Classification

Recall that classification means we have some blue points and some red points, and we want to find a boundary that separates them.

But now we want the boundary to be as far away from the points as possible. To do this we add two boundaries (shown in the video as equidistant parallel lines to the main line), and try to maximize the distance between the two.

There are two ways to measure the model:

- By how many points it mis-classifies
- By how wide the margin is

We can 'punish' the misclassified points, which means make them part of our classification error. We also do not want any points in the margin, so we would 'punish' those points as well and include them in the classification error.

Lastly, we want the margin to be as wide as possible and will see how to calculate a margin error a little later. Our new error for this algorithm is going to be a classification error + a margin error. Minimizing this error is what's going to give us the algorithm for Support Vector Machines.

Quiz Question

Which of the following is true about SVM?

a. We want the boundary to be as close as possible to the points.
b. We want the boundary to be as far away from the points as possible.
c. The boundary should intersect at least 1 of the points

Quiz Question

Which of the following is not true about SVM?

a. The error for the SVM algorithm is classification error + margin error
b. The error for the SVM algorithm is classification error - margin error
c. minimizing the (classification+ margin) error is the goal

Perceptron algorithm

Recall that the perceptron algorithm works in the following way.
You have some data points like the blue and red, and we want to find a perfect line that splits them.

A perfect line has the equation $Wx+b=0$, where W and x are vectors and b is a scalar. So it looks more like this, $w_1x_1+w_2x_2+b=0$.
As a quick example, let's say the equation is 3x1+4x2+5=0.

So we want to 'punish' this line with some error function, and the error function roughly measures how many points are misclassified. It actually measures something closer to the sum of the distances of these misclassified points to the boundary.

We can split this into two sets, the red points and the blue points, and our error function is going to punish those two red points on the left that are on top of the line and those two blue points on the right that are below the line, but it's going to punish them according to their distance to the main line. It's going to be the blue area that is over the line.

The more a red point is into the blue area, the more it gets punished. This means that the point that is close to the line gets punished a little bit, and the point that is far from the line gets punished more. And the other part of the error does the complete opposite. It punishes the blue points that are in the red area. The points that are close to the line don't get punished much, and the ones that are far from the line get punished a lot more. But let's actually put some math here. Let's ask how much the error is at every point.

So we have that our equation $Wx+b=0$, which is a linear equation. $Wx+b$ takes all the values, so in particular, it is one of the lines that is parallel to the main line and a little bit above and then it takes the value 2, 3, 4 successively. And the same thing on the negative side, it takes the values -1, -2, -3, -4, etc. Basically, this is going to be the error. It's going to be the value of $Wx+b$. Since we have negative numbers, it's going to be $|Wx+b|$

Quiz Question

The error calculated in the Perceptron Algorithm uses the Absolute value of (Wx+b). Why is the Absolute value needed?

a. There are misclassified points both below and above the line
b. Sometimes the red points or blue points are below the x-axis

SVM algorithm

Let's do the same thing we did for the perception algorithm, except with the SVM algorithm.
Recall that the equation of the line is $Wx+b=0$.
Now we don't just want a single line, we want the line with two *extra* lines that create the margin. And the equations for these lines are going to be

- $Wx+b=1$
- $Wx+b=-1$

We still want to 'punish' the points that are incorrectly classified. But now, we're going to think of the **region** as points that are incorrectly classified since we don't want anything in between the two lines.
Let's split the error in two. In order to punish the points that are within the margin

- The blue error will now start from the bottom line
- The red error is going to now start from the top line

We look at the values of $Wx+b$. As we go up, it's going to be 1, 2, 3, 4, etc . And as we go down, it's going to be -1, -2, -3, etc. In order to build the error, we take the absolute value of those errors and translate it into one.

- The blue error starts at zero on the bottom boundary line and increases by one with each step
- Similarly, the red error is going to be zero on the top boundary line and then increase linearly in the opposite direction.

Quiz Question

As we saw with the Perceptron Algorithm, the error calculated in the SVM Algorithm also uses the Absolute value of (Wx+b). Why is the Absolute value needed?

a. There are misclassified points both below and above the margin line
b. Sometimes the red points or blue points are below the x-axis

Quiz Question

Which of the following statements is true about SVM classification errors

a. The classification error of each miscalculated point is based on the distance that it is from the main boundary line.
b. Points that are within the margin contribute more to the overall classification error than other misclassified points.
c. The classification error of each miscalculated point is based on the distance that it is from the extra lines that form the margin around the boundary line.

Building a margin error

Recall that the margin is the distance *between the two lines*, and we want to turn this margin into an error that we can minimize using gradient descent. We want a function that gives us a small error for the large margin case, and a large error for the small margin case. This is because we want to punish small margins, as our goal is to obtain a model that has as large a margin as possible.

We have our line with the two other boundary lines, and the margin is the distance between the two outside lines.
Our equation is a line:

- $Wx+b=0$

and the two dotted lines have equations:

- $Wx+b=1$
- $Wx+b=-1$

The margin is 2 divided by the norm of the vector W $2/|W|$

Remember that the norm of W is the square root of the sum of the squares of the components of the vector, which are W1 and W2.
For this error, let's find something that gives us

- A large value if the margin is small
- A small value if the margin is large

The norm of W ($|W|$) appears in the denominator. If we take the norm of W, that grows inversely proportional to the margin. To avoid dealing with square roots, let's take the norm of W squared, which is actually the sum of the squares of the components of the vector W. In this case, it's
$w_1{}^\wedge 2+w_2{}^\wedge 2$.And as we've seen, since W appears here in the denominator, then a large margin gives us a small error and a small margin gives us a large error. *That is exactly what we wanted*.

To clarify things, here's an example.
Let's say $W=(3,4)$ and our bias is 1. So our equation of the form $w_1x_1+w_2x_2+b=0$, is going to be $3x_1+4x_2+1=0$, and that's our main line. And the two companion lines, $3x_1+4x_2+1=1$, and $3x_1+4x_2+1=-1$. The error is $|W|\ {}^\wedge 2$, which is $3\ {}^\wedge 2+4\ {}^\wedge 2$ and gives us an error of 25. The margin is 2 / $|W|$, and the $|W|$ is the square root of 25 which is 5. So, the margin is 2/5 and the error is 25. Let's remember these two numbers: error:25 and margin:2/5.

Now, let's look at a very similar example

Instead of our previous weights, let's assume $W=(6,8)$ and our bias is 2. Our line is going to have the equation: $6x_1+8x_2+2=0$. If you notice, that equation is the same as before except multiplied by 2. So, it gives us the same boundary line because when $3x_1+4x_2+1=0$, then $6x_1+8x_2+2=0$. But now our dotted lines are closer to each other. Before we had $3x_1+4x_2+1=1$. And now, we have the twice of that equals one, which means $3x_1+4x_2+1$ is actually 1/2, which means the line is much closer. It's actually half the distance as before, and the same thing happens with the line below.

Our error is a square of the norm of this vector, which is $6 \wedge 2 + 8 \wedge 2$ which is 100. And our distance is going to be $2 / |\mathrm{W}|$, which is 2/10. That is the same as 1/5, so this is smaller than the previous margin of 2/5. Two model examples give us the same boundary line, but one of them gives us a larger margin than the other one.

Summary

We have our large margin, our margin of 2/5 that gives us a small error of 25, and our small margin of 2/10, which is 1/5 gives us a larger error of 100.
That is the margin error. It's just $|W| \wedge 2$. This is the exact same error that is given by the regularization term in L2 regularization.

Quiz Question

Which of the following are true about SVM (There's more than one correct answer)

a. Large margin = small error
b. Large margin = large error
c. Small margin = large error
d. Small margin = small error

Classification error + margin error

We've learned about the classification error and the margin error. Bringing them together is what SVM does. The official SVM error is just going to be the classification error plus the margin error. And how do we minimize this? We minimize it using gradient descent, just like we have been doing.

Quiz Question

How is the SVM error minimized?

a. By subtracting the margin error from the classification error
b. By using gradient descent
c. By subtracting the classification error from the margin error

The problem

How can you determine the best possible boundary line?
It depends a lot on the data and the problem that we're approaching, which means that we need some flexibility. That flexibility is going to be provided by the C parameter.
The C parameter is just a constant that attaches itself to the classification error by multiplying the classification error by the constant.

The **C** parameter - also referred to as the **C** hyper-parameter - determines how flexible we are willing to be with the points that fall on the wrong side of our dividing boundary. The value of **C** ranges between 0 and infinity. When **C** is large, you are forcing your boundary to have fewer errors than when it is a small value.

Note: when C is too large for a particular set of data, you might not get convergence at all because your data cannot be separated with the small number of errors allotted with such a large value of C.

- If we have a very *large* C, then the error is mostly the classification error, so we're focusing more on correctly classifying our points than on finding a good margin.
- If the C is very *small*, then the error is mostly a margin error, so we're focusing mostly on a large margin than on classifying the points correctly.

Quiz Question

What is true about the C parameter? (There may be more than one correct answer)

a. It is used to modify the classification error
b. It is a hyperparameter that provides some flexibility during training
c. A large value for C will usually result in a small margin
d. A small value for C will usually result in a large margin

Classification problem and the Kernel Trick

Sometimes a line will not be good enough to split our data. *What we need is a more complex model.*
Our points are in a line, but here we see them on a plane. To exploit that, we switch from a one-dimensional problem, the line, to a two-dimensional problem, the plane, by adding a y-axis. And then use a parabola. Let's draw the function
$y = x \wedge 2$. Then we lift every point to its corresponding place in the parabola. All of a sudden, our points are nicely separable because now, using the original SVM algorithm, we can find a good cut. It's the line $y=4$.

How do we bring this back to the line and find the boundary there? Our original equation is $y=x \wedge 2$. Our line is $y=4$.

How did these two combine? By equating them, we get a new equation:
$x \wedge 2=4$, which factors into two linear polynomials with solutions

- $x=2$
- $x=-2$

So, those will make our boundary. We bring this back down to the line and we have $x=2$ and $x=-2$ as the boundaries for this model. Notice that they split the data really well.
This trick is known as the Kernel Trick and it's widely used in Support Vector Machines, as well as many other algorithms in machine learning.

Quiz Question

When separating points, when does it make sense to use a polynomial solution

a. When the points can be divided vertically
b. When the points can be divided horizontally
c. When the points cannot be divided by a straight line

Radial Basis Functions (RBF) kernel

Polynomial kernels are not the only tool that we have to split data. In this section, we will learn about the RBF kernel.
This process has us pull points off of the line and plot them on a "mountain range," split the points on the range, and then translate them back to their original position with multiple cut points.

The technique

Build a "mountain" on top of every point. The technical term for these mountains is radial basis functions. How could we combine these functions in a way that helps us separate the blue and the red points?

We can multiply the mountain over the red point by -1, which flips it, and then add the functions together. At every point, *just add the three heights*. Next, move the points to the corresponding sum. After we do this we can easily draw a line that splits the points in two. This line will intersect the mountain range at two points. When we project down to the line, we get two cuts that match where the line intersected our mountain range.

```
Notice that these cuts separate the blue points from the red points as
we wanted.
```
We are not limited by addition, we can actually reach each mountain by any constant that we want.

How do we find this weight?

- Place one mountain/function on top of each point. Under each point, record the value of each function, or how tall the mountain is at that point.
- Continue doing this for each point and each function.
- Each point will have one value of 1 in their vector of heights, since the height of the mountain corresponding to that point is one by construction. In general, the other values will be small.

How do we find the right linear combination of functions which will be able to separate the blue and the red points?

Take the three height vectors and plot them in a three-dimensional space. Since we have as many dimensions as points, we'll be able to separate our points well. Using the application of our known SVM algorithm, we should be able to separate these red and blue points with a plane with the equation, $2x - 4y + 1z = -1$. If we take the constants of the equation off the plane, they become the constants of our model. The first mountain has a weight of 2, the second -4, and the third has a weight of 1. The line that separates these points is the line at height -1.

Quiz Question

When drawing a graph using radial basic functions, the 'mountain peaks' should be located where?

a. Directly over the points
b. Between the points

How to decide what to use

We can use some very wide ones or some very narrow ones. This is a hyperparameter that we tune during training and it is called the **gamma parameter**.

- A large gamma will move us to a narrow curve
- A small gamma would give us a wide curve

In higher dimensions, this is very similar.

- A large gamma will give us some pointy mountains
- A small gamma would give us wider mountains

The gamma matters a lot in the algorithm. Large values of gamma tend to overfit, and small ones tend to underfit.

What is Gamma?

Well, here's where we define these radial basis functions. We'll use the Gaussian or normal distribution for this.

In the general case, when mu is the very center of the curve, and sigma is related to its width, we have these rules of thumb:

- If sigma is large, then the curve is very wide
- if sigma is small, then the curve is very narrow

So in order to define **gamma**, we just use

$\gamma=1 / 2\sigma^2$

1

And keep in mind

- if gamma is large, then sigma is small, so the curve is narrow
- If gamma is small, then sigma is large and the curve is wide

In the higher dimensional case, this formula becomes a little more complicated. But as long as we think of gamma as some parameter that is associated with the width of the curve in an inverse way, then we are grasping the concept of the gamma parameter and the RBF kernel.

Quiz Question

Which of the following are true about 'Gamma' (There may be more than one correct answer)

a. if gamma is large, then sigma is large, so the curve is narrow
b. If gamma is small, then sigma is small and the curve is wide
c. [✓ If gamma is small, then sigma is large and the curve is wide]
d. [✓ if gamma is large, then sigma is small, so the curve is narrow]