Good morning everyone. Welcome back. I. I would like to start this lecture talking briefly about the assessment, because I don't know whether you're aware of it, but the deadline is in a little bit over two weeks. So the deadline was, I think, the 7th of December, which gives you two weeks and two days, though I would strongly recommend that you try to submit a little early. You will remember that last time we had an outage of turn It in exactly on the day of my deadline, which was perfect timing and sent all of us in a bit of a panicked frenzy. So you will also know that if these kinds of university wide outages happen, there will be no deductions of marks as long as you keep trying to submit. That being said, I'm sure if you were affected by that, you were also a bit of a yeah, in a state of panic if something doesn't happen. So I would recommend try to submit early so that you can sit back and relax and enjoy the show if something happens again. Are there any questions right now about the assessment which you would like to raise when you think about the exam or anything like that? Are you working on your assessment? Please tell me you are. Okay, so you have started. It's not like you haven't started at all. Okay. That's good. That's very good. So given that the assessment does allow you to kind of. Use any method that you deem suitable for the task. Obviously, as we go on in the courses. You might think, okay, I will wait until we cover support vector machines or decision trees or whatever in class before I actually try to apply them. So I don't expect you to be done yet, but I would strongly suggest that you try to kind of start writing it up and wrapping it up, because that will give you a bit of time at the end. Any questions on the exam? I remember last time you had a few questions about that. Have you had a chance to think about it some more? Yes. Was that. The slides, the recommended readings and the content of the slides is slightly different, as in terms of in the slides, we have a lot more about distributions and everything, but the recommended readings technically doesn't talk much about it. So there is a difference between wondering are we? Is there anything else that we should be knowing about as in the the distribution doesn't? Is there anywhere else that we. Yeah. So generally what we discuss in class. So what I tell you or what you tell each other or what we kind of have on the slides and then the reading is mostly supplementary. So it's kind of adding more depth and more understanding to that. So sometimes reading kind of gives you a bit of a different perspective or it recaps and if it doesn't, if it's not covered in the reading, but you have a comfortable understanding what what we discussed, you don't have to go out of your way looking for additional reading material, if I didn't suggest any. That being said, if it helps you, then you can do additional reading, obviously. So for example, when we talked about distributions, it's kind of it would be maybe a natural fit to go for the textbook of. Instead, if you feel like you would like to have a recap of that. Yeah. Anything else. Yes. Are there going to be any questions on the output? And. Or is it more just theory? Yeah. So it's theory based. The idea is basically that the coding part of the course is assessed with the group coursework, and then the theory part is assessed for the exam. Okay. Happy? More or less happy. I heard you're talking about more coursework. Do you have more deadlines coming up? When? Two the week before. And two of the same. Oh, gosh. So it's very intense. Is it? Yeah, I can imagine. I think the problem with these kind of programs is that we try to kind of teach a lot of material, and then obviously we have this end phase where we actually assessing all of what we taught. Yeah, I can imagine that. It's very intense. I had the same problem when I was doing my masters. We actually had block teaching. That's a whole different story. So we actually had a block of teaching and then the next block and the next block. So only one class over like two weeks, but we still had all the exams at the end. So you kind of then had to recap all the material at the very end for the exam. So that was a different challenge. I think this kind of structure we have right now works a little better for most people, because it kind of keeps everything a bit fresh. Okay. Should we get started? So last week in week nine we talked about decision trees. And you will remember that we were talking about decision trees for both regression and classification problems. And you will also remember that the idea of that was it's basically building a structure of decisions. So you start somewhere at the top and then you make these kind of stepwise binary splitting criteria where you decide on which side of the tree your sample should be, and you build a tree using your training data, and then you basically run the test data through the whole tree and decide what happens to that particular test data sample. It could be a classification or it could be a regression. And you also remember that we said decision trees themselves tend to be not super accurate. So they can tend to for example, overfit. They tend to not have that high of an accuracy on their own. But there are a couple of steps we can do to improve them. Most notably, we talked about bagging. That was this whole bootstrap aggregation or as I like to call it, throw a lot of bags into a tree and shake it up. We had boosting. That was the idea of using the residuals. So basically all the spaces that our trees do not look into and take that to train small kind of mini trees, which are then attached to the whole model. And we talk about random forests, where the idea was that we introduce additional randomisation in particular, not only all kind of all our trees. So we grow a lot of different trees. They are random, but we also restrict how these trees can grow. Specifically, we restrict them on the number of variables they can use for each of their splits. And this kind of added randomness and added pressure on each of these individual trees leads to random forests being quite popular and quite strong classifiers and and predictive models in general. So random forests are very popular. And today's lecture we will actually look into another technique, which is also very popular because it's also proving to be a surprisingly strong predictor for a lot of different problems. So we will talk about support vector machines today. And support vector machines are a really interesting part of machine learning. And you will see when we talk about how they actually work, that they are basically coming from the area of optimisation, more so than kind of a lot of other areas of machine learning. So a lot of machine learning will be discussed, like decision trees and all of that. They feel very kind of intuitive. So it's basically this idea of in machine learning we don't really care why it works, it just works. So we're happy with that right? Support vector machines really take this approach of optimisation again. So it feels a bit more like the idea of regression. Maybe what we were talking about earlier in the course. And in my own experience, SVMs are very popular with mathematicians dabbling into machine learning, for example. So we will talk about maximum margin classifiers and optimisation of those. Then we will talk a bit more about how these then developed into support vector machines via a couple of different steps. So this idea was developed in the computer science community. And it was it was developed quite a few years ago. So actually it is one of the older techniques we are looking back at the 1990s, which is kind of why we have to swarm optimisation focus in that maybe. But they have only grown in popularity, so they used to be fairly popular in the back and back in the days, but now they are kind of even more growing and becoming more and more popular. So in a lot of papers, if you look into the literature, you actually have a comparison, for example, of Random Forests support Vector machine and then maybe a neural network. So these are like the big three that you often see pop up in the literature. So as I stayed here, they are often considered one of the stronger classifiers. And I always say that with caution because it depends strongly on your problem. So don't fall into this trap of overly relying on just one type of model. I think a lot of kind of data scientists, analysts in general make this mistake of kind of having a favourite. So they have a favourite modelling approach. And I'm I'm kind of doing the same thing. I mean, I try to force clustering into every problem it could possibly fit because I think it's so amazing. But on the other hand. I think it kind of leads you to tend to look for something that isn't there. So frequently seen problems where people try to force and use a specific model because they think it's the best classifier can be a random forest, can be a neural network, whatever. And then they look for ways of proving they are right. So it kind of goes into this area of p value hacking where you're explaining, hey, this model might have the same accuracy as this other model, or even a little worse, but it's better on this very specific metric. And therefore it's the best model is just proving my point again. So don't fall into this trap. Be kind of open minded. Understand that the best model always depends on your specific problem that you're trying to answer. I also mentioned that SVMs in general are basically a generalisation of this concept of a maximal margin classifier. So we will see that this is one of our linear models or our linear classification models again. So if you remember when we were talking about linear regression, we basically said okay, one of the advantages of linear regression is it's simple. And one of the disadvantages is it's linear. And you will see a very simple, very similar concept with our maximum margin classifier here because we are looking again for classes which are separable by some kind of linear boundary. The good thing is then that we generalised it away from this idea. And SVM stand are actually classifiers that can accommodate non linear class boundaries really, really well through a little trick which we call the kernel trick. So we will talk a little bit about that later. But it's basically a clever idea of projecting data into space in such a way that you can use a linear classifier for it to separate it. So it's really quite clever because the classifier itself is still at the boundary we draw is still linear, but because we first project the data into a different space, we are actually allowing a linear boundary to work on non linear data. So you will see that later. And I will also have a picture which hopefully explains that. Really clever. Sorry. Sorry. Okay, so before we get started with these ideas of classifiers, we have to talk briefly about hyperplanes. Hyperplanes are really important concept in that because we just said we kind of looking for these linear linear boundaries, these linear decisions that we're trying to draw and we draw them in the form of hyperplanes. Now a hyperplane is really just any flat surface and flat and the kind of wider sense which we can draw with space. So in two dimensions a hyperplane would be a line. For example. You can think about that if you have two dimensions and you have some kind of point distribution, you draw a hyperplane through that to separate it into two parts. The idea is always that a hyperplane is able to separate the separated space completely into two parts. So in two dimensions there would be a line. We can draw a line through that space. In three dimensions it would be a flat surface. So we can draw like a flat surface through the whole space. I can't tell you what it would like. We look like four or 5 or 6 dimensions. I'm not that intelligent. I'm sure some people can imagine it in four dimensions. I try to do that once. They are really, really clever pictures online. If you ever wondered what four dimensions might look like, they are like abstract visualisations of that. But as soon as we get more than five more than four dimensions, it's impossible to imagine, but mathematically possible. And that's all we care about here. So I made a little correction here. If you have the old version of the slides, you will see that there was one more variable added, but it makes much more sense like that. So in two dimensions as I said a hyperplane is a line. So you can imagine a bit like a linear regression line. Basically you have some kind of intercept and you have some kind of slope for that line. And in three dimensions it would be just more and more dimension, because you're then describing a flat surface. So always keep this concept in mind when we talk about SVMs. So we can generalise that to a p dimensional setting. So that's a lot of different dimensions. Remember dimensions in our case always refer to basically the data that you have. So how many dimensions do you measure a specific observation in. That can be for example, if you're conducting some kind of survey or you're collecting data on an individual, how many dimensions do you measure about that person? So there will be their age and their income and kind of all of these different dimensions, all of these questions. So P is basically determined by the size or by the width the number of columns in your data set. So sometimes you will have points which don't satisfy this definition of a hyperplane. So they might lie on one or the other side of that of that hyperplane. So imagine again our kind of space in a simple term. You can imagine it just as the line in the two dimensional space. And then you will have points on either side of that hyperplane. Now remember that yes, it's a classifier. Obviously, we're trying to draw the line in such a way that the points are on the correct side of the hyperplane. So we think there are two classes for example, and we're drawing a hyperplane through that so that we are dividing the points into these two classes in a in the correct way, basically. So what does that look like? Something like this. So imagine we have here our two dimensional space. We have points and we know which class they belong to. So we have blue points and we have pink points. And we're trying to find the line, the hyperplane which divides this space in such a way that there are two halves. And they describe kind of these two classes that we think blue and pink. You can see there's a lot of different ways that we can draw this line, but there will be one line which is the perfect way of kind of dividing the space of classifying these points. You can then see that for each of these points, we can basically look whether they lie on the right side of that plane or they don't. So whether they were classified correctly or they weren't. The interesting thing about this kind of concept is if you notice here, that some points will be closer to the to the line than others. So in some cases this classification will be kind of a bit more unsure. Maybe if that makes sense. So you can see here that even if the line even kind of wiggles a little bit into one or the other direction, then that will affect these points which are closer to that line. And I think this is a really important concept to understand that whenever we try to classify and kind of separate the space, there will be some points for which a small change to the equation to the hyperplane will have a much larger effect than for other points for which the classification is much clearer. Yes. So if we have this equation in this case this is our multi dimensional our p dimensional example. We have two dimensions here. Then we can basically say that the value for each of these points x will be larger of one larger than zero for points on one side and then smaller than zero for points on the other side. So we're basically calculating on which side of our line do these points for. Now, the interesting thing which has just said about how far are the points away from the line, is actually how we start thinking about this classifier and how we find the best separating line. So we call this the maximum margin classifier. The idea is that we want to draw the line in such a way that we are maximising the margins to all of these points, to kind of the closest points to the margin. So you can see here that we're trying to draw this line, this the solid line in the middle in such a way that the margin that's kind of a dotted line on the left and the right of the margin of the line, we want to maximise the space. So we want to maximise the the distance we have from the closest points. Why do we want to do that. Think back to our example here. If we have lines which are not optimal, then they will be too close to some of these points. You can see here. For example, we have this line which is really close to the point. We have this line which is really bad as well because it's much closer to this point. So we know that is kind of some space between these two classes. And we want to draw the line in such a way that we're maximising the space to the left and the right of our decision boundary of our line here. And of course, I mentioned earlier we make that into an optimisation problem because why not? The idea is that we're basically trying to maximise the margin m subject to our better parameters. Remember better will be for example, the intercept and the slope. So they are basically describing what the line looks like. So we want to maximise the margin by adjusting how the line lies in our space. And then we do that subject to a couple of restrictions. Because it's an optimisation problem. We have to come up with some kind of restrictions in those cases. So that's two. And one is a bit easier to understand than the other. So one is quite simple to understand, because this constraint here is basically just looking at each observation should be on the right side of the hyperplane. So provided M is positive because it's a margin, obviously it should be positive. We kind of look at each of these points and the value it has in relation to our line, and then we want to draw the line in such a way that the points lie in the right side of it. So the reason for that is here. You can see in this area here we could draw another line which might be further away from all of the points. So the margin might be larger. But all of these points would, would be on the wrong side of that line. So if we drew a line through here, it might be better in terms of size of the margin, but it would be worse in terms of classifying the points. So this is the best kind of place where we can put the line in such a way that the margin is maximised, while still making sure the points on the right side of it. And then there's this other restriction that we have. Don't think about that too much. It makes mathematical sense. The idea is basically very similar to to the above, that the sum of all of our beta values should be one, or the sum of the squared beta values should be one. That's a very long mathematical reason for that. I'm not going to bore you with it, but the the effect it has on our model is basically that each of the restriction, each of the observations is on the correct side of the hyperplane. So it's kind of doubling up on this with this constraint up here and is at least a distance m from the hyperplane. So that makes sense. We don't want any points within the margin. The margin can only go so far as the points go. So the margin is basically the minimum distance that we have because it's kind of the closest points. Determine how wide the margin can be. So these are the two restrictions that we have obviously. It doesn't always work out like that, because it doesn't. I mean, your data will not look like that and you know that. So in most cases, we have some kind of data which looks like this. So we will have like. Maybe a vaguely possible line through that somewhere that you can imagine that a line will probably be somewhere here, right? But it's not really separable by a linear boundary. And this is typically the case. And it's especially often the case if in this case, we say the sample size of samples is smaller than the number of predictors. So n is smaller than n a smaller than p, which would be the only case in which it is unlikely that this happens because then your sample size is just so small and the number of predictors is so large. Then there's a lot of space basically for the data to spread. And then it's easier to find a linear boundary. Typically we have something that we call non separable data or noisy data or something like that. And we are still trying to draw this dividing line through that somewhere. So let's look at the easier case first because that explains then how we treat the more complicated case. In some cases we have data which is separable by our linear margin, our linear line. So we can see that on the left we can draw that perfectly perfect separation. We do all it just the way we did earlier. But we would call this a noisy example. Or I would call this an example with kind of influential points. So you can see here on the right, for example, what happens if we add just a single extra point here. So you can see here on the left there was no point. Then we add a single blue extra point and suddenly our old line would be incorrect. And we should rather draw this new line. So there's a lot of change to the slope of our line just by adding the singular one. Observation. Now the question we could have is, is the singular observation really so important that we want to ensure our lines are perfectly separating our space? So how much importance do we give this kind of singular line at a singular point, the singular individual? Maybe then we say, okay, we're kind of shifting the whole classifier that we worked so hard on building. That this is something that we would call an influential point. Basically, it's a point that has a lot of influence over the model and the model that how the model is constructed. You will hear us talk some sometimes about the effect of influential points, and they are not necessarily outliers. We wouldn't call this an outlier because it doesn't really have kind of extreme values in one or the other. Dimension is just behaving a bit odd compared to the other points in the same class. So it's having a big influence on the model. So it's an influential point. And this is then giving us basically this idea of how do we expand our maximum margin classifier in such a way that they are allowing for these kind of influential points? So do we really want to change the whole classifier into this direction, or are we happy with some points affecting maybe the model or not affecting the model in such a way? Thinking about what to best do about the timing. Okay, I will go on a little bit longer before the break. So we talked about this idea of maximal margin classifiers. Keep that in mind because there was the original idea how a support vector machine was developed. But let's put it to the side for now. Support vector machines in general, just like all the other methods which we discussed. So just like our decision trees, for example, that we had earlier are a group of approaches. So we have different methods under the umbrella of support vector machines. And in general we can use them just the same way we can use decision trees for both classification and regression. We will focus on classification. Here they are much more popular for classification tasks. They can technically be used for regression. It is a bit more rare to use them that way. So you will mostly find support vector classifiers. But because they are so kind of popular, we still call them support vector machines. So it's a bit it's a bit muddied up. If you read Support Vector Machine, probably that will mean a support vector classifier, but we don't really use that term that often. The basic logic is still the same. We are still finding this maximal margin classifier. So it's still an optimisation problem where we're trying to maximise. You remember the margins around this classification line that we're drawing what can be kind of a classification hyperplane. So this line doesn't have to be a line. It can be in any number of dimensions. But what we do now is we allow for something because slack. So we're adding this ability of the model to not be so sensitive towards these influential individual points. So instead of kind of saying this one point will change the whole model, we say. Little music. So instead of saying we allow the singular edit point to change the whole setup of our model, we instead saying, okay, we will allow for a little bit of slack, a little bit of error. It doesn't have to be the perfect line to divide it. A bit more on terminology and the correct term to talk about the basic form as often as VM would be to call them linear support vector classifiers, because they are linear linear boundary and they use support vectors. You will hear more about that what that is later to classify. So linear support vector classifiers. No one really calls them that. Everyone just says SVM. They will maybe say linear SVM if it goes far, or like linear SVM for classification, but this would be the technically correct term. That being said, I did say earlier that we will use a trick to actually allow us to classify non data as well, so you can use them for linear separation cases. They work perfect for that. They might be a little overkill sometimes but they work. But you can do that and you usually use them for non linear cases. So let's look at the linear version first. The linear support vector classification is basically a variation of the model that we had earlier. So it's a maximal margin classifier. But we just add a little bit of slack a little bit of kind of looseness to the whole model. So this is the same thing you said earlier. We want to maximise maximise margin M we can do so by kind of changing the parameters which determine where the line lies in space. We still have this slightly weird constraint, which you just have to trust me that it works to make sure that the margin is never kind of two. The points are not within the margin. We're still trying to maximise, which we were still trying to solve this whole problem under the same constraint we had earlier, which ensures the points are on the right side. But this time we add a little term here one minus epsilon. And we want to ensure all of these epsilons are positive or zero. And we want to ensure that the sum of all of these terms is smaller than C. And C is our kind of tuning parameter that you have to choose. So this term here is adding slack. It is basically allowing for yes, all points should be on the right side of the classifier. In most cases, I guess. So kind of adding this a little. So a bit of kind of yeah. It doesn't restrict us too much on it. So we kind of adding this, this little bit of slack, but we still want to make sure that there's not too much of that. So we're still trying to find the line that is doing a good enough job of separating, but not a perfect job, if that makes sense. So what does that look like? These small kind of epsilon terms are all what we call slack variables. So they allow for individual observations to be on the wrong side of the margin of the hyperplane. And there are three cases which you might encounter. Either everything is perfect. The observation is on the right side of the margin and everything is great. Or the term might be a little higher than zero. And then the observations on the wrong side of the margin. So we would say it's kind of violating the margin. That's not good. So you can see here we have these kind of points which are within the margin. I said earlier we kind of we don't really want points in the margin, but not we're saying we don't want them, but for some points we kind of allow it a little bit. And then sometimes you have points which are on the completely wrong side of the hyperplane. And in those cases our error terms would actually be larger than one. So this is basically a gradual range of how large is the error we're making. It's zero if everything is great when making no error is larger than it's larger than zero, it's kind of moving into the wrong direction. You can see here observation eight is moving away from its group of blue points towards the margin towards the the boundary. It's within the margin. So that's not good. And the same for this observation one here kind of moving towards it. So this would be an epsilon which is larger than zero. And then in some cases everything goes wrong. Epsilon is larger than one for an observation. And then it's just a completely wrong side here. For example observation 11 wrong side of the plane. It should be on this side. Same with observation number 12. One side of the hyperplane should be on the other side with all the other blue points. So we're kind of calculating how large is our error per observation. It's in theory it's the same kind of principle that we also had for regression problems. For example when we were calculating the error of the points. So you remember when we were talking about linear regression, how far away is each of the observations from the regression line? And we kind of summed up all of our errors that we were making and perfectly predicting where the points should lie. And then we kind of wanted to keep that error minimal. Right. And this is a very simple, similar process. We calculate the error. There are different types of errors and that kind of increase in how bad they are. And then we restrict the sum that we make in total. So the sum of all errors should then be constrained or restricted. There will be a maximum to that. And before we go on let's take a bit of a break so that you have a chance to ask a couple of questions as well. And we will go on in ten ish minutes. All right. Go ahead. Okay, so we just finished talking or we haven't finished. We kind of halfway stopped by talking through our slack variables. So I was asked. Yes, indeed. There are one slack variable for each of our observations. So you can really think about them as your error terms in regression. But you also calculate an error term per observation for for which you calculate how far it is away from the line. And in this case you calculate basically not only not really how far it is away from the line, but whether it's on the right or the wrong side of the line. So you calculate whether it is correctly classified, incorrectly classified, or correctly classified, but it's violating your margin. So these are kind of the three cases which you can encounter. And that's calculated for each of our observations. And then we said the sum of all of these. So remember they will always be positive. So the sum of all of them will also be positive. And the sum should be smaller or equal to c. So what is c. C is our regularisation parameter. So this is a number that you choose. It's your tuning parameter which you choose to decide how much slack you're allowing in total over the whole model. So how kind of how many wrong classifications do you allow? How many violations of your margins do you allow? And the sum of all of that, that is your parameter. C you will see in the computer labs how you will see in the computer labs. You will see in the computer labs that that is really kind of one of the input parameters for the function of, of your SVM model in in scikit learn as well, there is a default value for it. I think it's one I'm not entirely sure, but you can you will see how what what happens if you kind of decrease it and increase it. It always has to be positive. It can be smaller than one. Remember that our epsilons are smaller than one if they are just violating the margins, but are still correctly classified. So the sum of all of them C can be smaller than one, but it's always at least zero, which would be perfect. What that looks like visualised is basically it's allowing. It's deciding how wide and how never our margins are. So how do we draw our margins so that we have kind of the smallest amount of violations against those margins? You can see here we have really wide margins on the top left. So we are allowing for a lot of violations of the margin. And we have really narrow ones here on the bottom right, where we're not allowing for a lot of violations of the margin. So we don't want a lot of points to fall within them. So we're drawing them smaller. The model is kind of more restricted in that sense. Yeah. It determines the number and the severity of violations to the margin and the hyperplane at the same time. And it's basically like a budget. So how many violations and total do you allow for. And if you have C zero there is no budget. Everything has to be perfect. Every point has to be exactly on kind of the right side of the boundary and not within your margins. And then you can increase it. And the more you increase it, the more you allow for these errors to be made. In practice, that means C is basically our tuning parameter. So you can choose that typically via cross-validation because it depends on how well separable your data is. So if it's very well separable then obviously you will find a good line with fewer violations. If it's not very good separable, there will be violations. And you're just trying to restrict how many there are. You can also think about C as a control factor for your bias variance trade off. So you remember that we were talking about that. How narrow do we want to fit our model. How restricted should it be to the training data restriction to the training data can be good because it can increase accuracy, but there's a limit that you can reach at some point. The model is so narrow and so restricted to learning from the training data that it's not performing well on your test data. So if that happens, we would say the model is overfit because it's working too well on training and not at all on test data. There's actually a way of seeing that quite well if you look at your accuracy scores, if you have a score, if you have really good training accuracy and not a very good test accuracy, that's a sign that your model is overfit because it's performing better, much better on the training data than the test data. Typically, your model will perform a little better on the training data, but you don't want it to perform perfectly on training and then not at all well on test data. That's a clear sign for overfit model. And in that case, you want to kind of, for example, loosen the restriction a little bit if you're fitting an SVM. So what does that mean in terms of the value of C? A small value of C gives you narrow margins. So you are really looking for rare violations. You remember it's your budget. So a small budget for errors that would mean for your model you have low bias but a high variance. And if C is then increasing the margin is getting wider. You're allowing for more errors to be made. That means that you might be more biased, but you have a lower variance for the model. So in other in other ways, in other words, you fit to the training data might not be perfect, but you're kind of able to generalise better from that. So a larger C allowing you to generalise better to the test data. I think this is a really important concept to remember because when you hear see is your budget of error, you might think, well, I don't want to make any errors. Surely C should always be zero or should be at least be extremely small. But then you're falling into this trap of overfitting your model. A C value that is too small will overfit your SVM, so you will have to allow for little bit of slack. Remember that each of these observations adds to your error, so if you have a lot of observations, you're naturally kind of adding to your error. For example, because you just having more and more points, which could have a little bit of a violation. So in that case is it's really important to allow for a little bit. Now, what we've said so far was all about linear boundaries. So we had just some kind of space, two dimensional lines with three dimensional kind of a plate or hyperplane with it, but it sometimes just doesn't work. So sometimes your data looks like this, which for anyone with a linear model is a nightmare. You can see I try to draw through a linear boundary through it and obviously that's it. That's just not it. It's terrible. You have these pink points here in the middle. You have your blue points here and your blue points there. And there is just no way a linear boundary will be able to divide that into two classes. So it doesn't really matter what you see value is, it doesn't matter how you allow your boundaries or your margins to be. It's just not going to happen. So what do we do now? There are a couple ways of doing that. The first thing people tried was to let's just enlarge the feature space. So we use functions of the predictors instead of using the predictors themselves. You remember that it's a bit similar to thinking about hey let's just use a logarithmic transformation on all our data. Or in this case let's use a quadratic term of all of our predictors instead of just using the predictors as they are. So sometimes that can work. So in this case for example we would use quadratic functions of all of our values of x. And we would fit our SVM using a linear function. And it would work just well. The problem is exactly that. It becomes terribly complicated. It becomes really difficult to compute. So it's computationally very inefficient. So here for example, we still have our same maximisation problem. Maximise our our our value of m. So that's that was our margin. We're still trying to kind of fit it in such a way that the values are on the right side ish, allowing for slack. We're still trying to restrict the number of slack that we have in total, and we're still trying to make sure that they're on the right side and not within the margin if possible. So there are a lot of different ways of enlarging the feature space. So you can use quadratic functions and all of these. The problem is, if you try that, in the end, you often end up with just a really large number of features. And if you think about each feature is one of your variables and you think about having thousands of variables, suddenly you're looking at data which is just unmanageable. It's getting too large. So it's possible, but it's really inefficient. So what people thought is let's use something which we call the kernel trick. And the kernel trick is really quite clever because it allows us to use a linear boundary in non linear separation cases. So the idea is if we talk about support vector machines, typically we talk about the support vector machine using a kernel. So for a non linear problem technically I added a little note down here all the previous model. The previous model that we talked about was also a support vector machine. We sometimes call them linear support vector machines. And you will see in scikit learn that you can also fit your normal support vector classifier and just add linear as a kernel as a kernel. And then you're fitting what we are just talking about in the model previously. But when you read in the literature, in most cases, what they mean is an SVM with a kernel trick applied, and specifically in SVM with a Gaussian kernel is extremely popular. The idea behind this kernel approach is that we want to enlarge our feature space in such a way that we can draw a linear boundary through it and still separate a space, a computationally efficient matter. So there's a bit of maths involved for that. Specifically, we look at inner products of two vectors. Now I do have all the maths on here. The idea. If you don't remember what inner products are from linear algebra, I don't blame you. But the idea behind an inner product is just. We're looking for ways of similarity describing similarity between two vectors. So for example, the point product between two vectors would be described through their length that I think about as vectors as kind of arrows and their the angle, the angle between them. So in which how long do the vectors go and what is the angle between these vectors. That would be their point product. So it's a singular number with which we can describe how these two vectors relate to each other. And these inner products can then just be generalised. So that's the whole idea behind inner products. Here we're looking at similarity or relationship between two vectors. Don't worry about this too much. And if you think about two vectors a and d for example, we would describe the inner product as such. The sum of all of these of all of these observations up to J. Um. And then you take the product of those two. Think about them as kind of relational numbers. For vectors. That is good enough. The important thing is that if we do that for our linear support vector classifier, we have one of these observations per training observation. So we have one of these terms per training observation. So it's getting really kind of very abstract in a sense. We have to calculate the inner product between each new point and each of our training points, which is extremely expensive. The good thing is, and now we're coming finally to the part where I'm telling you why a support vector machine is called a support vector machine. We don't do that for each of our observations. So if you have a really large space and to think about where do we find. Uh, where do we find these lines? We don't think about how do we fit the lines in relation to all of the points we're basically looking for? Let me find a good plot for that. We're basically looking for what we call the support vectors. These are kind of the anchoring points which are deciding where our line should be. So in this case, for example, this might be this point and this point and this point. They we call them support vectors. Because remember each of these points is described through a vector. Each of your observations can be described through a vector of the dimension of the number of variables that you have for that for that point. So that's why we call them support vectors. They're supporting our hyperplane or our line or whatever in space. So deciding where exactly the line should lie. Uh, where are we? Exactly. So the good thing is that this some this this this value alpha here. Gosh, this value alpha here is non-zero only for the support vectors. So any training observation that is not a support vector. Think about those training observations far away from the line. We don't really care about them. Then alpha would be zero. So in other in other words we just look at these support points, support vectors. And then we can relatively easily calculate where exactly our our line, our dividing line should, should lie in space okay. Now comes the kernel idea. The idea is that we placing this inner product here. We just decided alpha is only ever a value for our support vectors. Now we're looking at this inner product between x and x I. And we are using the kernel approach to replace this inner product. I found kernel functions and kernel approaches terribly confusing and complicated, and I didn't really know. Okay, what does that exactly mean? So I always think about them. It's kind of. This is a bit silly. When I first heard the word Colonel, if English is not your first language, I thought about popcorn kernels, like little kind of popcorn kernels that you use. And then I thought, okay, what happens if I, for example, have. A two dimensional space, right. So I'm kind of two dimensional space. What happens if I have a popcorn kernel and I put it up in the middle of it, kind of pushing it upwards in three dimensions. So that's how I think about kernels. They're kind of little things that push our dimensions up by one. So two dimensions are pushed up into three dimensions. One dimension would be pushed up into two dimensions. That's how I think about kernels. I think this is much easier to remember than all of that. And then the question is what is the shape of your little kernel? So I get a little popcorn kernel and a comment can come in different shapes. So the most common shape that I mentioned earlier is the Gaussian kernel. So I think about kind of a Gaussian shape. That's that's a little kernel. You're pushing that up towards your space. You can have polynomial kernels. You can have radial kernels, different kernel shapes for different application cases. It's the same as usual. Which kernel to choose depends on your data. And your data will decide the shape that you need to kind of push the data away from its kind of simple dimensions. Let's visualise that because it helps think about this case. This looks very similar to what we had earlier, right? We have kind of this space here in the middle. We have these red points and then we have a lot of purple points surrounding it. And you think there's no way we can draw a linear boundary through that, but we actually can if we push that two dimensional space into three dimensions. So if we use a kernel here, for example, this should be a Gaussian kernel. You can see that suddenly it becomes this shape in three dimensions. Now all the red points at the bottom or the purple points are in the air. And you can draw this kind of green space, this green flat surface through it and perfectly separate the two. So this is the idea of the kernel trick. I also found this really pretty visualisation for that. So similar example we have points in the middle, we have points surrounding it and we are pushing the kernel through it. And suddenly we can find a lot. We can find a hyperplane that can actually divide these spaces. So this is how I remember the kernel trick. I think it's much more intuitive thinking about all the others, all the mathematical equations which are really important. So we find this hyperplane which can divide it in this additional space. And then if you look back at it in two dimensions, what we really found is this perfect dividing circle. So this is the idea behind the kernel trick. It's a way of basically allowing us to divide spaces, which shouldn't be divided by linear boundaries. By pushing them into additional dimensions. So we mentioned earlier, I have different shapes of kernels. And in this case, for example, it's relatively easy to see this should be a Gaussian kernel. So this is the perfect space for that. We can push it up or push it down boards and then find a perfect separation to that. But there will be other spaces. You will see a couple of these examples in in the in the computer lab tomorrow as well. So we think okay. Yes. Okay. I thought something happened. So you test a number of different kernels tomorrow. The problem is it's really difficult to tell in advance what should be the perfect shape. So in some cases it is relatively easy. You would plot it and you would have a look at what is a realistic kernel kind of kernel shape to take. In most cases what we do is we do some type of cross validation again. So we will use Gridsearchcv in the computer lab tomorrow we will test different kernel shapes. We will test different values of C. And then in the end you basically look for the perfect combination, the perfect amount of slack that you allow, and the perfect shape of kernel that you would like to look for. So the advantage of working with kernels, they allow us to separate non-linearity without really working in this and large feature space. So working in a large feature spaces is computationally very expensive. So what we basically do is we just look at our support vectors, our supporting points, we decide this is where we want to fit, basically the kernel function to it. And this allows us to, for example, divide. This is what we had earlier. You can see now this is this would fit with a polynomial kernel with degree three perfectly. Or you have this example on the right here. This is a beautiful example for a Gaussian kernel. So you can you can plot these spaces and then you can try different numbers of kernels. Yeah. This is a really good example for different values of this kernel. So you will see in the computer lab tomorrow as well that obviously we just said this is a polynomial kernel with degree three. And I know that because I tested different values for the degree of the polynomial kernel. So yes, you have a lot of parameter fitting to do unfortunately. So there's some models which work with much fewer parameters. In SVMs you do have to make choices on C, the regularisation parameter, the shape of the kernel and any kind of degrees and parameters and all of that relating to that kernel as well. So these are like your three parameters of your three decisions that you have to make for your SVM. It's still relatively easy to fit the. Okay. So we've talked a lot about support vector machines. I'm sure you have a couple of questions, but let's first talk to your neighbours. Again. Think about support vector machines. You can think about them in comparison to other methods. You can think about what exactly we discussed, what might be advantages, disadvantages. Your chance to think about what I told you in my beautiful monologue all morning long. And then we will collect your points again. So take ten minutes or so, and then we'll talk again at 1130. Okay. How do you like my clapping? I always feel I would like a teacher like. Ah, you can do like. Wasn't it this fox? This. Be quiet. Fox. Is that something you teachers did in school? No. Okay, good. Yeah. Who would like to start? We have advantages, disadvantages or comments that you would like to make or questions you want to ask. Should we start in the middle for change? It's not. Really applicable in. Formal. Declassification, so it has to be applied. Yeah. So multi-class classification is a bit tricky. You're absolutely right. So it is possible. Be difficult. What basically does is it uses multiple binary models and then kind of fits them together for multi-class problem. So each of these steps we do is basically doing binary classification. And you have to combine multiple different ones for a multi-class problem. So it's possible SVMs can be used for multiclass, but it's not as straightforward as other cases. Absolutely right. Something else in the middle. Yes. I think that you said is really sensitive to outliers when building a model. Like if you. Consider the outliers, it will generate another new model. And if you reject the outliers including. Yes. So it's sensitive to outliers and noise in particular. So thanks to our kind of slack variable, we can allow for that to work a little bit better. But if you really have an outlier on the completely other side of the data set, then SVMs can be a bit bit more tricky to fit. Absolutely right. So let's call that sensitive to outliers and noise. So. Yes it depends. It depends a bit on, on your, on your kind of cost function that you're using. Basically on the, on the c parameter that you're using. But you have to clean carefully before using an SVM. Yeah. Any advantages? Yes. What would you. Right. So here is more. If that parameter would lead us to generalise. Uh huh. Yeah. So basically so to control overfitting. Right. So you really have this one parameter that you choose, which you can carefully choose in such a way that you avoid overfitting with the model. So we were talking about how do we fit the margins, how close do we want to fit them, how strict do we want to be. And. And this is, for example, in contrast to, I don't know, dual networks, for example, we have a lot of parameters that you have to fit. So it can be the tuning process can be more complex in that sense. Or you have things like clustering, which it's difficult to compare to that. But it was even more difficult to control how the model is performing. So here we have this kind of control or tuning parameters C with which you can control overfitting for example. Yes. Very good. But there's also a disadvantage with that namely. Yeah. Do you have to choose C. So you have to have. You have to tune your parameters. And you not only have to tune your parameters. So that's your C and then any parameters of your kernel. But you also have to choose your kernel. And that is tricky. So we saw these relatively simple examples in two dimensions where I just told you, hey you have these two dimensions. Plot them. Hey this looks like a Gaussian kernel. Well what do you do if you have 50 dimensions, you can't really plot 50 dimensions and then decide how your Gaussian kernel will fit that. In those cases, you have to carefully choose the kernel, typically based on cross-validation, and you do the same for your parameter c, and for any parameters that the kernel comes with depending on its shape. So this is parameter tuning. It's not easy. I have five advantages on my paper. For that, I shall be more. Yes. Is it because. Why does it allow us to do to specify non-linear? Yes. So non-linear classification. Thanks. Two kernel trick. That's really handy. We were talking about linear classification, about linear regression, and how one of the major disadvantages is the linearity of the model. So we're avoiding the linearity here by using the kernel trick. So yes that's a big one non linear. What else do we have on my paper. Mhm. Yeah. So that's one of the reasons why many people use SVMs. Handles high dimensional data well. So it actually works really well in high dimensions compared to other methods compared to other models which kind of struggle. We talked about k and N, you remember that Kanan does not like many dimensions SVMs do. That's great to big data, right? Yeah, it is computationally a little expensive. So it does not perform very well with large datasets. Computationally expensive, so it's not super expensive. Like it's not terrible. You can use it on fairly large data, but it really shines on like medium size, like medium size data, a lot of dimensions. SVMs are amazing for that. And when I talk medium size, we're talking like a couple thousands, 10,000 data points depending on how good your computer works. Yeah, the same point. With large data sets, it's harder to. Yes. So that's one of the reasons why they also struggle. They're not only computationally expensive, but if you have a lot of data points, it can also become more difficult to find a separating space. So it depends a little bit because the data set can be really large but well separated. But in many cases the more data you collect, the more randomness you will have in your data, so the noisier your data will naturally be. So you have to be more careful. Which comes back to this idea of parameter tuning and kind of choosing your C a bit depending on the size of your data set as well. So allowing for more noise and allowing for more violations to that. Yes. Um, I said something at the very beginning about SVMs. Very general. Huh? So they perform pretty well. I mean, that's something we can say as we are generally a pretty good classifier. We said about decision trees, individual single decision trees, they are not performing very well on their own SVMs. You don't need any kind of bagging approaches or anything like that. They perform fairly well on their own, so they are pretty good classifier and kind of one model, if that makes sense. Do we have that? We have that. We have this. I have two more disadvantages. Yes. Sorry. Interpretability? Yes. Interpretability is extremely difficult. Its ability. So compared to decision trees, for example, compared to K and N, they were all really easy to explain. You notice that when I was explaining them to you, it was really easy, right? You could understand them quickly as VMs. I mean, I hope you understand how they work, but if you imagine having to write a business report and trying to explain how an SVM works and you start explaining, well, we have these support vectors, they are like these important points. And then we use a kernel trick. And we kind of pushed it into a higher feature space. And then we use a dividing hyperplane. It's really kind of tricky right. So they are more difficult to explain. They are also pretty black box. Not a pretty big box but kind of I mean yeah. So you can interpret them to a degree. So they are not like a neural network which is completely black. But SVMs are extremely difficult to kind of get to how exactly they make how exactly they make the decisions, especially if you have a lot of dimensions. So if it's a high dimensional case, a lot of data points, then they are difficult to interpret, difficult to look into, even if you do have the technical knowledge. So the way I kind of differentiate between interpretability and black box model, to me, interpretability is about being able to communicate how the model works. And a black box model is even if you have the technical knowledge and you are the model builder, how well are you able to look into the model and explain why it made the decisions that it made? So one more, which I would like to add here is scale sensitivity. And that is a really important one. Whenever you have a model where we're looking at kind of multidimensional distances and this is all we're doing right. Think back to your maximum margin classifier. We're looking at the line and we're looking kind of at the distance of the points to that line. So that only works if the scales that you have are the same. So you have to do a lot of careful cleaning. You have to clean your outliers. You have to remove a kind of these or identify these influential points. And you have to take care of the different scales that you might have in your data. So that's really important for SVMs. And we talked about decision trees which don't care about about scales. So there would be a big difference between the two. That's those two. And then one last advantage I would like to add to that is work for regression and classification. So that's a really important example for this as well. Remember we were talking about for example logistic regression. Logistic regression is a binary classifier. Full stop. That's just what the model does. We talk about linear regression regression only. We just have a numeric output. SVMs can be adjusted to work for both classification and regression problems. We only covered classification here, but there can be used for regression as well. So in this in a similar way to decision trees, you would then basically predict the numeric value for each of the new test points instead of the class membership. So they work in a similar way to decision trees in that sense. But they can be used for both, which makes them fairly. Versatile. So svm's in a summary great for high dimensional data, good for kind of medium sized data sets that you have. Quite powerful, difficult to explain and kind of difficult to tune. That's the high level of overview. Yeah. For. Yes. The. Yes. The reason for that is think about the spaces that you think that you decide. Decide. So if you're trying to draw a hyperplane through that space, if you have mixed data, in theory it might work, but it might completely mess up your optimisation problem. So I haven't seen a well working SVM on, for example, on, for example, categorical data or binary data or anything like that. So numeric data is the best fit. Mm. So if we are using categorical data like we want hot encoding that so that it is numerical, it will add more dimension. Yeah. So you. Can. There's always a bit of a story of caution. What are you really saying? By one hot encoding, you're basically creating a lot of binary variables. And binary variables are really difficult for these types of models to handle. Because if you think back to our kind of example of drawing the points and drawing a line through them, if you have binary data. The points are not really spread. There's not really kind of space for which you would draw your line. So there's a lot of models which work in theory for binary data, so I'm sure you can do it. How much information that variable stand adding to your model as a whole? Different question. Okay. Are you happy with that? Ish. More or less. Happy? Very good. So I will be here for a couple more minutes if you have any other questions. Otherwise, I will see you tomorrow. Also, remember that next week is our last lecture. So next we will cover neural networks. That will be the last of it. And therefore if you have any questions any final course questions please bring them with you next week. And then we can discuss them. Okay. Thank you.