Support Vector Machines, part 2

Logistic Regression, SVM, Learning Curves

Thinking in Statistics

...

Discuss.

How to Have Confidence

Talk about leave-one-out cross validation.

- `train_test_split`
- learning curves
- cross validation *(did you play with matplotlib: scatter, plot?)*
- LOOCV

Careful in all this about what you can compare!

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
```

Review: Linear Models

Logistic Regression. Start with Bayes Theorem, ask what is the probability that a point belongs to a distribution.

Switch to odds, take logs, assume linearity.

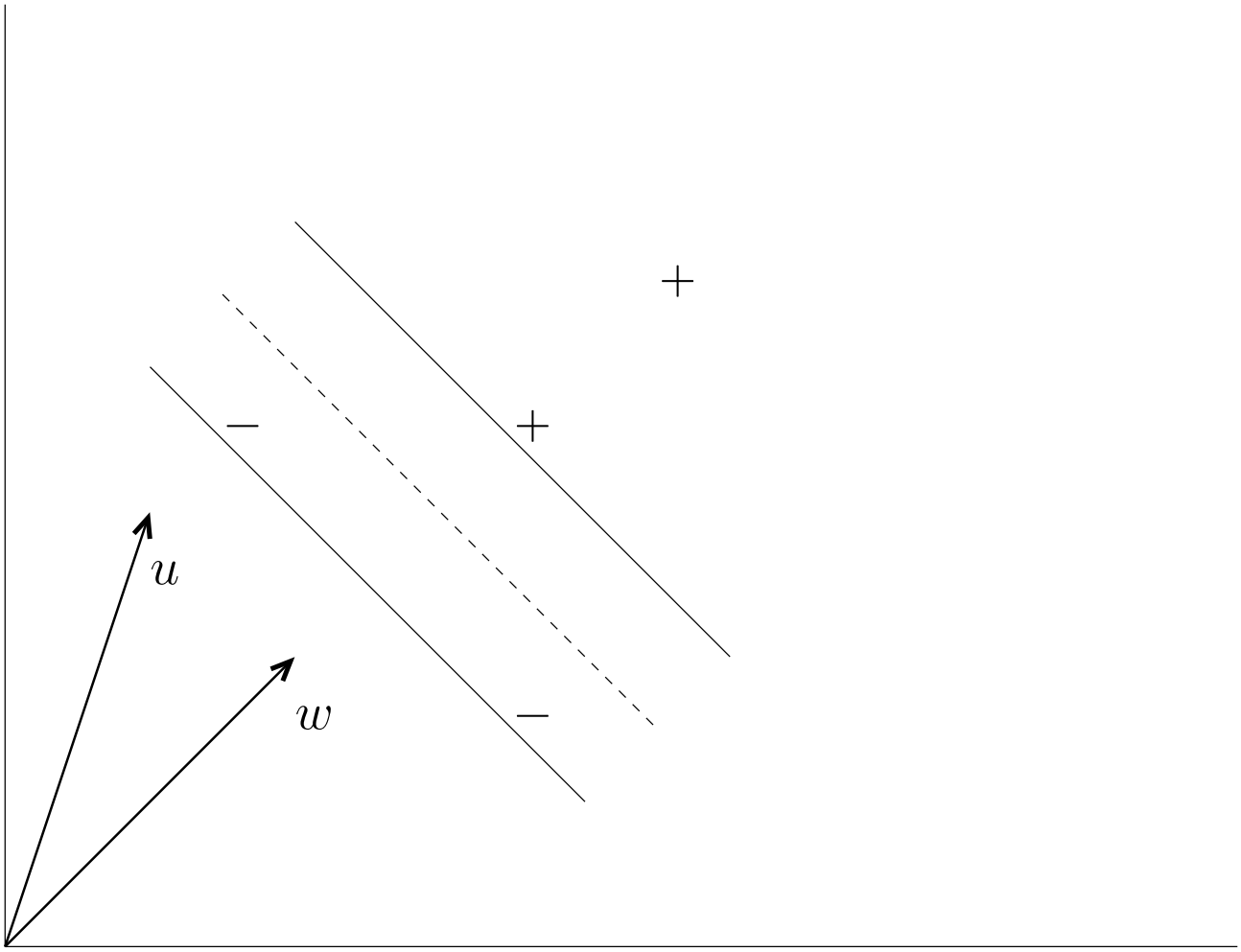
Take exponents and switch back to probability.

Theorem 1. *Bayes*

$$\mathbf{Pr}(H \mid D) = \frac{\mathbf{Pr}(D \mid H)\,\mathbf{Pr}(H)}{\mathbf{Pr}(D)}$$

SVM (linear)

We postulated the separator we wanted, called w a vector perpendicular to it, postulated x_+ and x_- points on the margin, and asked how to maximise the margin (the perpendicular distance between x_+ and x_-).



We set our decision rule thus (u on the positive side of the separator):

$$\boxed{w \cdot u + b \geqslant 0 \Rightarrow +} \tag{1}$$

Said differently, the equation of the hyperplane separator is $w \cdot u + b = 0$.

But we don't know b or w.

Note that the ± 1 indicates the margin. Zero is the decision boundary (the middle line).

We added some constraints so let us calculate them. We discovered we want to maximise (2).

$$\text{width} = \frac{2}{\parallel w \parallel} \tag{2}$$

So we can maximise $\frac{1}{\|w\|} \Rightarrow$ minimise $\parallel w \parallel \Rightarrow \boxed{\text{minimize } \frac{1}{2} \parallel w \parallel^2}$.

Our optimisation toolkit (gradient descent, mostly) doesn't work for constrained optimisation problems.

We use the Lagrangian and learn that w is a linear combination of the samples and only depends on the samples.

$$w = \sum \lambda_i y_i x_i \tag{3}$$

The thing to optimise is (5), this is where we learn, what we optimise.

$$\sum \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i \cdot x_j \tag{4}$$

So the optimisation depends on on the dot products of pairs of samples!

Let's go back to our decision rule, eq. (1), and write it as a recognition rule:

$$\sum \lambda_i y_i x_i \cdot u + b \geqslant 0 \Rightarrow + \tag{5}$$

Thi is the the decision boundary: it only depends on the sample vector and unknown!

It can be shown that this is a convex space, so we can't get stuck at local maxima.

We've seen that we can learn and decide using only dot products among sample data points and between sample data points and unknown (to be classified) points. This is the property that lets us use kernels.

Next: soft margins and kernels.

Soft Margin

First let's review SVM. In blue I note the more important changes needed for soft margins.

The further a point is from the separator, the more confident we want to be.

The idea is that we still want to maximise the margin, but we want to permit some errors. And we want to penalise errors based on how bad they are. And, ideally, not have too many of them.

If the margin is too narrow, it's more likely to overfit. Misclassifying some points can reduce overfitting. But if we misclassify too many points, the utility of the classifier decreases.

Let's add slack variables $\boxed{\zeta_i \geqslant 0}$. Let's set ζ_i to zero for correctly classified points and equal to the distance of misclassification otherwise. This is called a *hinge function ("fonction de perte de charnière")*. This corresponds to moving the misclassified points to the right side.

Note that this means a point on the separator incurs a penalty of 1 regardless of which way it's misclassified.

C is a regularisation term. We call it the *slack penalty* or *hardness (dureté)*.

(slack variable = variable d'ajustement / variable muette / variable d'écarte)

We had a decision rule

$$\boxed{w \cdot u + b \geqslant 0 \Rightarrow +}$$

We introduced points x_+ and x_- such that

$$w \cdot x_+ + b \geqslant 1$$
$$w \cdot x_- + b \leqslant -1$$

For convenience, we introduced

$$y_i = \begin{cases} 1 & \text{positive samples} \\ -1 & \text{negative samples} \end{cases}$$

Multiplying by y_i, we learned that

$$y_i(w \cdot x_i + b) - 1 + \zeta \geqslant 0$$

So for x_i on the margin, we had

$$\boxed{y_i(w \cdot x_i + b) - 1 = 0}$$

which is the same as

$$\boxed{y_i(w \cdot x_i + b) = 1}$$

and off the margin we have

$$\boxed{y_i(w \cdot x_i + b) \geqslant 1 - \zeta}$$

We want to maximise the margin, and we learned that this meant we should look at minimising

$$\frac{1}{2}w \cdot w + C\sum_i \zeta_i = \frac{1}{2} \parallel w \parallel^2 + C\sum_i \zeta_i$$

Writing $t_i = w \cdot x_i$ for the misclassification distance of x_i, we could also write this

$$\frac{1}{2} \parallel w \parallel^2 + C\sum_i \max(0, 1 - t_i \cdot y_i)$$

The left part is trying to maximise the margin, the right part is the *empirical loss* (how well we fit the training data).

Note that as $C \to \infty$ we get the old hard margin classifier. If $C = 0$, then ζ_i can be anything, so we ignore the data.

Kernels

But what if our data is not linearly separable? (Example: XOR.)

The optimisation only depends on dot products, so what if we had a function ϕ that mapped our problem into a higher dimensional space where it is linearly separable?

The learning rule (4) only depends on dot products. So we can map using ϕ and optimise in the new space.

In addition, the recognition rule (5) also only depends on dot product. So we can recognise in the new space, too.

In addition, we just need to define a function

$$k(x_i, x_j) = \phi(x_i) \cdot \phi(y_i)$$

and we don't actually need to know ϕ. (We need this to satisfy some technical conditions. Cf. Mercer's condition.)

We have something like

$$\phi : \mathbb{R}^d \to \mathbb{R}^D$$

and we could have D very large, even infinite. So K is a good efficiency gain!

You've already seen kernels (ridge regression, aka Tikhonov regularisation).

Popular Kernels

Linear kernel: $(u \cdot v + 1)$

Polynomial kernel: $(u \cdot v + 1)^n$

Radial basis function (RBK, RBF):

$$e^{(-\|x_i - x_j\|)/\sigma}$$

(Note: If σ is too small, we overfit.)

Explain how this works in practice, especially that you probably just always want RBF unless you get much, much more competent. Note that we can easily overfit with Gaussian kernels, because linear combinations of them are also kernels and can approximate any continuous function.

History

Vapnick did his PhD on this at Moscow University in the early 1960's. He worked at an oncology institute. He didn't have access to computers, so this was mostly of theoretical interest. No one in the West knew about his work.

Someone at Bell Labs discovered him, invited him to visit. He decided to emigrate to the U.S. in 1991.

In 1992 he submitted three papers to NIPS. All are rejected.

In 1993–1994, Bell Labs becomes interested in OCR and ANNs. Vapnick thinks ANNs suck, bets a colleague a good dinner that SVM will do a better job than an ANN.

Colleague tries linear kernel with n=2, and it works. This was the first use of kernels. (It was in Vapnick's PhD thesis, but he didn't think it was very important.)

So it took 30 years from his PhD (understanding the problem) to appreciating the importance of the technique.

To Do

Play with separator hardness. Plot it. Explore. Come back with plots and pictures and stories.

Play with RBF. It has parameters. What are they? What happens when you vary them? Come back with plots and pictures and stories.

How does using RBF affect computation time? With size of problem?