mathrm{w}^\top \mathrm{x}^{(i)} + b) \geq 1, i \in 1..m. \end{array}$$

Convince yourself that this is a convex optimization problem with linear constraints that "rule out" parts of the parameter space as candidate solutions.

Support vector machines

Quadratic programming problem

So! If we want to find the optimal margin classifier for a linearly separable data set $\{(y^{(i)}, \mathrm{x}^{(i)})\}_{i \in 1..m}$, all we need to do is use a quadratic programming solver to find w, b satisfying

$$\begin{array}{rl} \min_{\gamma, \mathrm{w}, b} & \frac{1}{2}\|\mathrm{w}\|^2 \\ \text{subject to} & y^{(i)}(\mathrm{w}^\top \mathrm{x}^{(i)} + b) \geq 1, i \in 1..m. \end{array}$$

See the formalization of the general quadratic programming problem on the next page...

Support vector machines

Quadratic programming problem

The (general) quadratic programming problem

Given

- $\mathbf{c} \in \mathbb{R}^n$
- $\mathtt{Q} \in \mathbb{R}^{n \times n}$
- $\mathtt{A} \in \mathbb{R}^{m \times n}$
- $\mathbf{b} \in \mathbb{R}^m$

find

$$\mathbf{x}^* \in \mathbb{R}^n = \begin{array}{ll} \operatorname{argmin}_{\mathbf{x}} \frac{1}{2}\mathbf{x}^\top \mathtt{Q}\mathbf{x} + \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad \mathtt{A}\mathbf{x} \leq \mathbf{b} \end{array}$$

[Check that our problem can be cast as a QP problem.]

There are many standard QP solvers.

For example, in Python, try `cvxopt`:

- `sudo pip3 install cvxopt`

See the sample Jupyter notebook on the course home page.

Support vector machines

The dual optmization problem

Turns out that although we could stop here, QP is not efficient in high dimensional spaces.

We will see later that we want to build SVMs in high dimensional feature spaces in order to get interesting non-linear classification boundaries in the original attribute space.

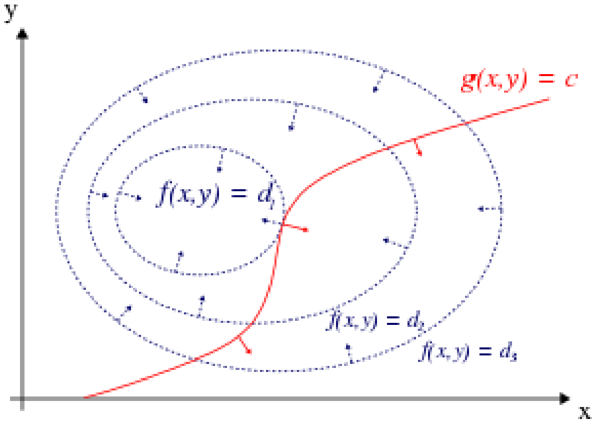
So instead of throwing our direct SVM problem at a QP solver, we will reformulate the problem in a way that it can be solved efficiently in high dimensional spaces.

This requires a discussion of Lagrangian optimization.

Support vector machines

Lagrangian optmization

Let's consider a 2-dimensional example where we want to maxmimize $f(x, y)$ subject to $g(x, y) = 0$:



https://en.wikipedia.org/wiki/Lagrange_multiplier

Support vector machines

Lagrangian optmization

We introduce a new variable λ called a Lagrange multiplier and consider the Lagrange function or Lagrangian

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda g(x, y).$$

It turns out that if (x^*, y^*) is a maximum of $f(x, y)$ satisfying $g(x^*, y^*) = 0$, then there exists a λ^* such that (x^*, y^*, λ^*) is a stationary point of $\mathcal{L}(x, y, \lambda)$.

That just means (x^*, y^*, λ^*) will satisfy

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial \lambda} = 0.$$

OK! So if we want to find the constrained maximum of $f(x,y)$ subject to $g(x,y)=0$, we just need to

- Write the Lagrangian.
- Set the derivatives of the Lagrangian to 0.
- Solve the resulting system of equations.
- Check whether the solution is a maximum or not.

A constrained minimum would be found in the same way, with a check for a minimum (see next page).

Support vector machines

Lagrangian optmization

Hessian analysis for an unconstrained optimization problem

A minimum of $f(\mathrm{w})$ will have a Hessian $\mathtt{H}_f$ with all positive eigenvalues at w^*, and a maximum will have a Hessian with all negative eigenvalues.

Hessian analysis for a constrained optimization problem

For constrained optimization problems, critical points of the Lagrangian are always saddle points ($\mathtt{H}_{\mathcal{L}}$ will have a mix of negative and positive eigenvalues).

For m constraints and n total variables including the Lagrange multipliers, we examine the determinants of the $2m+1$th to nth principal minors of $\mathtt{H}_{\mathcal{L}}$. If the critical point is a minimum, all of the minors' determinants will have a sign of $(-1)^m$. A maximum will have alternating signs of the minors' determinants!

It would be worthwhile to do a simple example at this point to make sure we've got it.

Suppose we have $f(\mathrm{w}) = \|\mathrm{w}\|^2$ and $g(\mathrm{w}) = w_1 + w_2 - 4$.

Go ahead and draw a visualization of $f(\cdot)$ and $g(\cdot)$. You should be able to determine by inspection the optimal w^* (a minimum).

Then try to find w^* using the Lagrangian and verify it.

Support vector machines

Generalized Lagrangian optmization

Next, we generalize the Lagrangian notion to an arbitrary parameter vector and the case where we have inequality constraints in addition to the equality constraints.

Following Ng's lecture notes, set 3, we aim to solve the optimization problem

$$\begin{array}{rl} \min_{\mathrm{w}} & f(\mathrm{w}) \\ \text{s.t.} & g_i(\mathrm{w}) \leq 0, i \in 1..k \\ & h_i(\mathrm{w}) = 0, i \in 1..l \end{array}$$

We write the generalized Lagrangian

$$\mathcal{L}(\mathrm{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathrm{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathrm{w}) + \sum_{i=1}^{l} \beta_i h_i(\mathrm{w})$$

Support vector machines

Lagrangian duality

Turns out there are a couple ways to solve the optimization problem using the generalized Lagrangian.

First: the primal problem:

$$\begin{aligned}\theta_{\mathcal{P}}(\mathrm{w}) &= \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i \geq 0} \mathcal{L}(\mathrm{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\ &= \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i \geq 0} f(\mathrm{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathrm{w}) + \sum_{i=1}^{l} \beta_i h_i(\mathrm{w})\end{aligned}$$

Convince yourself that if w violates any of our constraints, then $\theta_{\mathcal{P}}(\mathrm{w}) = \infty$, and if w satisfies all the constraints, then $\theta_{\mathcal{P}}(\mathrm{w}) = f(\mathrm{w})$.

So! If you convinced yourself of the fact on the previous page, we can now say that the w satisfying

$$\min_{\mathrm{w}} \theta_{\mathcal{P}}(\mathrm{w}) = \min_{\mathrm{w}} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i \geq 0} \mathcal{L}(\mathrm{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

solves the original optimization problem.

We call $p^* = \min_{\mathrm{w}} \theta_{\mathcal{P}}(\mathrm{w})$ the value of the primal problem.

It turns out the primal problem is sometimes not as easy to solve as another problem, the dual problem, that has the same solution under certain conditions.

So to get to a problem that might be easier to solve, consider the dual optimization problem in which we let

$$\theta_{\mathcal{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathrm{w}} \mathcal{L}(\mathrm{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}).$$

All we've done is swapped the order of the minimization and maximization.

The dual optimization problem is

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\beta} : \alpha_i \geq 0} \theta_{\mathcal{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \max_{\boldsymbol{\alpha}, \boldsymbol{\beta} : \alpha_i \geq 0} \min_{\mathrm{w}} \mathcal{L}(\mathrm{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}).$$

The value of the dual problem's objective is $d^* = \max_{\boldsymbol{\alpha}, \boldsymbol{\beta} : \alpha_i \geq 0} \theta_{\mathcal{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$

Support vector machines

Lagrangian duality

It is always true that

$$d^* = \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\geq 0} \min_{\mathrm{w}} \mathcal{L}(\mathrm{w},\boldsymbol{\alpha},\boldsymbol{\beta}) \leq \min_{\mathrm{w}} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\geq 0} \mathcal{L}(\mathrm{w},\boldsymbol{\alpha},\boldsymbol{\beta}) = p^*.$$

Under certain circumstances, it will also be the case that

$$d^* = p^*,$$

which would mean we can solve the dual problem instead of the primal problem if we so choose.

What are the conditions under which $p^* = d^*$? It turns out that if

- f is convex[3]
- The g_i's are convex
- The h_i's are affine
- The g_i's are strictly feasible (there is a w such that $g_i(\mathrm{w}) < 0$ for all i)

If these conditions hold, ... (see next page)

[3] Loosely speaking, if we consider the points w on a straight line between w_1 and w_2, $f(\mathrm{w})$ is always below the line.

Support vector machines

Lagrangian duality

If the conditions on the previous page are met, there must exist $\mathrm{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ so that

- w^* is the solution to the primal problem,
- $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ are the solution to the dual problem,
- $p^* = d^* = \mathcal{L}(\mathrm{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$, and
- $\mathrm{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ satisfy the Karush-Kuhn-Tucker (KKT) conditions[4] (next page)!

The KKT conditions will allow us to easily find the optimal $\mathrm{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$.

[4] First discovered by Karush in a 1939 Master's thesis, then rediscovered by Kuhn and Tucker in 1951.

Support vector machines

Lagrangian duality

If the assumptions are met, then any $\mathrm{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ satisfying the KKT conditions are solutions to both the primal and dual problem. The conditions:

$$\begin{aligned}
\frac{\partial}{\partial w_i}\mathcal{L}(\mathrm{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) &= 0, i \in 1..n \\
\frac{\partial}{\partial \beta_i}\mathcal{L}(\mathrm{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) &= 0, i \in 1..l \\
\alpha_i^* g_i(\mathrm{w}^*) &= 0, i \in 1..k \\
g_i(\mathrm{w}^*) &\leq 0, i \in 1..k \\
\alpha_i^* &\geq 0, i \in 1..k
\end{aligned}$$

Pay attention to the 3rd condition. This will enable us to show that the SVM has only a few support vectors.

OK, now we know how to write the primal and dual optimization problems for SVMs.

Previously we had

$$\begin{array}{rl} \min_{\gamma,\mathrm{w},b} & \frac{1}{2}\|\mathrm{w}\|^2 \\ \text{subject to} & y^{(i)}(\mathrm{w}^\top \mathrm{x}^{(i)} + b) \geq 1, i \in 1..m. \end{array}$$

To put this into the dual Lagrangian framework, we write the constraints as

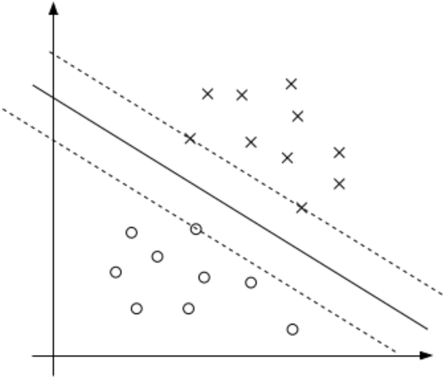
$$g_i(\mathrm{w}) = -y^{(i)}(\mathrm{w}^\top \mathrm{x}^{(i)} + b) + 1 \leq 0$$

i.e., one constraint for each training example.

From the KKT conditions, we will have $\alpha_i > 0$ only for training examples with functional margin exactly 1, where $g_i(\mathrm{w}) = 0$.

Support vector machines

Dual optimization problem for SVMs



Ng, CS229 lecture notes set 3

The points on the dashed lines are the ones for which $g_i(\mathrm{w}) = 0$ and $\alpha_i > 0$. They are called the support vectors.

Support vector machines

Dual optimization problem for SVMs

Next let's write the generalized Lagrangian for our optimization problem:

$$\mathcal{L}(\mathrm{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathrm{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}(\mathrm{w}^\top \mathrm{x}^{(i)} + b) - 1 \right]$$

There are only α_i's not β_i's because the problem only has inequality constraints, not equality constraints.

Support vector machines

Dual optimization problem for SVMs

For the dual optimization problem, we first minimize $\mathcal{L}(\mathrm{w}, b, \boldsymbol{\alpha})$ with respect to w and b, for fixed $\boldsymbol{\alpha}$.

To find the w and b, we take the partials and set to 0:

$$\nabla_{\mathrm{w}} \mathcal{L}(\mathrm{w}, b, \boldsymbol{\alpha}) = \mathrm{w} - \sum_{i=1}^{m} \alpha_i y^{(i)} \mathrm{x}^{(i)} = 0$$

from which we can obtain

$$\mathrm{w} = \sum_{i=1}^{m} \alpha_i y^{(i)} \mathrm{x}^{(i)}.$$

For b we obtain the equation

$$\frac{\partial}{\partial b} \mathcal{L}(\mathrm{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

This doesn't tell us anything about b, note, but we'll solve that problem shortly.

Support vector machines

Dual optimization problem for SVMs

Now that we have a definition of w (previous slide), we plug into the Langrangian, obtaining

$$\mathcal{L}(\mathrm{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (\mathrm{x}^{(i)})^\top \mathrm{x}^{(j)} - b \sum_{i=1}^{m} \alpha_i y^{(i)}.$$

Using the result from $\partial \mathcal{L} / \partial b$, we know the last term is 0, reducing us to

$$\mathcal{L}(\mathrm{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (\mathrm{x}^{(i)})^\top \mathrm{x}^{(j)}.$$

Next, for the dual optimization problem, now that we've written down the equivalent of $\theta_{\mathcal{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$, our job is to now maximize this expression w.r.t. $\boldsymbol{\alpha}$.

We'll write $\theta_{\mathcal{D}}$ as W for the specific case of the SVM:

$$\begin{array}{rl} \max_{\boldsymbol{\alpha}} & W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \left\langle \mathrm{x}^{(i)}, \mathrm{x}^{(j)} \right\rangle \\ \text{such that} & \alpha_i \geq 0, i \in 1..m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{array}$$

where $\langle \cdot, \cdot \rangle$ is the inner product.

If the assumptions for $p^* = d^*$ hold (verify that they do), then we can use the KKT conditions to solve this dual problem instead of solving the primal problem.

Support vector machines

Dual optimization problem for SVMs

Once we solve the dual problem for w to obtain w^*, we have the orientation of our maximum margin hyperplane, and all that remains is to find b, the bias or signed distance of the hyperplane to the origin.

$$b^* = -\frac{\max_{i:y^{(i)}=-1} \mathrm{w}^{*\top}\mathrm{x}^{(i)} + \min_{i:y^{(i)}=1} \mathrm{w}^{*\top}\mathrm{x}^{(i)}}{2}$$

You should be able to verify this easily.

Note that another way to calculate b (not knowing w) is

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{i \in \mathcal{S}} \left(y^{(i)} - \sum_{j \in \mathcal{S}} \alpha_j y^{(j)} \left\langle \mathrm{x}^{(i)}, \mathrm{x}^{(j)} \right\rangle \right),$$

where $\mathcal{S}$ is the set of indexes of the support vectors $\{i \in 1..m \mid \alpha_i > 0\}$ and $N_{\mathcal{S}}$ is the size of that set.

Support vector machines

Prediction using SVM

Once we have the optimal w and b, we would make the prediction

$$h_{\mathrm{w},b}(\mathrm{x}) = \begin{cases} +1 & \mathrm{w}^\top \mathrm{x} + b \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

However, it is important to note that

$$\begin{aligned} \mathrm{w}^\top \mathrm{x} + b &= \left(\sum_{i=1}^{m} \alpha_i y^{(i)} \mathrm{x}^{(i)} \right)^\top \mathrm{x} + b \\ &= \sum_{i=1}^{m} \alpha_i y^{(i)} \left\langle \mathrm{x}^{(i)}, \mathrm{x} \right\rangle + b. \end{aligned}$$

which shows that our prediction for x is a weighted summation of inner products with the support vectors (the $\mathrm{x}^{(i)}$ for which $\alpha_i \neq 0$).

Support vector machines

Coming next

Well, now we've understood very well the optimization problem for support vector machines in the linearly separable case.

As of yet, we haven't discussed:

- The case where the training data are not linearly separable.
- How we can turn the linear SVM into a nonlinear classifier.
- An efficient optimization algorithm for the α_i's

These issues are coming next!

We'll introduce kernels as a way to obtain nonlinear transformations of the input data then perform linear classification in the transformed space.

We'll introduce regularization to allow for non-separable data and have some resilience to outliers.

We'll look at how a new optimization algorithm, coordinate ascent, can be adapted to the case of our dual optimization problem.

"Raise your hand if this makes sense!"

Support vector machines

Kernels: feature mapping

When we learned linear regression, we found that it was a simple matter to generate new features that were nonlinear transforms of the original attributes.

Let's make some terminology:

- Let's call the original inputs for x in the training set the input attributes for the problem.
- We'll call the newly generated values the input features.
- Let ϕ denote the feature mapping that maps attributes to features.

Example: if we mapped an original scalar feature x (such as height) to x, x^2, and x^3 to obtain a cubic hypothesis, we would write the mapping

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}.$$

Support vector machines

Kernels

To allow for mapping in the SVM optimization algorithm, we simply replace x with $\phi(\mathrm{x})$.

Where we have inner products $\langle \mathrm{x}, \mathrm{z} \rangle$, we will have instead $\langle \phi(\mathrm{x}), \phi(\mathrm{z}) \rangle$

New term: given a feature mapping ϕ, the corresponding kernel is

$$K(\mathrm{x}, \mathrm{z}) = \phi(\mathrm{x})^\top \phi(\mathrm{z}).$$

Now we can replace $\langle \mathrm{x}, \mathrm{z} \rangle$ with $K(\mathrm{x}, \mathrm{z})$.

Support vector machines

Kernels: prediction using kernels

Note: thus far we haven't done much!

We have just added a mapping function and rewritten the SVM inference algorithm in terms of a kernel function.

Instead of computing

$$\sum_{i=1}^{m} \alpha_i y^{(i)} \left\langle \mathrm{x}^{(i)}, \mathrm{x} \right\rangle + b,$$

then predicting based on the sign of the result, we have rewritten the computation as

$$\sum_{i=1}^{m} \alpha_i y^{(i)} K(\mathrm{x}^{(i)}, \mathrm{x}) + b.$$

Support vector machines

Kernels: efficiency

What is the benefit of rewriting our inference rule in terms of kernels?

It turns out that sometimes computing the kernel is more efficient than transforming the input attributes into feature space then computing a dot product.

Ng's example: consider the kernel

$$K(\mathrm{x}, \mathrm{z}) = (\mathrm{x}^\top \mathrm{z})^2.$$

What ϕ does this correspond to?

Support vector machines

Kernels: efficiency

$$
\begin{aligned}
K(\mathrm{x},\mathrm{z}) &= (\mathrm{x}^\top \mathrm{z})^2 \\
&= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_j z_j\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j \\
&= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j)
\end{aligned}
$$

which you can verify corresponds to $\phi(\mathrm{x})^\top \phi(\mathrm{z})$ with (using $n = 3$ for example)

$$\phi(\mathrm{x}) = \begin{bmatrix} x_1x_1 & x_1x_2 & x_1x_3 & x_2x_1 & x_2x_2 & x_2x_3 & x_3x_1 & x_3x_2 & x_3x_3 \end{bmatrix}^\top .$$

Support vector machines

Kernels: efficiency

In the example, computing $\phi(\mathrm{x})$ takes $O(n^2)$ time, but computing $K(\mathrm{x}, \mathrm{z})$ takes $O(n)$ time!

So we see that kernels can be more efficient than dot products in feature space.

Support vector machines

Kernels: interpretation

Kernels are efficient. Good. Then how can we think about what kernel to use for a particular problem?

Consider that a dot product to a certain extent measures how similar two vectors are:

- The dot product is 0 for orthogonal vectors
- The dot product grows as we rotate the vectors toward each other.

We can use this intuition to generate ideas for different kinds of kernels.

Support vector machines

Kernels: interpretation

For example, the most popular kernel besides the linear kernel is the Gaussian kernel or radial basis function (RBF) kernel

$$K(\mathrm{x}, \mathrm{z}) = e^{-\frac{\|\mathrm{x}-\mathrm{z}\|^2}{2\sigma^2}}$$

which is 1 when $\mathrm{x} = \mathrm{z}$ and decreases as they move apart.

It turns out that this choice of $K(\cdot, \cdot)$ is a valid kernel, although it corresponds to a ϕ transforming x into an infinite dimensional feature space!!

Support vector machines

Kernels: validity

OK so we have two ways to design kernels:

- Come up with a ϕ we think useful, and compute $K(\cdot,\cdot)$ in the feature space directly.
- Come up with a $K(\cdot,\cdot)$ we think useful, verify that it is a valid kernel, and just use it without ever computing $\phi(\mathrm{x})$ at all.

So we need to know what makes a kernel valid.

Mercer's theorem (roughly): $K(\cdot,\cdot)$ is a valid kernel iff the kernel matrix or Gram matrix

$$\mathtt{K} = \begin{bmatrix} K(\mathrm{x}^{(1)},\mathrm{x}^{(1)}) & K(\mathrm{x}^{(1)},\mathrm{x}^{(2)}) & \cdots \\ K(\mathrm{x}^{(2)},\mathrm{x}^{(1)}) & K(\mathrm{x}^{(2)},\mathrm{x}^{(2)}) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

is symmetric positive semidefinite for any training set $(\mathrm{x}^{(1)},\ldots,\mathrm{x}^{(m)})$.

Support vector machines

Kernels: validity

See Hastie et al., Bishop, or Ng for (a lot) more detail on valid kernels, proofs of positive semidefiniteness of the kernel matrix, and so on!

If we're doing research on kernel methods, we'll need those details.

Also, we can apply the kernel trick to any machine learning algorithm, not just SVMs, to get a "kernelized" version of that algorithm.

So, like the GLM trick, the kernel trick is a kind of generator of new machine learning algorithms.

If we are primarily a user, however, we can just try the various kernel options provided by a SVM library.

Support vector machines

Non-separable data

Recall what allowed us to formulate the SVM maximum margin problem as a QP problem: linear separability made the constraints satisfiable.

Now suppose the training data are not linearly separable and/or contain outliers.

We can extend our objective function introducing slack variables:

$$\begin{array}{rl} \min_{\gamma,\mathrm{w},b,\boldsymbol{\xi},\mathrm{r}} & \frac{1}{2}\|\mathrm{w}\|^2 + C\sum_{i=1}^{m}\xi_i \\ \text{such that} & y^{(i)}(\mathrm{w}^\top \mathrm{x}^{(i)} + b) \geq 1 - \xi_i, i \in 1..m \\ & \xi_i \geq 0, i \in 1..m \end{array}$$

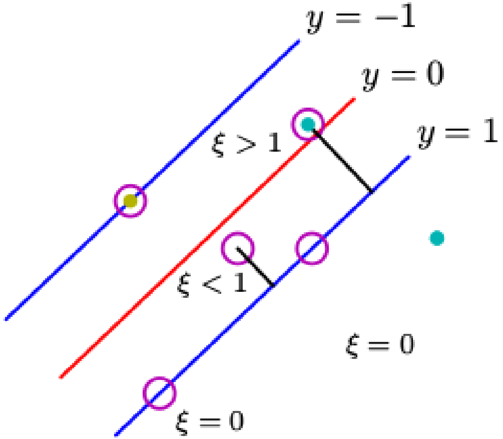
This is still a QP problem (now we have a linear term in the objective function and a modified set of linear inequality constraints).

However, to allow the use of the kernel trick, we'll use the dual Lagrangian form of the optimization again.

Support vector machines

Non-separable data

Illustration of slack variables:



Bishop (2006), Figure 7.3

Support vector machines

Non-separable data

The Lagrangian is

$$\begin{aligned}\mathcal{L}(\mathrm{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathrm{r}) = \quad & \tfrac{1}{2}\mathrm{w}^\top \mathrm{w} + C \textstyle\sum_{i=1}^m \xi_i - \\ & \textstyle\sum_{i=1}^m \alpha_i \left[y^{(i)}(\mathrm{x}^{(i)\top}\mathrm{w} + b) - 1 + \xi_i \right] - \textstyle\sum_{i=1}^m r_i \xi_i .\end{aligned}$$

Taking derivatives with respect to w and b, setting them to 0, substituting them in, and simplifying, we can obtain the dual optimization problem

$$\begin{aligned}\max_{\boldsymbol{\alpha}} \quad & W(\boldsymbol{\alpha}) = \textstyle\sum_{i=1}^m \alpha_i - \tfrac{1}{2} \textstyle\sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \left\langle \mathrm{x}^{(i)}, \mathrm{x}^{(j)} \right\rangle \\ \text{such that} \quad & 0 \leq \alpha_i \leq C, i \in 1..m \\ & \textstyle\sum_{i=1}^m \alpha_i y^{(i)} = 0.\end{aligned}$$

Support vector machines

Non-separable data

Amazingly, the ξ_i and r_i drop out of the optimization.

This leaves the original objective function for the α_i with an a new constraint that $\alpha_i \leq C$.

So if we can learn the α_i for the separable case, all we need to do for outliers and overlapping data is put a limit on α_i.

Larger C makes the final model less tolerant toward training points crossing the decision boundary.

Smaller C will allow more training points to cross the boundary in order to get a good overall fit to the data.

The KKT dual-complementarity conditions give the following:

$$\begin{aligned}
\alpha_i = 0 &\implies y^{(i)}(\mathrm{w}^\top \mathrm{x}^{(i)} + b) \geq 1 \\
\alpha_i = C &\implies y^{(i)}(\mathrm{w}^\top \mathrm{x}^{(i)} + b) \leq 1 \\
0 < \alpha_i < C &\implies y^{(i)}(\mathrm{w}^\top \mathrm{x}^{(i)} + b) = 1
\end{aligned}$$

So the initial assumption of linearly separable data turns out not to be a big deal!

One thing to note, however: the parameter C is a free parameter or hyperparameter that must be set to some reasonable value, usually through a cross validation experiment (more on this later).

So now all we need is an efficient algorithm to solve the dual optimization problem.

We begin with the coordinate ascent algorithm.

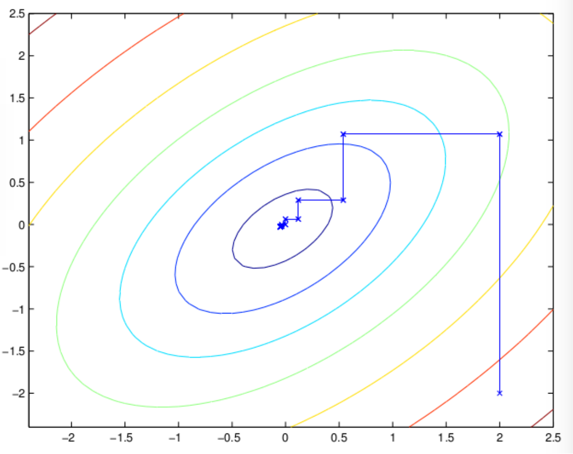
Suppose the goal is to solve the (unconstrained for now) optimization problem

$$\max_{\boldsymbol{\alpha}} W(\alpha_1, \alpha_2, \ldots, \alpha_m).$$

Rather than gradient ascent or Newton's method, we might now try to optimize the parameters one by one:

```
While not converged do
    for $i = 1, \ldots, m$ do
        $\alpha_i \leftarrow \mathrm{argmax}_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$
```

Visualization of coordinate ascent:



Ng, CS 229 lecture notes, set 3

To adapt coordinate ascent to the SVM dual optimization problem, consider that we have a set of α_i's that statisfy the constraints.

Can we hold $\alpha_2, \ldots$ fixed while optimizing with respect to α_1?

No, because the constraint $\sum_{i=1} \alpha_i y^{(i)} = 0$ means that if we increase (decrease) an α_i with $y^{(i)} = -1$ we have to make a correspnding decrease (increase) in an α_i with $y^{(i)} = -1$ or a corresponding increase (decrease) in an α_i with $y^{(i)} = +1$.

Support vector machines

Optimizing the dual

The SMO algorithm by John Platt (1998) therefore updates pairs of α_i's at a time:

```
While not converged
    Select a pair $(\alpha_i, \alpha_j)$
    Optimize $W(\boldsymbol{\alpha})$ with respect to $\alpha_i$ and $\alpha_j$.
```

The convergence criterion is whether the KKT conditions are satisfied within some tolerance.

Suppose we've selected α_1 and α_2 as the parameters to optimize on one iteration.

Clearly, due to the constraint that $\sum_i \alpha_i y^{(i)} = 0$, we can write

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = -\sum_{i=3}^{m} \alpha_i y^{(i)},$$

or, equivalently for some constant ζ:

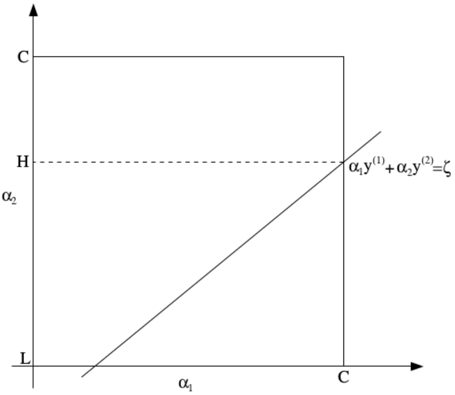
$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta.$$

Support vector machines

Optimizing the dual

The α_i's are already constrained to lie within $0 \leq \alpha_i \leq C$.

Imagine this constraint as a square. The new constraint means α_1 and α_2 must lie on a line:



Ng, CS 229 lecture notes set 3

We can now rewrie α_1 in terms of α_2:

$$\alpha_1 = (\zeta - \alpha_2 y^{(2)}) y^{(1)}.$$

Substituting this into our objective function, and keeping $\alpha_3, ... \alpha_m$ fixed, we obtain a quadratic function in α_2 that can be solved by setting the partial derivative to 0.

To ensure that the new α_2 satisfies the box constraint, we just "clip" it so that it and α_1 will lie in the range $0 \leq \alpha_2 \leq C$.

You now know enough to read the SMO paper by John Platt for more detail and a nice pseudocode description of the full algorithm.

Support vector machines

Production implementations

LIBSVM is probably the best open-source implementation of SVM model training and prediction:

- It implements an algorithm similar to SMO.
- It has wrappers for every major programming language and a command-line interface on Linux (`apt-get install libsvm-tools`).
- It is included in scikit-py and OpenCV.
- `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`

Another popular open source library is SVM^{light} from Cornell.

- `https://www.cs.cornell.edu/people/tj/svm_light/`

Hyperparameter selection (C, other parameters depending on the kernel) is extremely important with SVMs! See LIBSVM's `easy.py` script to automate the process.