# Support Vector Machines

# Thinking in Statistics

A beer company did live demonstrations during football halftime: invite a few hundred people who prefer the competitor's beer to a blind taste test. Show that half of them actually prefer our beer.

This was done repeatedly on live television.

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: Logistic Regression

**Definition 1** (Kolmogorov)**.**

$$\mathbf{Pr}(A \mid B) = \frac{\mathbf{Pr}(A \cap B)}{\mathbf{Pr}(B)}$$

**Example 1** (validating points)**.** *Given a set of points and a new point, what is the probability that the new point is part of that distribution?*

*Solution.* This is actually logistic regression.

**Theorem 1.** *Bayes*

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

*Proof.* From definition and symmetry. □

Changes to odds formulation,

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

To make this look linear, we took logs:

$$\ln(C(H \mid D)) = \ln\left(\frac{\mathbf{Pr}(D \mid H)}{\mathbf{Pr}(D \mid \overline{H})}\right) + \ln(C(H))$$

and then we noted that the log prior is basically a constant and we can assume the first term, the log likelihood, is roughly a linear function of the data.

$$\ln(C(H \mid D)) = \beta D + \beta_0$$

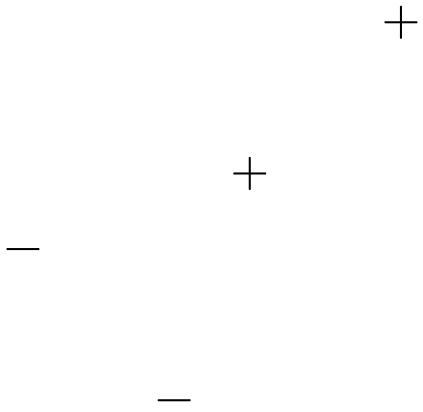
And then, if we want to do a bit more arithmetic, we can work backwards and find our familiar equation for logistic regression.

*solution/*

# Decision boundaries

Consider these points. What would be the decision boundaries for

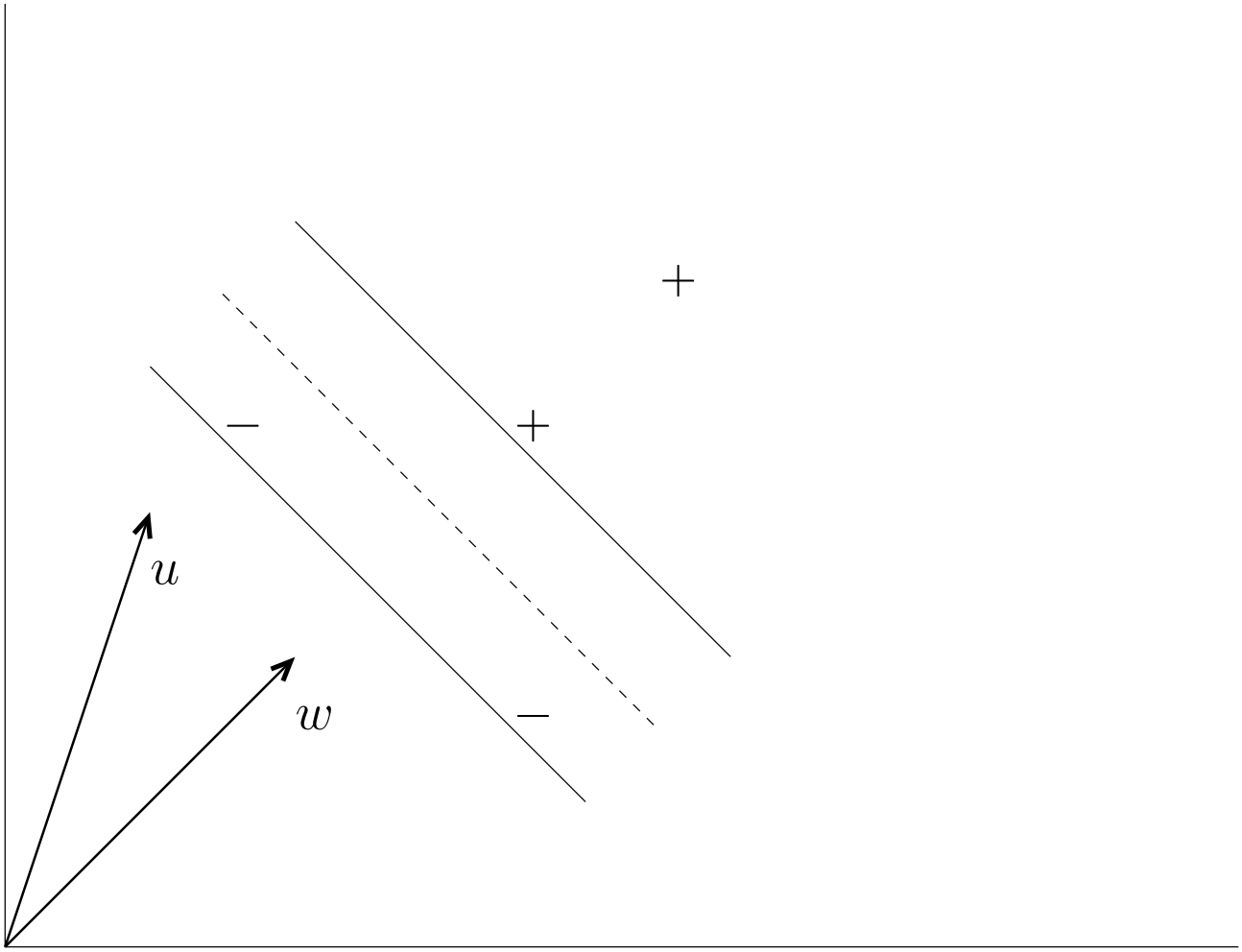
- logistic regression
- CART
- ANN



Consider a sort of "ideal" decision boundary. Stay linear, but which line?

Concept of widest street (or: bicycle lane).

How do we make a decision boundary that will use that line?




Consider a vector $w$ that is $\perp$ to the separator that we want. (We don't know its length.)

Let $u$ be a vector pointing to some unknown point that we want to classify.

Is $u$ on the $+$ or the $-$ side of the decision boundary (zone, margin)?

Let's project $u$ onto $w$. Then we want

$$w \cdot u \geqslant c$$

which is the same as saying this, which we'll call our decision rule:

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

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

But we don't know $b$ or $w$.

We want to add contraints so that we can calculate them.

Pick positive sample point $x_+$, negative sample point $x_-$.

Want

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

Note that the $\pm 1$ indicates the margin. Zero is the decision boundary (the middle line).

For convenience, introduce

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

Multiplying by $y_i$, we get

$$y_i(w \cdot x_+ + b) \geqslant 1$$
$$y_i(w \cdot x_- + b) \geqslant 1$$

These are the same!

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

So for $x_i$ on the margin, we have

$$\boxed{y_i(w \cdot x_i + b) - 1 = 0} \tag{2}$$

We want to maximise the margin. But what is its width?

If we have a unit vector, we could set the width. But we can subtract $x_+ - x_-$:

$$\text{width} = (x_+ - x_-) \cdot \frac{w}{\parallel w \parallel} \tag{3}$$

But we have

$$1(w \cdot x_+ + b) - 1 = 0 \iff w \cdot x_+ = 1 - b$$

and

$$-1(w \cdot x_- + b) - 1 = 0 \iff -w \cdot x_- = 1 + b$$

So combining with eq. (3), we get

$$\text{width} = \frac{2}{\parallel w \parallel} \tag{4}$$

We want to maximise (4).

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

Let's use the Lagrangian!

Talk about this, but skip the detail unless asked.

$$L = \frac{1}{2} \| w \|^2 - \sum \lambda_i \left[ y_i (w \cdot x_i + b) - 1 \right] \tag{5}$$

(It will turn out that $\lambda$ is non-zero only on the margin.)

$$\frac{\partial L}{\partial w} = w - \sum \lambda y_i x_i = 0$$

and so (show $w$ and note dependence on samples)

$$w = \sum \lambda_i y_i x_i \tag{6}$$

So $w$ is a linear combination of the samples!

In addition,

$$\frac{\partial L}{\partial b} = - \sum \lambda_i y_i = 0$$

and so

$$\sum \lambda_i y_i = 0 \tag{7}$$

So what happens to $L$ (eq. 5)?

$$L = \frac{1}{2} \left( \sum \lambda_i y_i x_i \right) \left( \sum \lambda_j y_j x_j \right) - \sum \lambda_i y_i x_i \left( \sum \lambda_j y_j x_j \right) - \boxed{\sum \lambda_i y_i} b + \sum \lambda_i \tag{8}$$

The boxed portion is zero, so this leaves us with pick up again here.

$$\sum \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i \cdot x_j \tag{9}$$

This is the optimisation: the part where we learn.

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

Let's go back to our decision rule, eq. (1).

$$\sum \lambda_i y_i x_i \cdot u + b \geqslant 0 \Rightarrow +$$

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 week we'll come back to this and talk about soft margins and about kernels.

# Scikit Learn Notes

```
model.fit
model.predict
```

There are toy data sets to play with.

```
import sklearn.datasets as skds
skds.fetch_covtype
```

# Soft Margin

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

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.

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

$C$ is a regularisation term. We call it the *slack penalty* or *hardness*.

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

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

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 $\zeta_i$ can be anything, so we ignore the data.

Then we used the Lagrangian to find an objective function

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

and a recognition function

$$\sum \lambda_i y_i x_i \cdot u + b \geqslant 0 \Rightarrow +$$

# 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 $\phi$ that mapped our problem into a higher dimensional space where it is linearly separable?

The learning rule (8) only depends on dot products. So we can map using $\phi$ and optimise in the new space.

In addition, the recognition rule (9) 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 $\phi$. (We need this to satisfy some technical conditions. Cf. Mercer's condition.)

We have something like

$$\phi : \mathbb{R}^d -> \mathbb{R}^D$$

fand 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 $\sigma$ is too small, we overfit.)

# 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.