**Isabelle Guyon:** 0:00

The vectors are actually the coefficients, or the features, the components that you compute, that get into the system. So the vector is the list or the collection of all the information that you gathered. That includes my example of medical classification. It could include the age, the amounts of certain proteins, the family history, anything that goes into your medical files. The amazing thing is that it looks like this is just a very simple training strategy, to predict what comes next from the past part of the sentence. You can get these chatbots like chatGPT that give very sophisticated answers, but this should not fool us, because really we've trained parrots.

**Craig Smith:** 0:46

Hi, I'm Craig Smith and this is Eye on AI. It's an increasingly confusing time in machine learning, with generative AI consuming all the oxygen and a plethora of startups offering new services based on large language models. While I'll continue to talk to select founders, I'm going to try to get back to fundamental research and pay attention to the many avenues that have sort of fallen by the wayside but are still pushing AI forward. This week I speak to Isabel Guillaume, a French researcher in machine learning and pattern recognition. She's known for her work on support vector machines, otherwise known as SVMs, a type of supervised learning algorithm used for classification and regression analysis. Svms are based on the idea of finding the boundary that separates data into different classes with the largest possible margin. Support vector machines are still widely used today in a variety of applications, including image classification, text classification and bioinformatics. Svms are particularly useful when dealing with high dimensional data. While deep learning has dominated AI in recent years, svms remain a powerful tool in the machine learning toolkit. I hope you find the conversation as informative as I did. Before we begin, I want to give a shout out to our sponsor, YouAi. They're a fascinating company that does a number of things, but I'm most interested in their Mind Studio, which is a platform for building AIs on top of large language models that you can deploy for free or for profit. Think back to the days when smartphone apps were first getting started, and if you were able to have been among the first couple of thousand people to build a smartphone app, you would have learned a lot and may have earned a lot of money. So give Mind Studio a try. Visit <https://bit.ly/MindStudioEyeonAI> to build a platform. Mind Studio to build your own AI today.

**Isabelle Guyon:** 3:18

I'm a professor at Université Paris and I'm in the detachment at Google as a director of research in San Francisco. I'm also president of a nonprofit organization called Chalern, which is dedicated to organizing challenges in machine learning, and I've had a long time interest in benchmarking machine learning algorithms. As early as during my thesis, I was already connecting data samples with handwritten digits and I connected one of the very first data sets called the little 1200 that had 1200 digits written by people at Bell Labs, while I was doing an internship there.

**Craig Smith:** 3:57

Yeah, when you were at Bell Labs. I'm guessing that's where you worked with Vladimir Vapnik and others on support vector machines. Is that right, or did your work on that start before that internship?

**Isabelle Guyon:** 4:09

Yeah, that's right. Actually, I started working towards the end of my PhD on kernel methods. I was guided by that, by one of my mentors, john Denker, who was with whom I was doing this internship at Bell Labs before the end of my PhD, and also I was working on methods that we now refer to as the kernel trick, that one of my co-advisors, leon Personas, had pointed out to me. There was a paper from Tommy Poggio in 1974 that had induced the polynomial kernel, and I was using that in my thesis to make Hopfield networks behave as nonlinear learning machines that are nonlinear in their parameters, to be more precise. So my PhD was originally on architectures of neural networks. I started off with studying Hopfield networks that were inspired by magnetic systems. The reason for that is that I was studying in a physics school, the School of Physics and Chemistry of Paris, where Marie Curie and Pierre Curie discovered a radium. So I was always very inspired by Marie Curie. So I'm very fortunate that I studied in that school. And, yes, initially I was a physicist and worked with Hopfield networks with my advisors, gerard Refus and Leon Personas, and so, yes, I was familiar with the kernel trick and I was already using it during my PhD, I was also familiar with large margin methods that is, the other component of the support vector machine algorithm, as there were people from Ecole Normale right across the street from my school who were working on methods called from minimum overlap. So these are methods that ensure stability and basically also see a large margin. And so, yeah, when I started working at Bell Labs, the situation was very different from when I did my internship. Jan LeCoune and Bernard Boxer to young men had just joined and there was a lot of excitement around back propagation networks that at the time were called multi-layer perceptrons. And I was strongly advised by my department head, Larry Jackal, to work on backprop nets. And so I started working on them and I did some work with Jan LeCoune on applying convolutional neural networks to pen computers. So how to replace the keyboard with a pen. The time we thought this was very promising and handwriting would be a good user interface. We were proven wrong and only basically two years after I joined Bell Labs, Vladimir Vapny came. At the very beginning he shared my office because there was no office space for him yet. So I was very lucky to have many long conversations with him and he talked about this algorithm for optimal margin that he had an appendix of his book and that nobody had ever implemented. And I thought this was interesting. I mentioned to him work that I had done around that and I was like, oh yeah, I should do that, I should implement that algorithm, but I was busy with many other things so I never did it. And then all of a sudden my husband got a position at UC Berkeley and he didn't have time to start a new project before we left and he was seeking a short thing he could do and he asked Vladimir Vapnyk what is some small project I could do? Well, we're waiting to leave for California. I can't start a new hardware project. He's a hardware guy, something small. And Vladimir Vapny said, oh well, implement my optimal margin classifier. And so he did. And I thought, oh my god, I'm going to be scooped. I really wanted to work on that. And then my husband, with Bernhard Bozer, the third co inventor of a support vector machine, came to me and said, oh yeah, it works nicely now. And Vladimir said I should move on to a nonlinear version of it and we should make products of the inputs to make it nonlinear. And I said, no, you shouldn't do that. You should use the kernel track. It's much more powerful. That way you have, almost for free, the possibility of plunging your problem in a very high dimensional space, but just essentially taking every dot product and replacing it by another similarity measure, and, for example, you're going to take every dot product and raise it to a certain power. He said, wow, this is simple. So he didn't really know how to do that with the algorithm that was in the appendix of that book, so I just rewrote the algorithm and he implemented it and, sure enough, it worked very nicely. Yeah, so to make a long story short, this is how the invention of SDM happened.

**Craig Smith:** 8:38

That's interesting. I didn't realize that you were already working on neural nets when you did that work or on back propagation, I should say because I'm a novice, so correct me if I'm wrong, but support vector machines are not deep learning. They're a much simpler way of classifying data and work very well on small data sets, Whereas deep learning you need very, very large data sets. Was there a connection between your work on the nonlinear support vector machine and back propagation, or were they very separate?

**Isabelle Guyon:** 9:12

No, they were not separate at all. They were not separate from me, it was. I should tell you why I didn't do the work on the support vector machine earlier than I did and I was dragging my feet to implement that algorithm. The reason is that I thought this would make that much of a difference because the way I was training neural networks was with a strategy very similar to support vector machines. I had realized early on that in order to effectively train, you need to be focusing on those examples that are hardest to learn, which ended up being called the support vectors, and that the examples that are closest to the decision boundary. So this is already underlying also the other works going on finding large margin algorithms that are stable. The idea is very simple. The idea is that if you have a variety of possibilities to separate examples and that happens often when you have very few examples and you are in a large dimensional space in particular so if you have very few examples, there are many ways in which you could create a decision boundary. There's no man's land between the decision boundary and the examples, and it's natural to say that you want to have the largest possible distance between the decision boundary and those examples, which is the idea of the large margin classifiers, the large stability classifiers, etc. How do you achieve that? Well, you actually achieve that by showing more often the examples that are hardest to learn. And this is what was underlying this algorithm that had been proposed by many people at UNS, close to my institute, when I was doing my PhD. What their algorithm was doing is that it was a modification of the very classical perceptron algorithm. In the very classical perceptron algorithm, you make a change in the weights when an example that is shown is misclassified and you don't make a change if the example is well classified. And the very minor change that they made in this algorithm is that they were going through all the examples of the training set and they made a change only for the example that was worst classified. And that simple change. You can show that asymptotically, the algorithm converges to the optimal margin classifier, so it reaches the same solution as the SVMA. I had many arguments in the way that he made about what was best. He told me. Well, you see, this algorithm only asymptotically converges. It is not guaranteed in a finite number of steps to get to the solution, but on the other hand, it's amenable to be combined with the very popular stochastic gradient descent methods that neural networks are trained with. Because the only thing that you need to do is train your neural network and you show more often the examples that are hard to learn. So in the extreme case, you would at each epoch, only show the example that is hardest to learn. But this is kind of inefficient because you have to do a forward propagation for every example, figure out which one is the worst one, and then only update the weights for that one example that is hardest to learn. So it's a waste. So you can do something in between to identify the examples that are hardest to learn and only do the updates on those. And this is what I had implemented and it worked very nicely. So I didn't really see a need to implement this algorithm. I was getting good results. Now, what happened is that when Bernard Bozome has been implemented the Vapniks algorithm he got good results and then I suggested to combine it with the kernel trick, which is the trick that I was using also extensively with other algorithms than the maximum margin algorithm already in my thesis and it's not a new idea, it comes from the 1960s. There's a paper about this idea of the kernel trick. Perhaps most interestingly, this paper was written in the same institute as Vapnik, and he never realized that the two algorithms could be combined. It was called the potential functional algorithm and his algorithm was the optimum margin algorithm. So it took 30 years before somebody had the idea of putting them together. My conjecture is that because they were fighting for funding, they wanted to differentiate their methods as much as possible, and so they had never considered putting them together. So at any rate, yeah, I finally pushed Bernhard to use the kernel trick with the optimum margin classifier, and then this gives this new algorithm. That basically sparked a big fire.

**Craig Smith:** 13:40

Yeah, it just occurs to me, for listeners that don't know what support vector machines are, that we should probably define them.

**Isabelle Guyon:** 13:47

Okay, let me try to explain that Without pen and a paper it's a little hard, but you stop me anytime with questions. So well, the auditors probably know about classification problems that have been popularized with AI and neural networks. We've seen a lot of applications of face recognition of objects, classification of text classification, for example, classifying between spam and ham, or classifying between bees and wasps, if you want to determine which insect is a good pollinating insect or a dangerous insect. Anyways, this has taken off recently using neural networks, for the most part because neural networks are very good at learning representations, whereas algorithms like support vector machines rely on the fact that you've already come up with a good representation. And this is not so obvious in the case of computer vision problems, because you basically start with an image that has been transformed into pixels, that you know little elements of image, and so the representation is bizarre. Right With the algorithm C is just little dots that don't seem to make so much sense. They don't have an idea of what the object is as a whole from these little dots. But for a lot of applications the representation that you start with makes more sense. For example, if you have a biomedical application and you're trying to classify whether patients have cancer or not, you may have a whole bunch of diagnoses like whether you have a sort of protein in your blood that is elevated. That is the case, for example, if you diagnose prostate cancer. Prostate cancer antigen is elevated often in prostate cancer patients. This is not going by itself to give you a certain diagnosis. But if you have many features together, like the age of the patient, family history of cancer, the weight of the patient, whether the patient has been exposed to certain chemicals, etc. These are all called biomarkers. If you use all of them you can try to create a classification method that will tell you with a certain confidence whether the person is at risk or not of a certain disease. And this is what these methods try to compute. They try to compute a number that rates the degree with which you believe that a certain person or a certain object belongs to a certain category. So you can do that with many different methods. But in machine learning what you do is that you get examples. So you get examples of all these different features and you have examples of normal patients and these, these patients, and then you try to determine whether you can combine these different coefficients, or these different features. We call them features, coefficients, biomarkers, etc. All these quantities that you've measured about the subjects of interest. You try to combine them and then come up with a single number. So how do you combine them? One of the easiest ways is that you turn them into numbers and you do a voting, which is a weighted sum. So you multiply each coefficient by a weight and then you average. So it's like voting where each voter would have a different number of votes. The sum of coefficients will be seen as more important than others. Well, these weights, these weights are subject to training. So they are adjusted during a session in which you show many examples, and then you try to make it so that, according to the combination that you are computing of these input coefficients, you can then separate, well, the training examples. So right, so you vote among these many features, you get this coefficient, that is the average vote, and you see if the average vote is higher than a certain threshold, I will say that this is a disease patient. If it's lower than the certain threshold, this will be patient. That is not at risk. So what does LVM do, more than any such method? Well, what it does is that if you come up with one such combination. What you want is to have the maximum distance between the disease patients, for example, in this example and the non-atrace patients. These are basically according to this rating that you've made, which is the voting that you've created according to this rating. If you would just represent all the patients on the line and look at what rating they get and you put on one side all the patients that have high rating and on the other side the patients that have low rating, you will have some that are borderline right, that have rating that is neither very high nor very low, and you would like to have the maximum gap between the ratings of the patients at high-risk and the patients that are at low-risk. And if your rating is such that you can make a really big gap, then you can be reasonably confident that you've made a good discrimination between these two types of patients. If you don't succeed in separating them well for example, if there is an overlap between the distribution of the patients at risk and the distribution of the patients that are not at risk then you can't really separate them well and then you kind of fail right. So the idea between a maximum margin classifier is that if, with the training examples, you can maximize this gap, then maybe when you get new patients that you haven't used for training, they will not be classified incorrectly. So you have basically this big gap. In the middle of the gap you can say, okay, this will be my decision threshold, I put it in the middle of the gap. Now, what you don't want is that patients that are in the wrong category climb over the decision threshold and end up on the wrong side. So if you have the maximum possible gap, then you can minimize your chance that the new patients will cross the gap and end up on the wrong side.

**Craig Smith:** 19:44

And this is primarily for labeled data. Is that right?

**Isabelle Guyon:** 19:48

is correct. So originally the original algorithm was created for classification problems with labeled data and, moreover, the labels are categorical, that is, it corresponds to different classes or categories. Later it was generalized by Weplic and other collaborators to the regression case in which you want to predict these not categories, but continuous values. For example, if you wanted to predict the age of somebody based on pictures, then this would be a regression problem, not a classification problem. And then it was also extended to unlabeled problems, so-called unsupervised problems, for example to the problem of determining the support of a density by Bernard Schalke, and also to some clustering methods that are also for unlabeled data. But the same idea always applies, and the idea is you create a no-man's-land, a big gap, so that you get some confidence that when you get new examples then they will not cross the border and end up making wrong decisions.

**Craig Smith:** 20:50

And at this point, because deep learning, back propagation, supervised learning, I should say deep learning has been so successful in classification. If you're building a model, what point do you decide? Oh, I should use deep learning or I'll just use a support vector.

**Isabelle Guyon:** 21:09

Again, I think the mistake people make is to oppose the methods, because they're completely complementary. They address essentially different problems. The deep learning methods address the problem of learning representations and once our presentations are learned, which can take a lot of compute resources and we see now that it's very common that big companies train for many days with very large forms of computers, a model that is then released publicly and we are very grateful that such companies are generous and share them with the public Then these models can be stripped off their last layer, that's basically making decisions, and used as a preprocessor to compute new features. So that's what's essentially missing with support vector machines is that they are not very good at coming up with good features, even though they can implicitly function in a very large feature space. I didn't explain the kernel trick, but I can explain that later. But essentially they still rely on original features that are man-made, that are the input, and so they can benefit greatly from using as a preprocessor the features computed by deep learning methods. And then, if you do that, then you can harvest the benefits of these methods because they can learn with very few examples and in fact there are many methods that are still called future learning methods that are essentially based on kernel methods. When I talk about kernel, it's a broader family to which support vector machines belong, and these are methods that you can call case-based or example-based methods that rely on memorizing examples. Essentially, the question is what are the vectors in support vector machines? The vectors are actually the coefficients or the features, the components that you compute, that get into the system. So the vector is the list or the collection of all the information that you gather. That includes, in my example, medical classification. It could include the age, the amounts of certain proteins, the family history, anything that goes into your medical file. So this would be a vector.

**Craig Smith:** 23:20

And why not take a stab at the kernel trick, since I don't understand it?

**Isabelle Guyon:** 23:25

Okay, so now I've only talked so far about the vanilla essentially support vector machine, the linear version that was already invented in the 60s by Vladimir Vapne and his collaborators. That consists in just making a vote among the inputs or the features that I was mentioning. To take this step further and go beyond linear classification, we can manipulate these features, and this was the first idea of Epic. Let's make products or sums, or let's make functions of these features and put that at the input of the classifier. In this way we're going to massage them first in many different ways and then hopefully this will create a richer vocabulary, in a sense that will make it possible to get more complex decision boundaries. That idea is not new. Already in the 60s by Rosenblatt and his collaborators, when they invented the perceptron, they in fact already had a first layer that was made of random functions. So you take the inputs and you combine them randomly, and so you expand your feature vector to a very large feature vector, and it can be shown that if you do that, then virtually every possible separation of a finite number of examples will be able to be made, provided the feature vector is long enough. So it's a question of the big question that people had in the 60s is whether the examples are linearly separable, which means that if you have some training examples, will you be able to compute a weighted sum such that all the training examples of the first class are on one side and all the examples of the other class are on the other side and you can just set a threshold on this combination and you can classify 100% correct all the training examples. So everybody was set on that problem and later people discovered that it's not really necessary to have all the examples of the training set perfectly separated to get good generalization on test examples. But at any rate this was really their focus, and so people had come up with that idea of let's make random functions and then we will have a much larger space of features and then, if that much larger space of feature is going to be much easier to be able to find a combination that will separate perfectly the training examples, so a linear combination. So, starting from this idea, then some people have observed that in fact you could use a different type of method, that is, an example-based method similar to what people call the nearest neighbor classifier, and transform it into one of these linear classifier methods. So how does it work? Well, first of all, let's understand the nearest neighbor classifier. It's a very simple method. You just calculate the distance between your new, unknown example with all the training example and you classify it according to the example that is closest to it. So if you have a new patient, compare that patient to all the patients you have in your database and find the medical record that is most similar to that patient's medical record and then classify that patient as disease or not disease occurring to this training example that you have. And so people invented many variants of this nearest neighbor method and those methods are, as a whole, known as kernel methods. And what is a kernel? Well, a kernel is very simple it's nothing but a similarity measure. I was talking about distance, but whenever you have a distance, you can also define a similarity measure. It's the opposite, right? The distance says how far away two examples are similarity measure, how resembling they are. Some similarity measures are very simple to compute. For example, if you just do correlation, correlation is a similarity measure. Everybody knows the correlation coefficient. So if you take two records and compute their correlation, then you compute their similarity in a way. Well, there are many ways in which you can compute similarities and you don't need to use personal correlation coefficient, which is basically a linear way of computing similarity. One popular way of computing similarity is to use a Gaussian kernel and that's a method for a distance like the Euclidean distance, and you raise it to the minus power, that distance square, and what the result is that you have high resemblance near the example that you compare with and then the similarity dies off quite quickly as you get farther away from the example. It doesn't die off gradually like the personal correlation coefficient, it just boom drops as you get far away. Another more dramatic way of dying far away would be to have just a similarity which is one in a neighborhood of a certain radius and then dies off completely. So these are various methods of doing that. So those people refer to that as kernels, but think of kernels as just similarity measures. So how do you actually compute a coefficient by which you're going to be classifying examples, with multiple examples, not just like with the one nearest neighbor example, which is the nearest example only, and base your decision on just that nearest neighbor? What if you know you wanted to take into account several neighbors and vote among these several neighbors? So what are these kernel methods? They basically vote among several neighbors according to how similar they are and according to some weights that decide which example is more trustworthy than another. And so, interestingly, you can use training algorithms that are very similar to those that are used for linear classifiers. But now for these kernel methods. One of them is the potential function algorithm that was invented in the sixteenths and it just consists in adding, in the process of learning, a new training example in the pool of examples that you're going to be used to make your decision only if that example was misclassified. Now you see, there is a parallel with the perceptron algorithm. The perceptron algorithm was saying you're going to modify the weights only if this example is misclassified, and here we're going to integrate it in the pool of examples only if it's misclassified. And so as you proceed with training, more and more, actually, you increment by one the weight of that particular example. Every time when you cycle through all the training examples, that example is again misclassified. You're going to cycle many times through your training example. It's very similar to what you do with the perceptron algorithm when you cycle many times through the training examples. Every time an example is misclassified, you change the weights a little bit. And in the case of the kernel method, every time you cycle through an example which is misclassified, you increment by one the weight of that example. Well, not surprisingly, you can show that in fact these two algorithms are exactly identical. They are a dual of one another. You can formulate one by rewriting the other one, and it's just what's, in mathematics, called a factorization trick. Basically, you change the parenthesis of position, you have these big summations over all the examples, all the weights, blah, blah, blah, blah, blah. Right, double summation and you put the parenthesis by grouping one thing in one way and the other thing in the other way, or you change the parenthesis of your position and you get one algorithm or the other. So it's very easy, and that's what's called the curl trick. Basically, the curl trick is just this computational trick that allows you to switch from one vision of the algorithm to the other. So the first vision of the algorithm is to compare your new, unknown example to all the training examples, one by one, calculate the similarity and then do a weighted sum over the similarities. The other vision is at training time. You're already going to do all these pre-calculations and you're going to come up with weights for all the inputs and that's what you're going to be using to make your calculations. So think of it as a kind of a compilation, at training time, of all the examples into one weight vector, as opposed to keeping these examples and making comparisons with them at test time. So why would you want to do that? Why would you want to use one vision or the other vision? Well, it's because of computational reasons. If you are in the simple linear kernel, well, the linear kernel, basically, is like the correlation coefficient. You can go from one method to the other in a very simple way. Either you take your new example to classify it compared to all the training examples, and therefore you have to make as many comparisons as you have training examples that you've kept In for the support vector machine algorithm. Those are called support vectors. You only have a few of them or you only make one comparison because you've compiled all the information in a weight vector and you only have to make one comparison at test time. So when will this not work? It seems likely. You would always want to do this one comparison. That costs you a lot less. Well, what kills you is that if the vector, if the input vector is of infinite dimension. Now, why would this happen? Well, it's because I told you can go from one method to the other with this factorization trick. But this assumes that your kernel can be expanded in a development, in a feature set development, and this expansion can be of infinite dimension. So when you go in one direction to the other. So if you start with a finite dimension vector and you try to kernelize your algorithm, that is very simple you replace your original linear algorithm by just using essentially the linear kernel, which is like the correlation. In that case it's hard to see why you would have a computational advantage to do that, because you replace just one calculation by many. But if you go the other way around, if now, instead of having the simple correlation coefficient kernel, you have a kernel like the Gaussian kernel I mentioned before, with that Gaussian kernel, unfortunately, if you want to use this kernel trick, you need to do an expansion of it and that expansion is going to be infinite. So that corresponds to an input space, to an infinite dimensional vector. So you don't ever want to see that. You want to operate in this kernel space, in this way making comparisons with vectors. Yeah, that's all there is to it. Not very complicated after all.

**Craig Smith:** 33:45

To some people. So this is supervised. You were saying that support vector machines can cluster so they can deal with unlabeled data, and supervised learning has certainly been on the deep learning side, very powerful and has many, many applications. I wanted to talk about the data problem or the data quality problem that you addressed in your talk at NeurIPS and the thing I didn't understand from that is how you address, how you correct or clean very large data sets. I understand benchmarks but an example is this chat GBT. Its responses in many cases are not accurate because the data is not clean. And a very small example is if you're asking it in natural language to describe a place and if that place, the place name out on the internet, which is the data source, exists in many different countries, there are a lot of facts that attach to that place name and chat GBT doesn't appear to differentiate. So you get a paragraph talking about the place that kind of mixes facts from different places in the world with the same name. So this data problem I mean. You mentioned in the talk that one solution is to improve data quality or reduce biases just to expand the volume of data, but that doesn't really work if there are problems in even that larger volume of data, and I understand that you want better curated data. But when you're working with large data sets, how do you curate the data to address these problems? So I don't know if I'm misunderstanding the talk, but that was my question coming out of the talk.

**Isabelle Guyon:** 35:40

Okay, well, yeah, thank you for raising this interesting issue. You mean it takes a village to raise a child, right? You know the saying. So it's going to take the entire world to train good learning machines. That's whatever we do. A single situation or effort is not going to be capable of providing data of the quality that we need to train well learning machines and also, to each rate, to make them better and better. My belief is that we cannot expect that in just a few months we can do the work we do by training a student and sending the student to school and to the university for 20 years or more. This is a very complex thing. What we learned during our education is not just to be a parrot, which is essentially what current language models are trying to do. They're trying to predict what's going to come afterwards. So basically, they're learning to be parrots. The amazing thing is that it looks like this is just a very simple training strategy, to predict what comes next from the past part of the sentence. You can get these chatbots like chat Gpt that give very sophisticated answers, but this should not fool us, because really we've trained parrots and if we want to go beyond that, it will require some constructed what I call model schooling, where we have real curricula to train models and real exams, like we have for students, to verify that the knowledge is properly acquired, and rectify eventually if there have been misunderstandings or wrong generalizations made by models. It's going to be tedious work compared to the excitement that there is now. It looks like you just dump more and more data and you get more and more intelligent models. Well, we're not getting really intelligent models. We're getting more savvy parrots.

**Craig Smith:** 37:39

More current models, but not necessarily more accurate models.

**Isabelle Guyon:** 37:43

Yeah, right, so what's the way forward? So, first of all, we have this new generation of benchmarks. One of them I mentioned in my talk, big Bench that tries to corner models and try to see if they are used beyond their original purpose, beyond their original envisioned application. What can we expect and this is one of the amazing things and also danger you put out to the public with a tool like a chat GPT and people are going to be very creative in what they ask for this tool and they will ask things that are completely out of the box. People have started asking these chatbots to program, so they return programs because they've been trained also on some code. Or they ask to solve math problems, which is pretty amazing. They can solve some simple math problems. We don't know, you know, what ideas people are going to come up with. They're going to ask about these methods. So, as a collective, we need to try to poke these models very hard and figure out what they are good at, what they're not good at, what are the dangers, etc. And iterate. What's test data today will become training data tomorrow and it will take a village to train these models.

**Craig Smith:** 38:55

But even on the question of bias, I can see that you, using less data but higher quality data, can improve outcomes. But these models require, as such I mean a large language model certainly require such vast amounts of data that how do you adjust for bias in a trillion parameters? I mean, how do you know that there's bias there until you happen across it, which has happened in the examples.

**Isabelle Guyon:** 39:25

Perhaps I will give you a disappointing answer. It's open area research. A lot of people are putting in a lot of effort at the moment on this problem. It's divided into three approaches. It can be pre-processing, so basically you can create data, or you can filter data in one way or another. In processing, where the learning machine, the process of learning, figures out that something goes wrong and discards the bad samples or re-weigh things, waits to make more to make fairer decisions. Or post-processing, the machine is already learned and you put some adjunct on top of it. That's some of the kind of the band A thing. Right, you try to filter out bad answers so that you minimize the problematic behavior of the machine.

**Craig Smith:** 40:14

And I realize I've already taken up almost an hour so I don't want to go too much longer. But can you draw a thread from your work and support vector machines and tell us what you're working on now? Is it this data problem and is there a clear logical line from one to the other, or have you sort of branched off into new areas of research over time?

**Isabelle Guyon:** 40:39

Well, what I think most people don't realize is that algorithms are just a means to an end, and I'm interested in solving problems. So I came up with this algorithm at some point because it was solving a problem I was interested in at the time. I've never tried since then to really use it when I didn't need it. I've needed it in my consulting practice when I was working on biomedical data, because it was particularly adequate for this kind of data. You can keep using them today, combined with deep learning. Evidently, people are doing it even without noticing. But even if you wanted to know how to keep training the representation with the support vector machine in the loop, this can be done. This can be done with something that is called Siamese neural networks, which is something that we also invented back at the time, and Siamese neural network is basically a neural network that computes a kernel that computes a similarity measure. So you can create a kernel with a deep network and then what you do is that you train your support vector machine with the fixed kernel, which is your big deep network. You can get the result of the weights of the support vector machine and then you can backpropagate through the machine and then retrain the weights of the kernel, which is the weights of the Siamese network, and you can then keep iterating with these two steps. Some people have been doing that or in just disregard the original support vector machine back algorithm and at the very beginning of my interview I was explaining to you that the reason why I wasn't immediately implementing the support vector machine algorithm when the Vapnik came, because I was using this other technique that was similar in spirit. That other technique has been formalized in an algorithm that really shows that stochastic gradient descent can be used to train support vector machines and it's implemented in the scikit-learn library. So what you can do is that you can train a deep network with the support vector machine algorithm using stochastic gradient descent. Everything can be done. But at the moment I'm working on other things and I'm particularly interested in branching out on using these very large language models in several applications, because I think they're doing amazing things and possibly I will need support vector machines at some point, but not necessarily. So it will depend on what are the issues that we are facing. Issues that we might face, indeed, are cases you mentioned when we have very little data. So the few-shot learning problem, few example problems. In that case, it makes sense to have a notion of support vector, a notion of examples that are unique, to represent your borderline cases, your special cases. But this is just one sub-problem of a much larger, much broader range of problems that we can face.

**Craig Smith:** 43:26

Do you have time for another question? Sure, I wanted to ask. You know, Jeff Hinton gave us a new algorithm, a forward-forward algorithm. I've been asking people for their impressions. He's famously interested in figuring out how the brain works, and forward-forward possibly gives an insight to that. But I'm just curious whether you've read the paper and what your thoughts are about how it might advance the field or be used.

**Isabelle Guyon:** 43:57

Well, Jeff Hinton has surprised us many times, so we don't know, maybe this algorithm will take off or will inspire other algorithms at some point to really advance the field. What's for sure is that about 10 years ago, before you knew the boom of deep learning, Jeff Hinton was pushing very much for unsupervised learning and using these methods that were developed by Yoshua Benjo and others of stacked autoencoders. And at the time DARPA organized an evaluation of deep learning and I was part of the evaluation team and we organized the challenge. Darpa pushed us to have a protocol to prove that unsupervised learning could do the job and that supervised learning wasn't really needed to learn representations. And we organized the challenge at that time and, sure enough, much to my surprise, it is true that you can train systems to learn representations in a completely unsupervised way and it cannot be beaten by supervised methods. And this is what we also see now with the boom of self-supervised learning. You can do almost everything unsupervised and add the supervised layer at the end, and that even is beneficial if you want to be robust against some sorts of bias. Yeah, so I'm all in favor of people like Jeff Hinton who do research a little bit out of the mainstream and continue pushing these ideas that could end up becoming very important in the future. But yeah, it's certainly something to keep an eye on.

**Craig Smith:** 45:25

Yeah, and you're famous, aren't you, for not following the mainstream through charting your own course.

**Isabelle Guyon:** 45:31

Yeah well, I don't know whether I'm famous for that, but I'm doing my best not to be influenced by the mainstream and try to venture in new directions, of course at the risk of falling into a hole, but I find this more fun than racing on the groom slopes.

**Craig Smith:** 45:46

That's it for this episode. I want to thank Isabel for her time. If you want to read a transcript of our conversation today, you can find one on our website, eyeonai. This episode in particular packs a lot of information, and so I encourage you to download and read it. In the meantime, remember the singularity may not be near, but AI is about to change your world, so pay attention. I also want to thank our sponsor, Mind Studio by YouAI, which is giving creators the opportunity to build and deploy generative AI apps for profit. Uai has an emerging AI marketplace and Mind Studio is the best way to build apps with generative AI. Anyone can do it. Mind Studio uses conversational language to program incredibly powerful AI tools. No coding knowledge is needed to start your AI business today. Check them out at UAI. That's Y-O-U-A-I dot AI and start building your AI app today.