Everybody, welcome.

The topic today is Support vector machines.

As always, I will go through the code and I encourage you to do the same.

We'll be using two special pet libraries today,

there is the e1071 for SVM or Support

Vector Machine function and pROC for making ROC curves.

We'll be using the iris dataset once again.

We've seen it before.

The idea is that I'm going to tell you a bit about how to actually fit this SVM.

We'll start by focusing on two classes.

Specifically, we will use virginica and versicolor.

Those are the two that are more entangled and we'll use sepal length and sepal width,

which are the two less effective variables,

just to make things challenging.

Now, this is making a plot.

The components are pretty straightforward.

We're going to take the Iris dataset,

filter out setosa,

as long as species is not setosa,

select only species sepal length and sepal width,

convert species to a factor and then assign it to iris 2.

We're going to plot it these particular variables

and color-coded by the species here.

We'll add two to make the colours a little nicer.

By default, it will be black and red.

Then finally, we'll use a legend here with the species and

use the appropriate colours

to those.

Now, the reason for this bit with species equals factor is that actually it seems picky.

If it sees vectors,

one of the predictor variables,

it will automatically convert it to a factor.

Not so with SVM,

you actually have to convert it manually.

How do we fit a linear SVM? Pretty straightforward.

SVM, species,

given the rest of the variables in the dataset,

data and we're told use a linear kernel.

Now we can actually plot this.

Actually makes sense.

Here support vectors, that is,

vectors that support the hyperplane that have a non-zero weight

in the dual representation are indicated with

X and non support vectors are the ones that are solved in

the interior of their group,

are indicated with O.

The colours are used to indicate what is the true class of a data point.

Notice the separation boundary is linear.

It's a bit weighty because of the way we plot things.

Now, let's try, say,

an RBF, Radial Basis Function,

SVM with a default value for the decay parameter,

which is 1 over the number of variables.

The specification is pretty straightforward.

We just say kernel equals radial,

which is, I believe,

the default as well.

Once we plot it,

we see that we can now get a slightly different structure to our classification boundary.

We can make the RBF decay faster by increasing gamma,

which we do by passing this here.

This is what it looks like now.

Notice that the curves are a bit sharper.

The shape is a bit more defined.

If we make the decay really fast,

then what actually begins to happen here is

that each point can create a potential island around it.

We see even a more extreme example when we set gamma equals 100.

We could also make this to really highlight which will have the same effect.

Now, that suggests that we need to tune our SVM,

because the lower we make our gamma and the lower we make our cost penalty,

the more linear-like it becomes,

the fewer support vectors,

which means that it's faster,

which doesn't mean faster prediction.

It means that the predictions are going to be more contiguous,

which could be an advantage in

itself and it'll lead to a higher misclassification rate in the sample.

On the other hand, the higher we make gamma and C,

the predictions used are allowed to have more complex shapes,

which could be a good thing if that's what the data suggests.

But in extreme cases it could lead to islands.

You do end up with many support vectors,

so it's computationally more costly.

It can be a form of overfitting,

so you end up with a higher misclassification rate out-of-sample, at least usually.

Here are the in-sample confusion matrices for the default and radial 100.

Notice that the off-diagonal entries here, that is,

misclassification is higher for the default value 1 over d rather than 100.

Now let's use cross-validation.

We'll use 10-fold to split the data into 10 chunks and each chunk will

take a turn being predicted and given a model fitted the others.

In fact, it turns out that it'll give us cross validation accuracies here,

which takes this into account.

What we see is that actually the higher gamma results in a lower total accuracy.

Here we have about 69 percent on average,

and here we get about 43 percent on average.

It turns out we can actually automate that.

Here we specify the range of gamma values or

the gamma values you wanted to try and the C values you wanted to try.

We use a function called tune.svm.

Here are the results.

This is an error or misclassification

performance rather than an accuracy. This is the result.

Here we can look at the best model.

It doesn't unfortunately tell us right off the bat what's our best gamma,

but we can actually look here because we want the smallest error.

Actually the best performance seemed to be with Gamma equals one and cost equals one.

Multiclass SVMs work again without too much trouble.

Here we have the same SVM with

the default settings and cross validated accuracy estimation,

and this is what the classification plot looks like.

Note that the axis are flipped from the discriminant analysis.

Now, of course we want to predict based on this,

so suppose we have measured a new flower which has

sepal width 3.4 and sepal length of 6,

what does its species likely to be?

What we can do is we can use the standard arguments,

predict new data equals,

and then data frame contain just the new values for the new data points,

and then we'll also going to ask decision values equals TRUE,

so these are the support vector predictions,

SVM based predictions for that,

which gives us not probabilities but at least some

notion of a soft clustering or measure of certainty.

As we discussed in the slides,

basically when you have a multiclass SVM,

you basically do these pairwise comparisons.

First you say, is it setosa versus versicolor and it says,

well, minus it's negative,

so it's more like versicolor than setosa.

Is it setosa versus virginica?

It's negative, so it's probably more like virginica than like setosa.

Finally, is it versicolor or virginica?

Well, it's negative,

so it's more like virginica than like versicolor.

Therefore we decide on virginica.

Next item we'll talk about how to actually compare these,

and one tool we'll talk about is ROC curves.

You may have seen them in other classes.

Now, the predictors we have for linear classifiers,

and that includes SVMs in their dual form,

so they're continuous we can actually use some threshold

that's not just whatever we fit in the model.

This can be useful because sometimes the cost of misclassifying in

one direction is different from the cost of misclassifying in the other direction.

This is where again decision values come in

useful and we'll once again be using the data iris2.

That's the one with only the two species that are mushed together,

and what we're going to do is we're going to ask for it to predict

from the SVM radio fit within your data being iris2.

Again just those two are the original data,

and then we'll ask it for decision values and just a bit clunky,

but then we'll extract the decision values attribute that's here.

This one. We'll assign it to the y.hats for

the radial based on

radial basis functions and plug them in histogram and then see what they look like.

We'll use the pROC package,

to get the ROC curve and I'll explain how to interpret it in a moment.

Now, the idea here is that we're going to say this is now ROC curve,

so we're going to tell them the truth,

which is whether or not the species is

versicolor and then we'll have the prediction which is the y.hat.radial,

so basically this quantity here for each of the data points of the dataset.

Similarly, we'll make a prediction based on linear SVM and again,

predict whether it's versicolor or not.

Finally, we're going to plot the ROC curve for one of them,

with one colour, with for linear with another colour,

and then make a little legend.

Add equals true means don't create a new plot,

it means add on top of the existing plot.

Here's a result. What does this mean?

Well, we're trying to detect whether a species is versicolor as opposed to not,

as opposed to virginica in this case.

Now, when we classify,

we can quantify the classification using some measures like sensitivity and specificity.

Sensitivity is how likely are we to detect,

in this case, versicolor?

When it's given that it's actually versicolor, and specificity is

if it's not versicolor,

so if it is virginica, how likely are we to classify it as virginica.

Now, where we set the threshold for the decision value or really discriminant as well,

same logic applies will affect

these two variables because the higher we set

the bar for classifying something as versicolor,

the more likely we are to not classify it as versicolor when it is,

so the lower with our sensitivity.

The lower we set the bar,

the lower specificity because the more false positive versicolors there will be.

We can therefore make a plot that looks like this,

if the classifier were completely useless,

it would lie along this line.

If the classifier were perfect,

it will go straight up,

and then straight to the right.

Usually there's something in between and generally the more bowed out in

particular ROC curve is the better.

In these, each point here represents

a different classification threshold or different bar for it to meet.

Here we don't really have a very strong difference.

It looks like maybe linear SVM is better

if you want high sensitivity

whereas RBF is better when we want to higher specificity,

but know that could just be noise.

That's support vector machine demonstration.

Again, I would encourage you to play

around with it yourselves and run through this code.