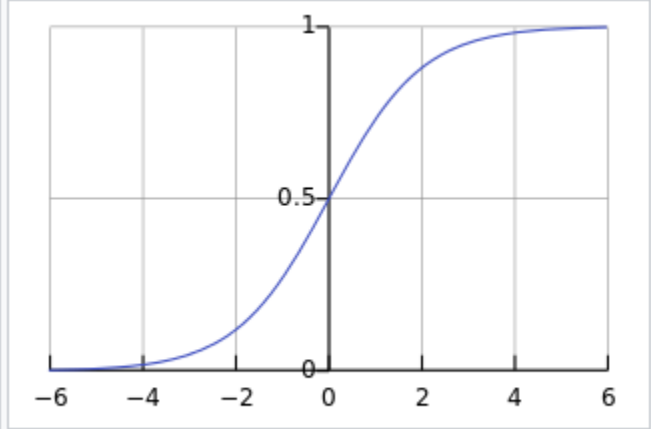
本文件的 notation: x^(i)表示第 i 個人的 feature vector, x^(i)是一個 vector, 例如 x^(小明) = (1.5, 100, 12), 即表示 小明的身高為 1.5 米, 體重為 100 斤, 年齡為 12 歲. x^(i)的分量記為 x^(i)_1 = 1.5, x^(i)_2 = 100, x^(i)_3 = 12.
θ^(i)也是一個 vector, 它是用來和 x^(i)來點乘(θ^(i) * x^(i))的, 例如 θ^(小明) = (0.2, 0.7, 1.8), 即 θ^(i)_1 = 0.2, θ^(i)_2 = 0.7, θ^(i)_3 = 1.8.
y^(i)是一個數(如+1 或-1), 表示第 i 個人的 lable, 例如 y^(小明) = 1, 表示小明會買這個東西.

The video time for this lesson is 78 minutes.

The recommended reading for this lesson is:
Logistic Regression
https://s3.amazonaws.com/content.udacity-data.com/courses/gt-cse6242/recommended+reading/logreg.pdf

From Wiki:
A logistic function or logistic curve is a common "S" shape (sigmoid curve), with equation:

$$f(x) = \frac{L}{1 + \mathrm{e}^{-k(x-x_0)}}$$

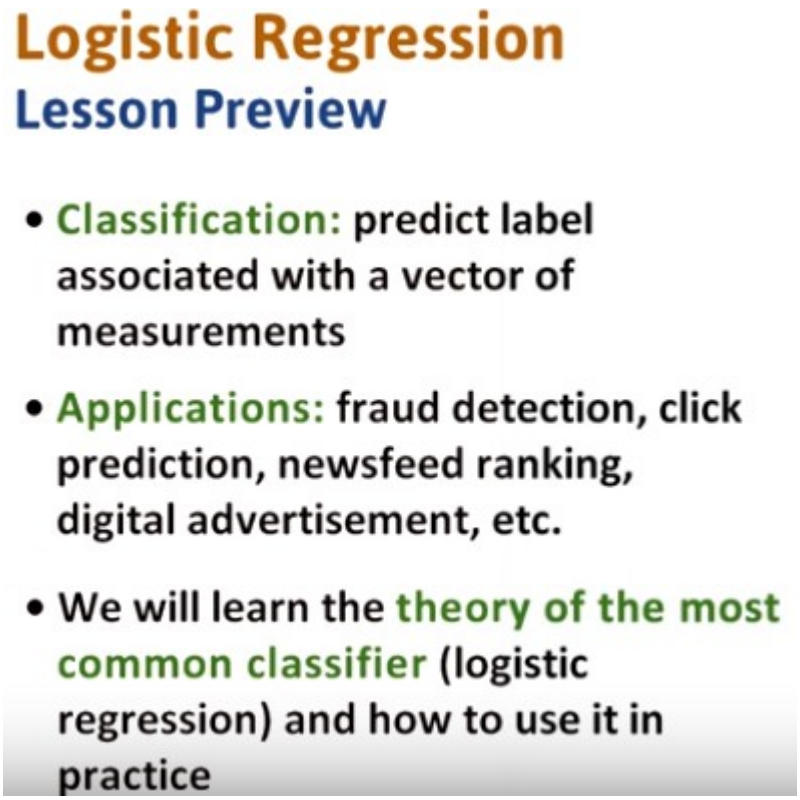


Standard logistic sigmoid function i.e. $k = 1, x_0 = 0$

下表中, “Predict the label”那一列(即用 y = sign(<θ, x>))就是 predict 一個-1 或 1 的結果,
而”Measure of confidence”那一列(即用 logistic 函數那個)就是 predict 一個在[0, 1]區間的結果, 或 predict 一個 probability.
它們都是 linear regression, 因為它們都是基於向量點乘.
而”Measure of confidence”那一列(即用 logistic 函數那個)就是 logistic regression.
為何 logistic regression 要用 exp? 因為這樣的函數使得 P(y = 1) + P(y = -1) = 1, 使得它更像 probability.

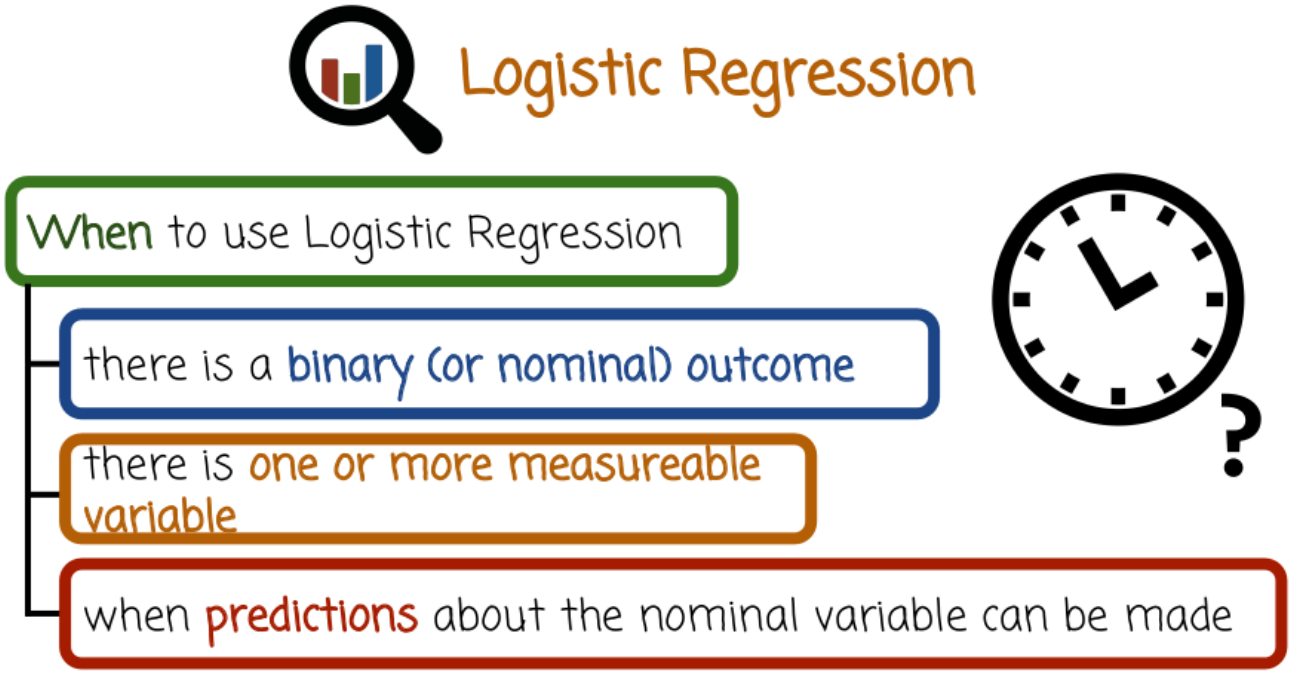
Task	Use
Predict the label associated with a feature vector x	prediction rule sign (θ, x)
Measure of confidence of that prediction	$p(Y = y \mid X = x) = 1/(1 + \exp(y<\theta, x>))$

正常 note 開始:

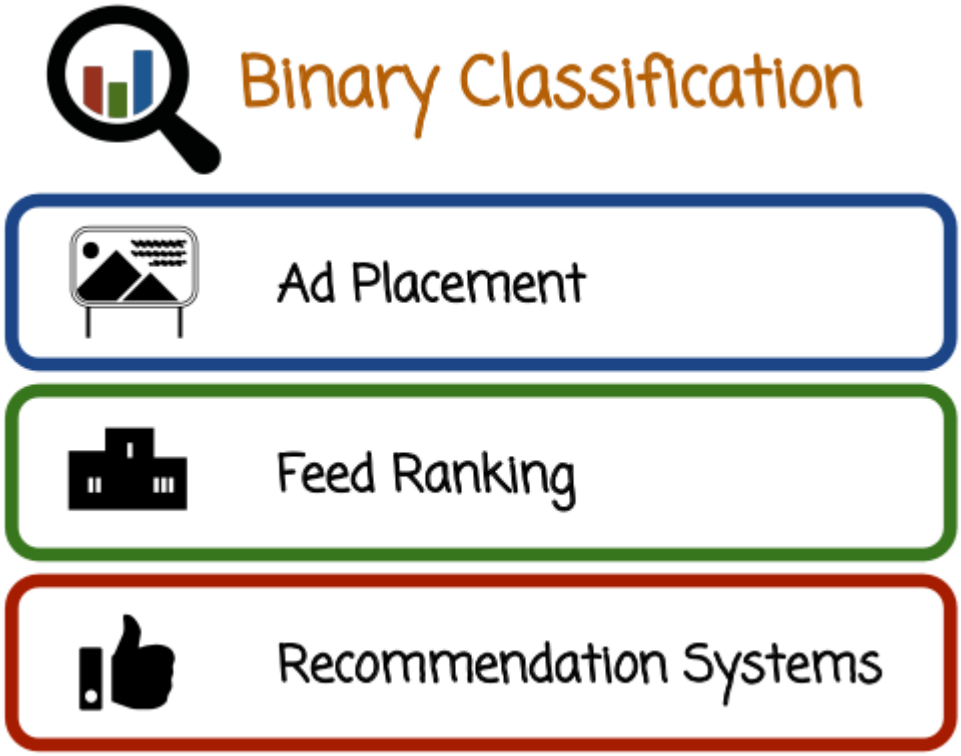


1. Classification is probably the most common problem in machine learning. Predicting the label associated with a vector of measurements. This problem is critical to many applications such as spam or fraud detection. The construction of a newsfeed like Facebook's and click prediction in digital advertising. One of the most common classifiers, if not the most common is logistic regression. It powers many of the machine learning systems in big companies like Google, LinkedIn, Facebook,

Netflix, and Amazon as well as many small start ups. In this lesson, we will learn the theory of logistic regression and how to use it in practice. Much of our study is going to be general, so it can also apply to other classifiers as well.



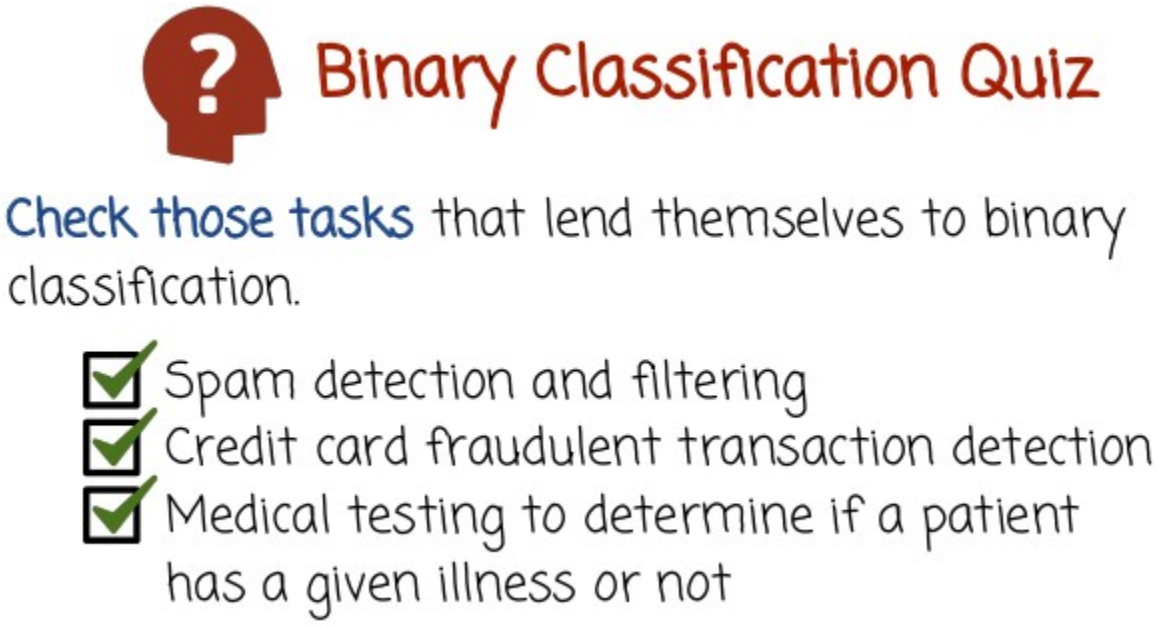
2. Let's see when is a good time to use logistic regression. Logistic regression is useful when there's one binary or nominal outcome. For example, we can think about whether a user clicks or does not click on a feature in a website. Or perhaps whether a patient does or does not have cancer. There are two possible outcomes that usually can be categorized as yes or no. In this case, we can use binary classifiers like logistic regression. There are other cases where there are multiple outcomes, in which case we need a multiclass classifier. Logistic regression does have multi-class generalizations, but we will not cover them in this module. In classification, we try to predict the binary outcome based on one or more measurable variables. The measurable variables are used to construct a prediction. For example, if we want to predict whether a patient is sick, for example, whether a patient has cancer, one measurable variable could be exposure to radiation or sun. Another measurable variable could be the weight, the third one could be the gender. The classifier uses these measurable variables in order to predict the outcome, therefore we need to have one or more measurable variables. Finally, we need to have a situation where predictions about the nominal or the outcome variable can be made based on the measurable variables. In some cases we can not make such predictions. For example, if the measurable variable is not related to the outcome, the logistic regression model will not be able to pick up the signal.



3. Here are a few examples of binary classification. The first example of ad placement, a website needs to determine which ads to show to which user, and in what order. In order to do that, we need to construct a model for the relevance of an ad to a user. Such relevance models are usually constructed based on the prediction of whether a user will click on an ad or will not click on the specific ad. That decision of user clicking on an ad or not clicking on an ad is the response variable, or the variable we are trying to predict. So, binary variable, click yes or click no. The measurable variables in this case could be things like the type of the ad, the history of the user, and so on. A similar situation to ad placement happens with feed ranking. By feed ranking, I mean the ranking of items in Facebook feed or LinkedIn feed, for example. When we construct the Facebook feed or the LinkedIn feed or any other feed in other websites, we need to again estimate the relevance of feed items to a user. One such way to determine that relevance is to learn a click or no-click model. Or alternatively, a like or not like, or comment, not comment, or reshare, not reshare model, based on the past history of users interacting with the feed. In this case, the response variable could be click or no-click, like or no-like, etc. And the measurement variables could be similar to what I described with the ad placement example. Another example is in recommendation systems, where the system is trying to recommend items to a user. Again, many recommendation systems are based on the construction of the binary classifier, trying to predict whether the user will be interested in the item. We can think of another example in this case, movie ranking, such as in Netflix or Amazon Video. Where an interest or no interest is a binary variable that is modeled by stream or no stream of a particular video. As we see in these three examples, we have classifiers, specifically binary classifiers, under the hood, even though at first glance the three problems do not seem that relevant to classification.

4. In this quiz on binary classification, check those tasks that lend themselves to binary classification. Spam detection and filtering, credit card fraudulent transaction detection, medical testing to determine

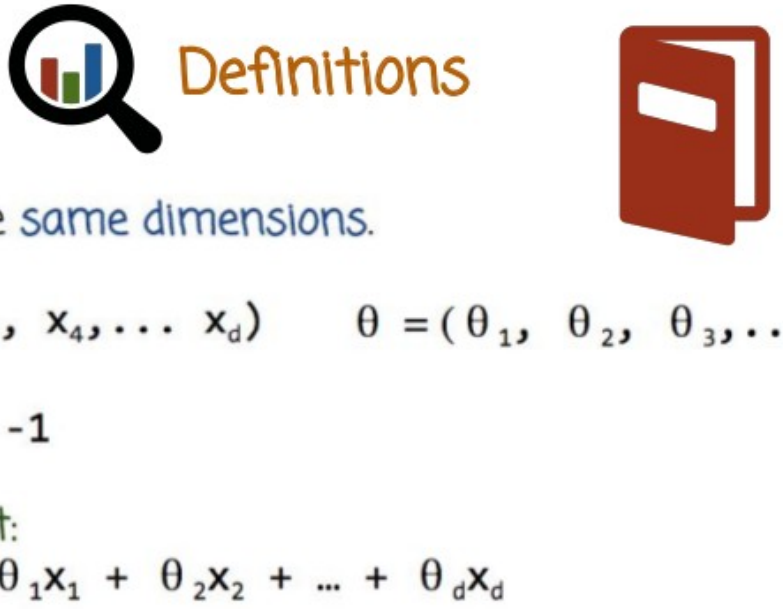
if a patient has a given illness or not.



5. The first case of spam detection and filtering. Certainly classification or binary classification is relevant because we can construct a classifier to predict whether a specific message is a spam message or maybe a specific message for a specific user is a spam message and then filter all the messages that the classifier predicts to be spam within certain confidence. The second example credit card fraudulent transaction detection also is clearly a binary classification problem. In this case we try to predict whether a transaction is fraudulent or not. This is a binary variable that we try to predict based on different measurement variables such as past history of the credit card account holder, type of transaction, etc. In the third case, medical testing to determine if a patient has a given illness or not, we can also relay binary classification by constructing a response variable that is yes or no based on whether a specific illness exists or not for a particular patient.

Many of the concepts discussed in this lesson are used on a dataset created by:
The Institute for Digital Research and Education

R Data Analysis Examples: Logit Regression can be found at: logit
http://www.ats.ucla.edu/stat/r/dae/logit.htm



6. Here are some definitions. We have a vector of measurement variables x. It's a vector of decomponents, x1, x2 through xd. We're going to denote it using the letter x and understand that in general, it can be a vector. The response variable, or the binary variable we're trying to predict, is going to be denoted by y. And it's going to take the values of +1 or -1, +1 to indicate yes, -1 to indicate no. Although it doesn't matter, you can flip it if you want, and that's going to refer to the variable we're trying to predict. In some cases, you may see y = +1 or y = 0. That's another setting, and much of the math that we'll see can be done in either case. It may be a little bit simpler If we use this definition of y being +1 or -1, rather than 1 or 0. So we're going to stick to the +1 or -1 situation because it's going to give us slightly simpler formulas. We going to have another vector called theta. Theta is a vector that describes the classifier. It's a vector of parameters, also known as the parameter vector, and it has also d components, where d is the same dimensionality as the dimensionality of the measurement vector, x. The inner product between two vectors x and theta is defined as the sum of the products of the corresponding components, theta 1x1 + theta 2 times x2 and so on until theta d times xd.

Definitions

Given a vector of features x,
assign a label y of +1 or -1

$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots, (x^{(n)}, y^{(n)})$

- Each $x^{(i)}$ is a vector
- Each $y^{(i)}$ is its +1 or -1 label.

The task of classification then is given a vector of features x that describes the measurements, assign a label y of either +1 or -1. We going to do that based on a training set. A training set consists of pairs of measurement vector and response or label. In this case, we have a training set of n pairs. We have n pairs, and each one of the pairs is a situation described by x and a label described by y. This is usually gathered from historic data. So for example, in the case of spam filtering, x can be a vector of measurements or features describing the messages, and y can be the label. So (x(1), y(1)) would describe one message and whether it's spam or not. (x(2), y(2)) would describe another message, whether it's spam or not. And so on. As we saw earlier, each x is a vector and we denote the specific pair index with a superscript because the subscript is going to refer to the components of the vector. So x superscript i, that's the ith vector in the training set. Each label is +1 or = -1 and that describes the label of the ith vector in the training set

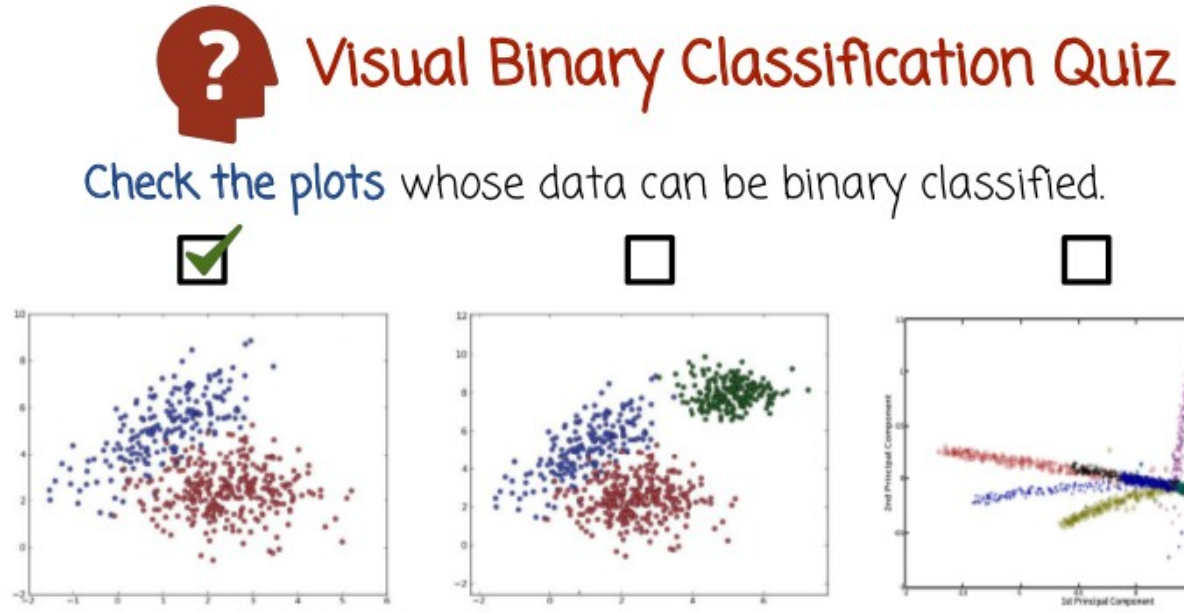
LinkedIn Example

	User Characteristics				
	x_1	x_2	x_3	x_4	y
					click
					no click

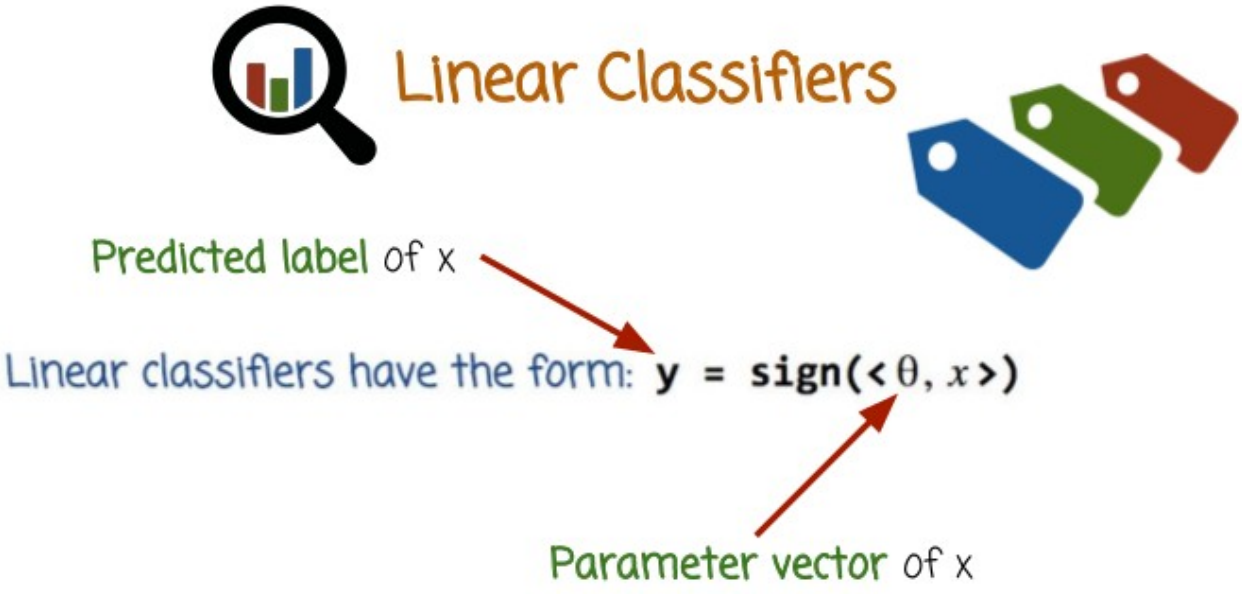
Training data = table (dataframe) of rows representing training set examples and labels

7. Let's see an example of a training set. In this case, we'll consider the problem of feed relevance. Let's say, at LinkedIn. Where the problem is to rank items for a specific user in the newsfeed. The labels, in this case, for example, could be click and no click. Click represents an interest. No click represents no interest. If we're able to predict whether a user will click on a specific item then we can just sort the newsfeed based on that score or probability of click. In this case we have a table. Where each row is a different pair of item and user and the different components of the vector X are measurements or features capturing the item and the user and the relationship between the items and the user. I'll give a few examples, the features could hold the words or the text phrases in the item. They could hold the length or how many words are in the item, they could hold whether there is an image or not. They could hold what is the relationship between the person that posted the item to the feed and the user who views the feed. How many times they have interacted in the past. Each pair of item and user will be a row and the label will be a click or no click based on historic data. This is a general case for classification, for other problems such as spam filtering and so on. We can also come up with such a table in the case of spam filtering for example. These ones may be the words in the message and label Y could be whether it's spam or not. Or if you think about spam being personalized, maybe one person's spam is another person's not spam, in this case, every row will be a combination of a message and a person. So we can think of it as a table, the training set is a table where the ith pair is the ith row in the table, the x vector corresponds with the first d columns and the last column correspond to the label, which is binary + 1 or- 1. If we want to represent this in Python or in R, we can do that using a data frame. But we can also use other formats to represent that information.

8. In this quiz on binary classification, check the plots which data can be binary classified.



9. In the first case we see there are two colors, red and blue. The red could correspond to +1, the blue could correspond to -1, and the coordinates of each point correspond to the two components of x, x1 and x2. So, every point has two components which correspond to its dimensions in the two-dimensional space and a label describing its color. This corresponds to a binary classification dataset. In the second case, we have a similar situation, but we have three colors. This seems to be more appropriate for a multiclass classification, rather than binary classification, where there are three different possible responses. Rather than just red or blue, here we have red, blue or black. So we're not going to checkmark this graph even though one could come up with binary classification situation such as, I want to predict whether a point has the color black or not, or whether a point is red or not. In the third case, we have even more possible outcomes because we have more colors. Again, this seems to be more appropriate for a multi-class classification, so we're not going to checkmark this. However, we could create a binary classification training set and build a binary classifier to answer the question or to try to predict the answer to the question whether a point is red or not. Now in each of these cases when I'm referring to predicting the color or whether a point is red or blue, we can certainly use the classifier to predict the color of a point in the training set. But that's not as interesting, we already know the colors of the points in the training set that corresponds to the labels in the training set. What's more interesting is using the classifier to predict the color of new points that appear later after we finish collecting the training set. This is where the true power of classification in machine learning appears.



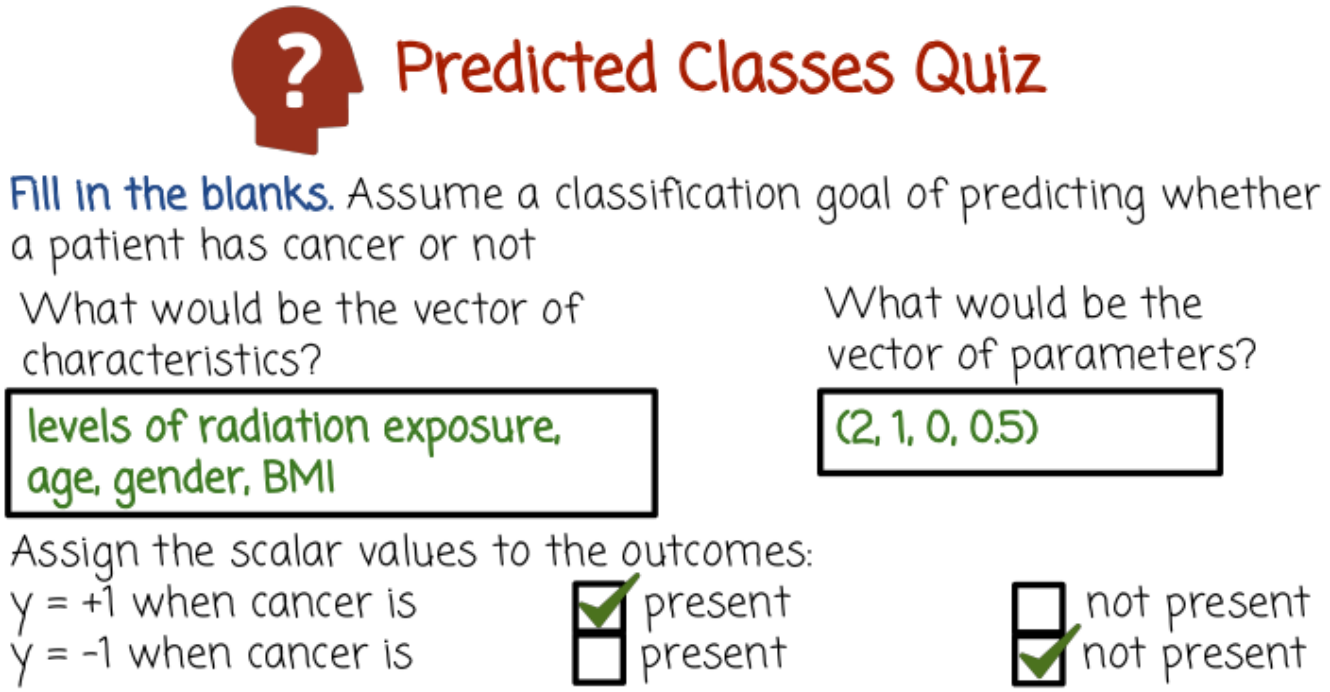
10. Linear classifiers are classifiers whose prediction rule has the following algebraic form. The prediction associated with a vector of measurements x equals the sign of the inner product between the parameter vector theta and the measurement or feature vector x. If the classifier has this prediction rule, it is a linear classifier. The vector theta is a parameter vector describing the classifier. If we change the vector theta, we will change the classifier. And the goal of the training process is to use the training data, the n pairs that we saw before, to come up or to learn a good vector theta that will work well in predicting future outcomes or labels.

Inner product	Predicted Class
$<\theta, x>$ positive	+1
$<\theta, x>$ negative	-1

The sign function works as follows. If its argument is positive, it assigns a label +1. If its argument is negative, it assigns an outcome of -1. If the argument of the sign function is 0, then it can be either +1

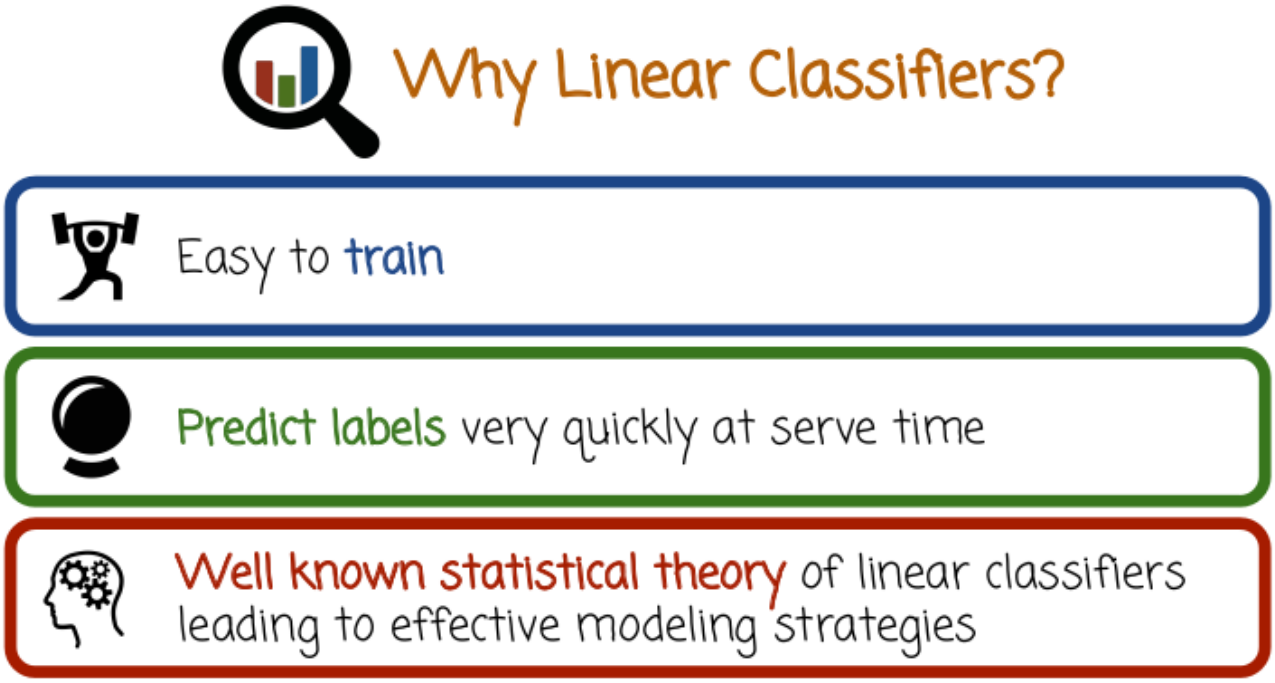
or -1. And it's not fully defined.

11. In this case we're going to assume a classification goal of predicting whether a patient has a cancer or not. We're going to enter here the vector or measurements, or features, or characteristics to vector that before we referred to as x. In this space, we're going to enter the vector of parameters theta, that define the classifier. In here, we're going to determine what corresponds to the value of the predicted variable y +1 or -1, whether the presence or absence of cancer corresponds to +1 or -1.



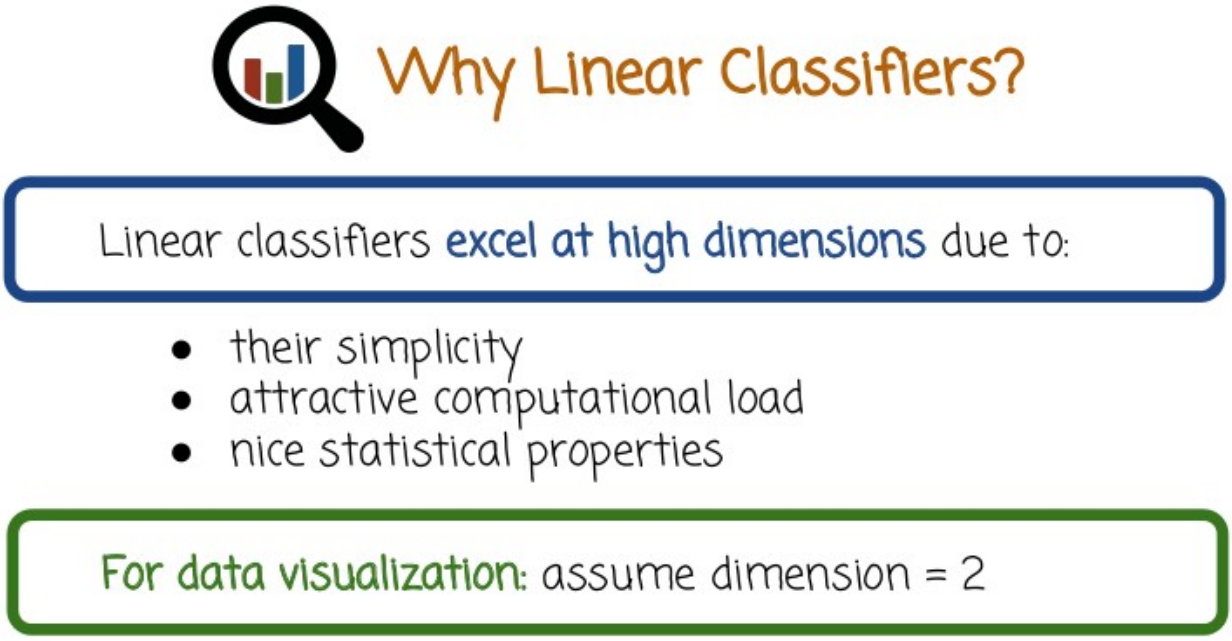
12. But here's a possible vector of measurements, this is just one such possible vector, there could be many other choices. This is a vector of four variables, level of radiation exposure would be x1, age would be x2, gender would be x3. For example, 0 corresponding to male and 1 corresponding to female. And BMI, or Body Mass Index, corresponding to x4. So we have four numbers corresponding to the components of the vector x. This is the vector that captures the measurements of the patient. In this space we need to enter a vector of four parameters. That vector needs to have the same dimensionality as the vector of measurement. Since here we have four numbers, we need to have four parameters here. And the prediction will be the sign of the inner product between the measurements and the parameters. In this case, the first measurement, level of radiation exposure gets a weight of 2. So 2 will multiply the radiation exposure. 1 will multiply the age, 0 will multiply the gender, and 0.5 will multiply BMI. In this case, this specific classifier would basically assume that gender is not a good signal in determining cancer, because it has weight 0. When we add all the components for the inner product, gender would be absent because it's multiplied by zero. BMI, age and radiation exposure are all important and their relative importance can be seen by the weights. In this case radiation exposure is the most important, age is second most important, and BMI is the third. Now I'm not a doctor or a cancer expert and this may be completely unrealistic or a very bad classifier. But this is just one classifier out of many, this is not being learned based on It's just one particular example that may have some merit but maybe that in many ways. So I'm not recommending to use these specific classifier, this is just an example. We're going to assign a +1 y when cancer is present, and a label y -1 when cancer is not present. One thing I want to show here is that because the weight here is positive

and + 1 corresponds to present, and remember that the prediction is the sign of the inner product. We can learn from this specific classifier that this specific classifier assumes that an increase in radiation exposure, and an increase in age, and an increase in BMI would increase the chance that our prediction would be positive, or yes cancer. If the weights here would have been -1 or more, that would mean that an increase in the measurement would cause a decrease in the chance of predicting cancer, or one.

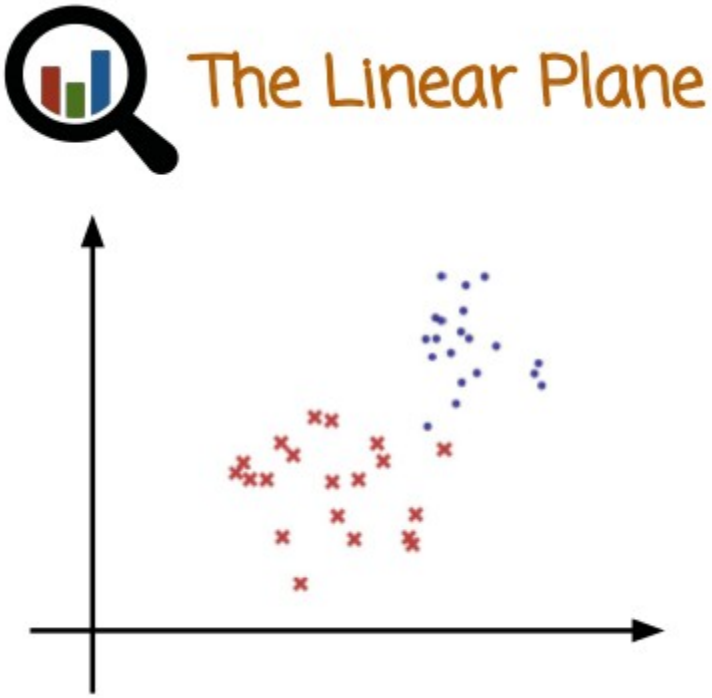


13. Let's see what motivates the use of linear classifiers. Like I said earlier, linear classifiers are just one type of classifiers that I define by saying they have the specific algebraic form that was introduced earlier, specifically that the prediction is the sign of the inner product between the measurement vector and the parameter vector of the classifier. The first reason is that they're easy to train. Of course, that depends on the details of the classifier. But many linear classifiers are very popular and the training process can be relatively easy. Besides the fact that the training process is easy or fast, another advantage is that the prediction can be very quick. The prediction is based on the very simple formula, the sign of an inner product that can be computed very fast. And specifically, it can be both parallelized if the dimensionality is very high, by assigning different processors to hold and to compute subparts of the inner product corresponding to different parts of the vector. And in addition to being parallelized, they could be computed especially fast if the parameter vector or if the measurement vector is mostly zeros. This case is called sparse case. The vector that is mostly zeros is called the sparse vector. And the inner product between a sparse vector and a non-sparse vector or between two sparse vectors can be done very fast, if we know how to keep track of where are the zeros and where are the non-zeros. In many cases in industry, the prediction time is much more important than the training time so that the second advantage here is crucial. Think, for example, about feed ordering or ranking when you construct a news feed or when you want to place ads. You need to make a decision at predicting relevance of items or ads and order them within a fraction of a second. And many, many items need to be scored within a fraction of a second and then ranked. And so, if we have nonlinear classifier with the complex prediction rule that takes a lot of time, that will make that classifier inappropriate to be used in many important cases in industry. The third reason is that linear classifiers have a very well-known and understood statistical theory. That theory helps us understand which classifiers makes sense to use and which classifiers we should not use. The theory has been developed over several decades, and it's much harder to develop a corresponding theory for nonlinear classifiers.

If we do not have a theory that tells us which classifier would work well or when we should use different classifiers, it becomes much harder to explore the different space of all possible classifiers and try to experiment with different classifiers will take a lot of time. The fact that we have very well-understood theory would narrow down that choice and guide us in trying to figure out which method, which training method, which prediction method, which families of classifiers, of linear classifiers we should focus on.



Linear classifiers are useful in general, but they are particularly useful at high dimensions. When I say high dimensions here, I mean when the number D is very large. If you remember, D is the number of measurement variables or the dimensionality of the vector x is also the dimensionality of the parameter vector theta. The simple form of the linear classifier scales up very well to high dimensions. As a result, computational load is very light. And even if the dimensions D is very high, in an industry, the dimension D could go up to millions or hundreds of millions, and in some cases billions, billions of features. And linear product or linear classifier would still be able to handle such high dimensionality, assuming we have sparse feature vectors or sparse parameter vectors. And again, the statistical properties become even more important at high dimensions. In this case, it's harder to get the classifiers to work well. And we need more theory and a better understanding of how to use these classifiers, how to train them, and how to predict with them. Despite the fact that we should usually keep in our minds classification in high dimensions when it comes to visualization of data and when it comes to understanding intuitively how the classifiers work, we're probably going to assume the dimensionality of 2. You're going to see in this module and in other cases, graphs of two-dimensional classification. This is just an illustration. In reality, in many cases, the classification happen in much higher dimensionality than 2.

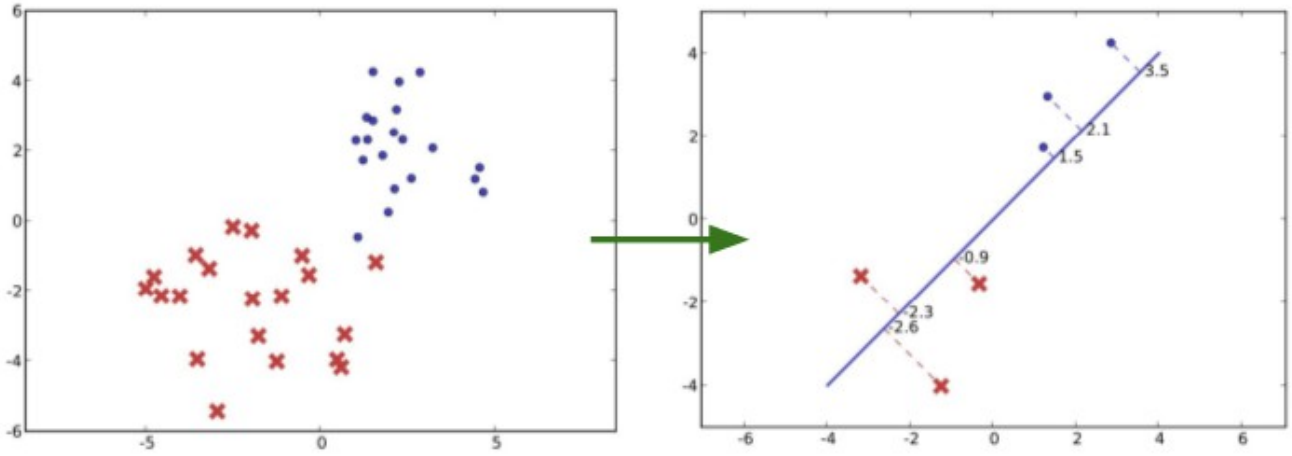


14. Another reason why linear classifiers are called linear is that the decision boundary, which is the area that distinguishes between what is predicted as positive, and what is predicted as negative takes on a flat shape. In the case of two dimensions it would be a line. In the case of 3D dimensions, it would be a two dimensional plane in a three dimensional space. In the general case of d dimensions, it would be a d minus one dimensional hyper plane in a d dimensional space. Like I said the earlier, we're going to usually visualize or illustrate things using two dimensions.

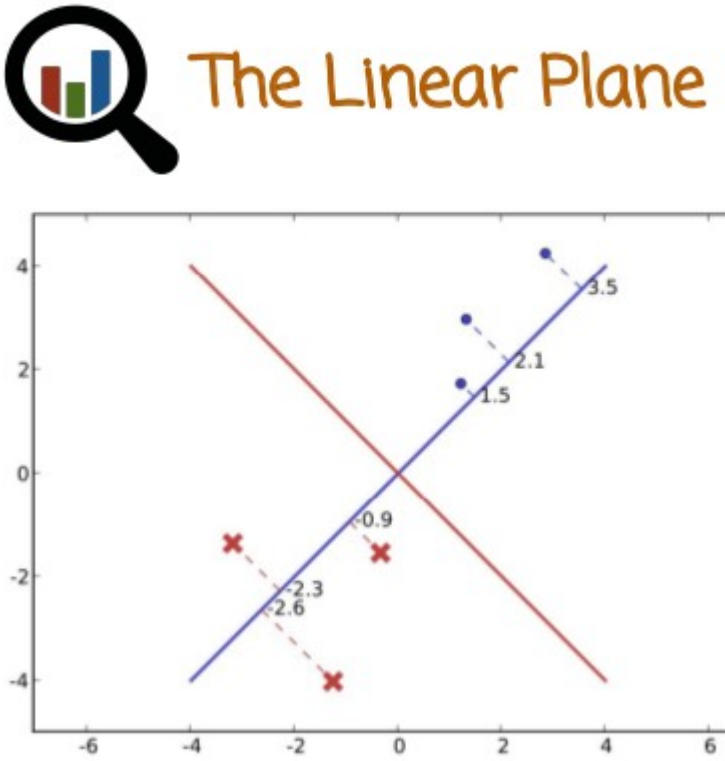
- Given $x = (x_1, x_2)$
- The classification is: $\text{sign}(ax_1 + bx_2 + c)$

Sign($ax_1 + bx_2 + c$)	Classification is
positive	+1
negative	-1

Let's assume we have data that we want to classify. In this case, we going to illustrate the label y using the shape, whether it's a circle or a cross. And so the classification problem is given a vector in this case of two dimensions X1 and X2. We want to predict the label so the linear classify would predict the label to be the sign(ax1 + bx2 + c). Where a corresponds to theta 1, b corresponds to theta 2, and c here, is called an offset term, we haven't seen that before. But we'll see soon that we can pack all three parameters, a, b, and c into a vector theta of three parameters. Theta 1, theta 2, theta 3, and all the formulas that we've seen before still apply because we can pad the vector x with the number 1, and then the inner product between the three dimensional vector theta and the three dimensional vector x, where the third component of x would be one, would be theta one x1 + theta two x2 + theta 3. In this case we just call theta one, theta two, theta three, we call them A, B and C. The prediction is +1 if the sign is positive and -1 if the sign of the inner product is negative. So, lets see how the linear classify works.

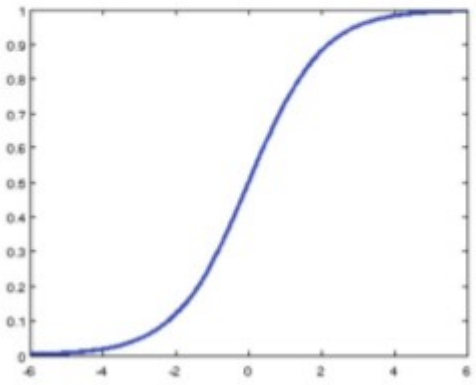


We have these two clouds of points in the two dimensional space and when we take an inner product between a vector of measurements and another vector, the other vector is normalized meaning it has unique length. The outcome of the unit product is basically the projection of the input vector into that unit vector. The unit vector is the theta vector that corresponds or describes that projection direction. The corresponding inner product would capture the distance of the point from this specific hyper plane which will correspond to the decision boundary.

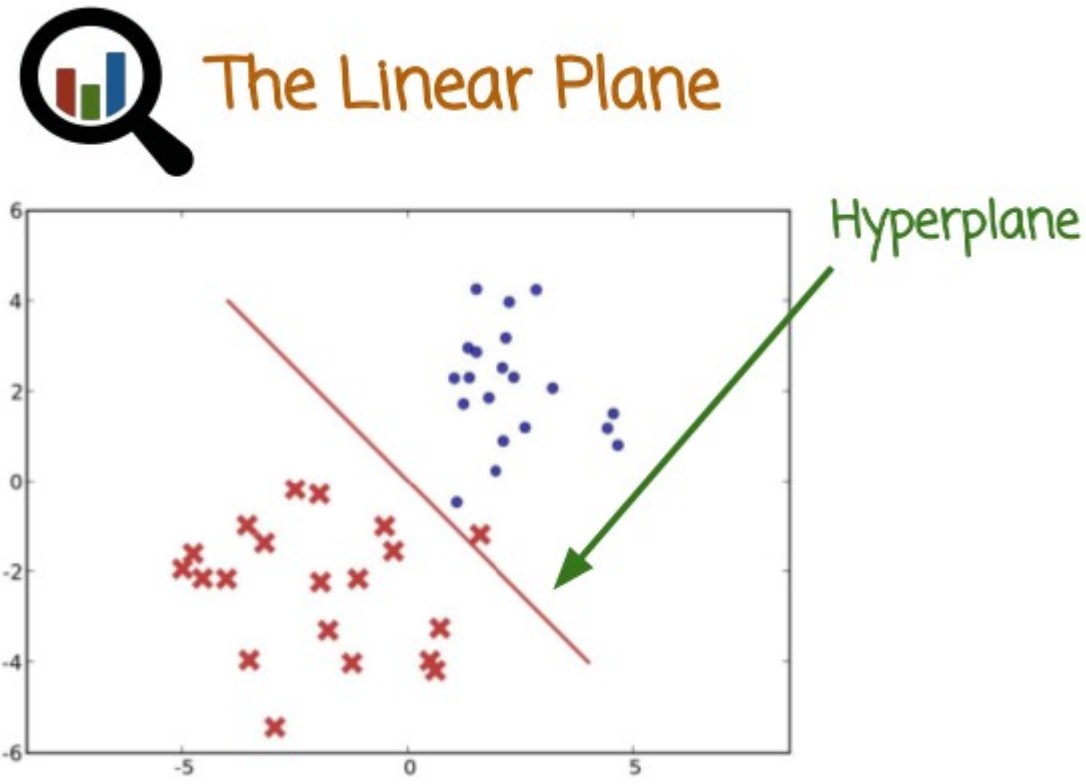


記住(別證): θ 就是那條藍線. <θ, x> = 0 求出的 x 就是那條紅線, 即分界線.

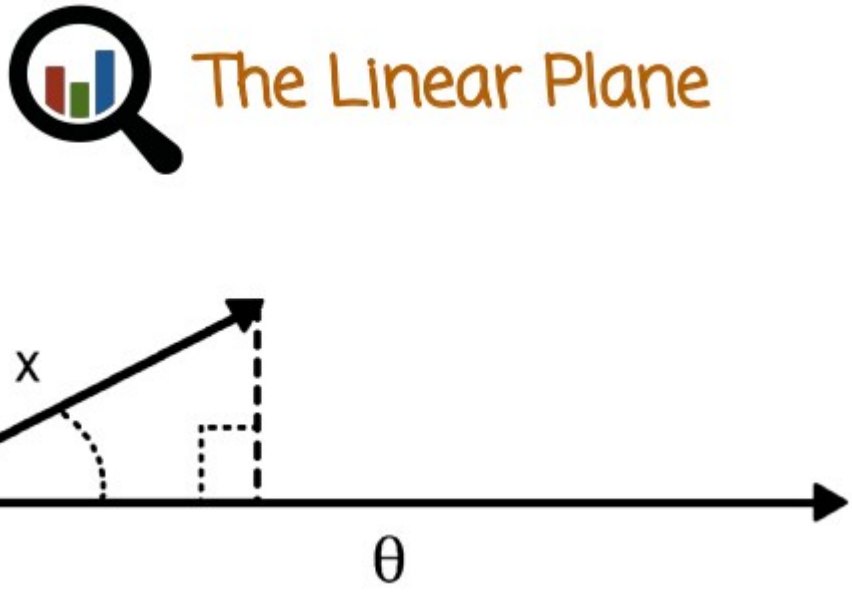
In this case, let's assume the decision boundary is the red line, the vector theta would be a vector that is perpendicular to the decision boundary, that is here described using the blue line. And the projection of each one of these points on a blue vector, which is a vector perpendicular to the red decision boundary, would correspond to the distance of that point from the decision boundary. We see that the three points here at the top get positive inner product, and these three points get negative inner product, and if a point was right in the decision boundary it would get an inner product of zero. That helps explain why a linear classifier actually provides in this case a line as a decision boundary, the red line that is perpendicular to the vector theta.



15. The prediction is just the first part of the classifier. In many cases we want to also predict some confidence or probability in our prediction rule. So we need a way to map that particular inner product into a confidence of probability. One such mapping between the inner product and a probability is the sigmoid function, which is plotted right here and we'll see that soon. This is one way of getting a probabilistic classifier that doesn't just predict whether a label is plus one or minus one. It also predicts a probability associated with that fact. As we saw before, when we talked about arranging ads or ranking feeds, it's also important that we know that probability of a user clicking on an item or in an ad. Because we can use these probabilities to rank the items in the feed or to order the ads. And so it's not just important that we know whether we think a click will happen or not. Rather, we want to know what's the probability that the user will click on an ad or not.



So let's review again the linear plane in two dimensions. The red line is the hyperplane, in this case, decision boundary that everything to the left of it or underneath it is going to be predicted red. Everything on top of it or to the right of it will be predicted blue, or plus one and minus one. It's called hyperplane rather than a line because it will be a line in two dimensions. But it will be a 2D plane in three dimensions, a 3D planes in four dimensions, and so on. So more generally, we refer to it as a hyperplane. That decision boundary describes the classifier.



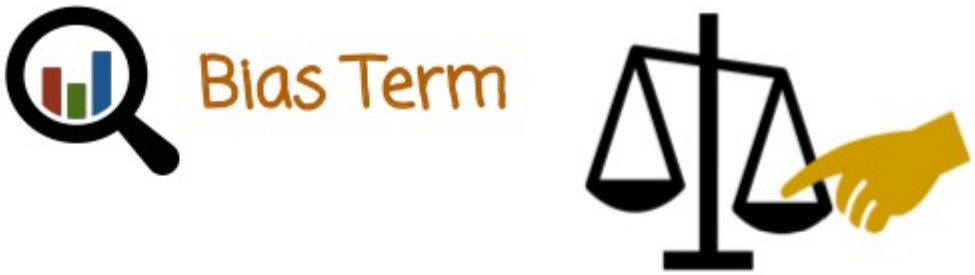
We have a vector capturing the measurements or the features that we're trying to use in our classification. Then we have theta, which is a parameter vector describing the classifier. That vector describing the classifier, theta, is going to be perpendicular to the decision boundary. Based on linear algebra, we know that the inner product of x and theta, if theta is normalized to be of length one, is going to be the projection of x into theta. The reason it doesn't matter what is the size of theta, because I said it has to be normalized to be of length one. But before that, when we saw in the cancer quiz, I gave as an example a vector theta that was not normalized to be of length one. Meaning to have L2 norm of one, or theta one plus theta two plus theta three square root equals one. We don't need it to be of length one is because the classification decision doesn't matter whether you renormalize theta or not. But if you want to see the geometric interpretation, it's easier to see that if you assume theta is normalized. But the decision boundary will not change whether you multiply theta by constant positive scalar or not. You'll get the same decision boundary and the same prediction rule.

16. In this quiz we will answer the question, should the decision boundary pass through the origin? Origin being the point where all the components of x, or the vector being classified, is zero.

Should the decision boundary pass through the origin?

Not necessarily. If we do not require that (by assigning one features to the value 1) the classifier becomes considerably more powerful.

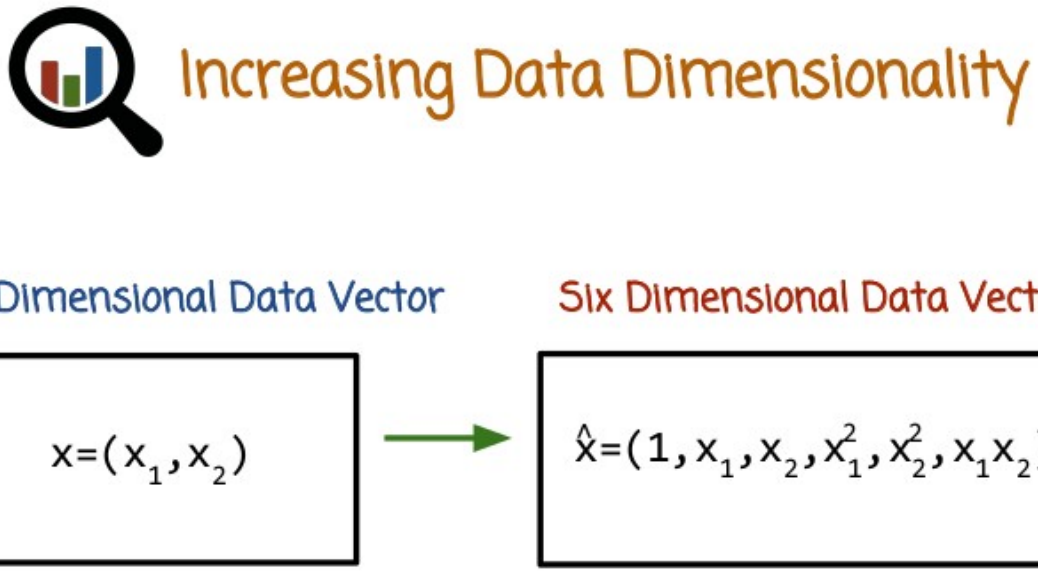
17. The answer is not necessarily. If we do not require that for example, by assigning one feature to the value 1, which becomes an inner product plus an offset. The classifier becomes considerably more powerful. We can learn much more general decision boundaries. They don't have to pass through the origin.



$$x_1\theta_1 + \ldots + x_d\theta_d + c = \langle x, \theta \rangle + c$$

$$\langle x, \theta \rangle$$

18. Let's see again in more detail, how do we insert or remove the Bias Term that we referred to earlier. The inner product is the first part here of the algebraic expression, and then we have the offset or bias term c. Having that bias term is very useful, because, without it, our decision boundary has to pass through the origin that restricts our classifier in a very significant way. We also add C to the classification rule that releases us from that constrained decision boundaries, which is allowed to not pass through the origin which makes it much more flexible. Here we write it down in a shorter format. The end product of X and data plus the bias term. And now, here is the trick. If we make theta, inside of d dimensions. D + 1 dimensions. And we define the d + 1 component of theta to be c. And then we define x to be a d + 1 dimensional vector. With a d + 1 component being always 1. Then the inner vector of the new vector x, which is d + 1 dimensional. And the new vector theta which is d + 1 dimensional, will be exactly the inner product of the old d dimensional x and d dimensional vector theta plus the off set c. This shows, that we can still use this equation when we discuss linear classifier in a product between an input vector and a parameter vector, and if we define the last feature or we pad x with one more feature that is always 1. The last component of theta will be automatically the biased term c and we get that more flexible formula right here. The decision boundary does not have to go through the origin but we can keep our simpler form of just x in a product with theta. Later on, that will be useful because we're going to have to do some math with this inner product in carrying the bias term in all our formulas will be quite a bit more complicated.



19. The biggest drawback of linear classifiers is the fact that the decision boundary or the area that separates the positive predictions from the negative prediction is a line in two dimensions or a hyperplane in higher dimensions. That's a fairly strong restriction, quite possibly the data does not behave that way. Meaning the cloud of points describing positive and the cloud of points described as negative is not nicely separated by a flat linear hyperplane. But fortunately there is a very easy and simple trick that we can apply to linear classifiers. That would get rid of that difficulty, and yet will let us keep the nice simplicity of linear classifier. We can keep the series of linear classifiers, the computational scalability, and all the other nice properties while getting rid of that very big constraint or apparent weakness. The decision boundary must be linear or must be a hyper plane. Let's see how that works, let's assume we have a two dimensional data vector that we want to classify, the components being x1 and x2. What we could do is convert the two-dimensional vector (x1, x2) in to for example a six-dimensional data vector. The first component will be 1, second one x 1, third one x 2, the next one x 1 squared, x 2 squared, and the last one is x 1 times x 2. So we're going to transform the original vector x into a new vector x hat of six dimensions. We're going to do that for all the input vectors, all the training set feature vectors or measurement vectors as well as for the vectors that we operate on in prediction time. And the thing to understand is that in this six dimensional space of the new vector, the decision boundary will be hyperplane, but that decision boundary could be highly non-linear in the original space of the features. It could be curved, it could be circular, it could be flat. It would be much more flexible. That mechanism allows us to capture non-linear trends when we use linear classifiers. The fact that we increased the number of dimensions is worth noting (不是 nothing). Now we have more computations to work with and we have to use more storage to store these features. But because linear inner product, linear classifier scales up relatively well with the dimensionality. Such an increase is not necessarily something that we cannot work with. Remember before when I said that in many cases in industry, we classify vectors whose dimensionality are hundreds of thousands, millions, tens of millions, hundreds of millions, and sometimes even billions. Assuming many of the feature vector values or the parameter vectors are zero, that's certainly doable, and that can be doable very, very fast. So it's not necessarily an impossible or a bad route to increase the dimensionality by adding non-linear transformations of the original measurements. We'll get then a linear prediction rule and a linear hyperplane boundary in the transformed high-dimensional space, that will be non-linear in the original space. Of course there's one more thing worth mentioning, that as we do this trick more and more, we increase the dimensionality of the data vector. We also increase the dimensionality of

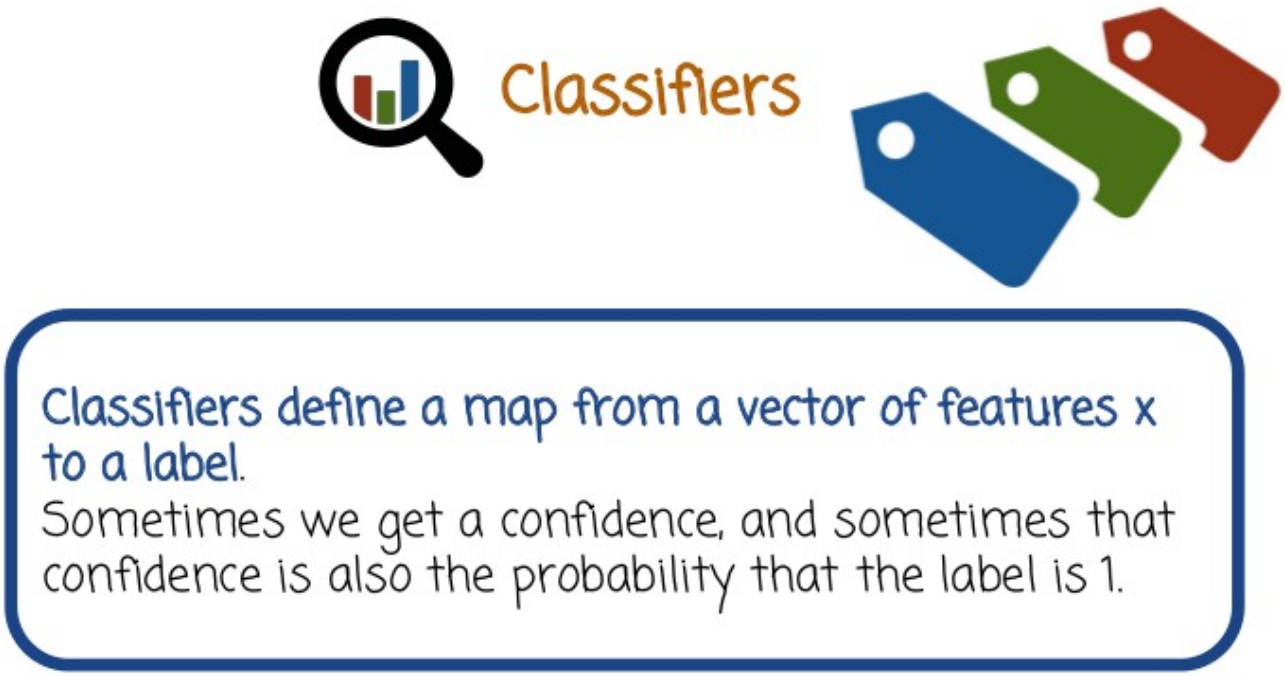
theta. And one difficulty there besides the computational difficulty that I mentioned earlier, Is the fact that this may over fit if we don't have enough data to learn a lot of parameters. And we'll talk about this later on.

Increasing Data Dimensionality

Transformed vector	$(\hat{x}^{(1)}, y^{(1)}), \ldots, (\hat{x}^{(d)}, y^{(n)})$	**Classifier is linear** in the coordinate system of $\hat{x}$
Original data	$(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$	But **classifier is non-linear** in the original coordinate system of x

MO: 上表第一行中間那個 x^ (d)應該是 x^ (n).

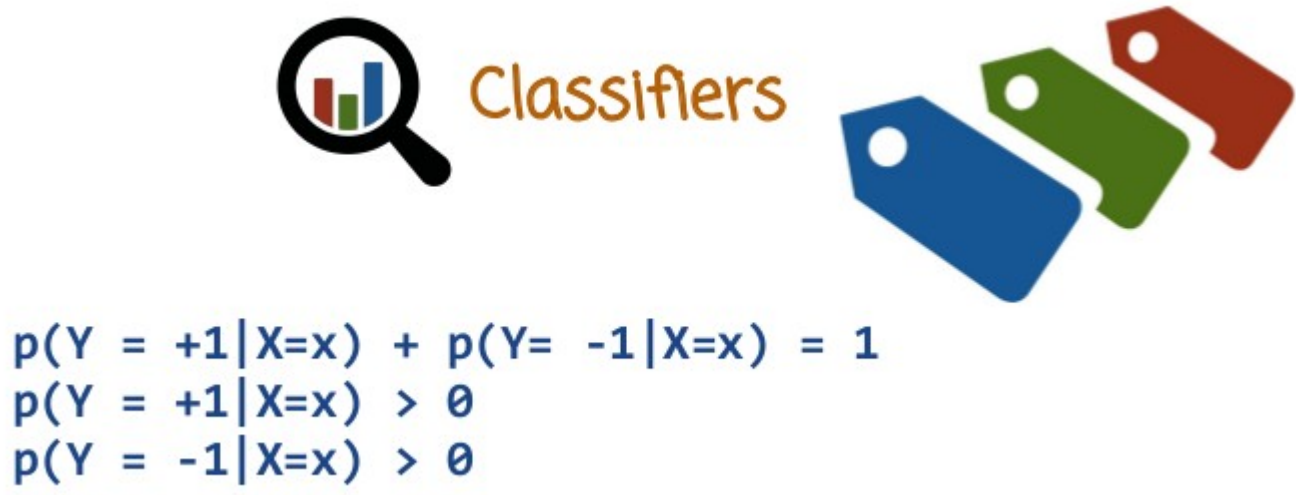
And here we see a summary of what we discussed earlier. The original training data is a vector of heirs of feature vectors or measurement vectors and labels We have n of them. And what we do is we transform each one of the x vectors into x hat and we use this as the training set to the linear classifier. We learn a linear classifier in the new higher dimensional space. When we get a vector we want to classify, we just supply the transformation and then operate the classifier that we learned in the top row.



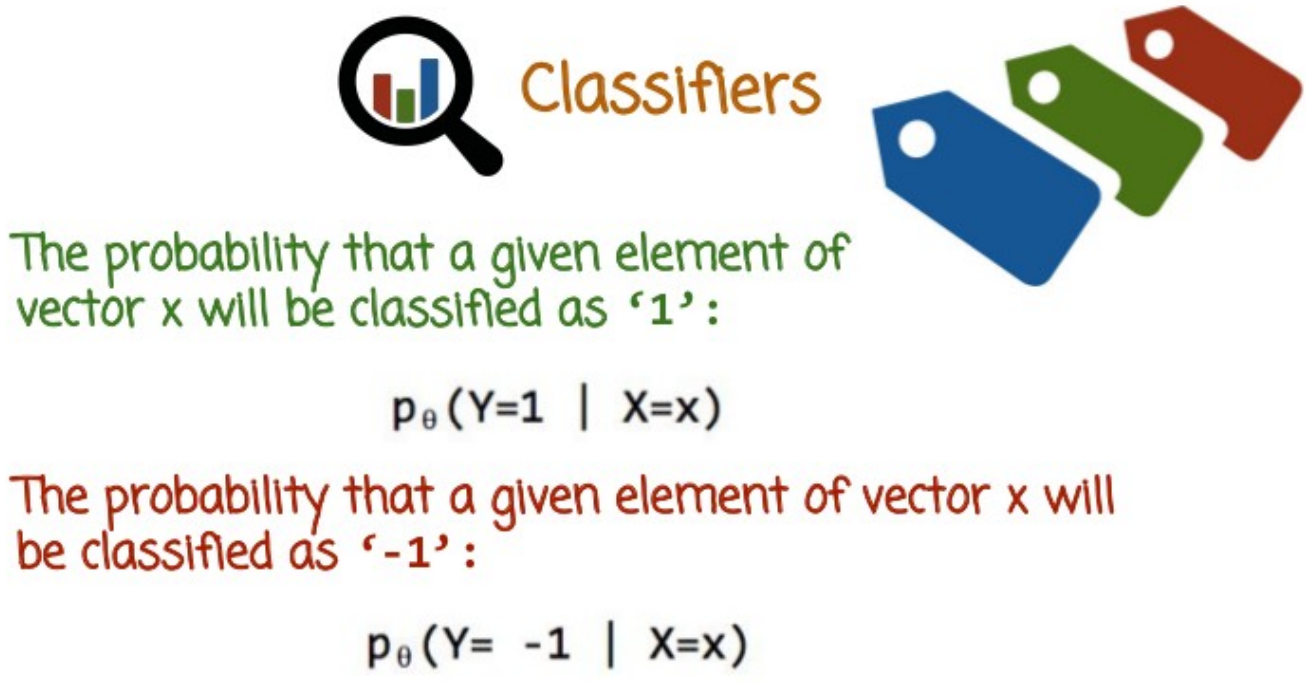
20. But we're going to move now to discuss confidence of probability of a prediction. Remember what I said earlier that confidence of probability is very important because we don't just want a prediction, we want the confidence or the strength of that prediction, so that we can order possible alternatives. For example, ads or other alternatives to the user, rather than just guess that there's 100 relevant ads but we don't know which one to prioritize over the others.

Probabilistic classifiers provide a tool by defining the probabilities of the labels +1 and -1 given the

feature vector x.



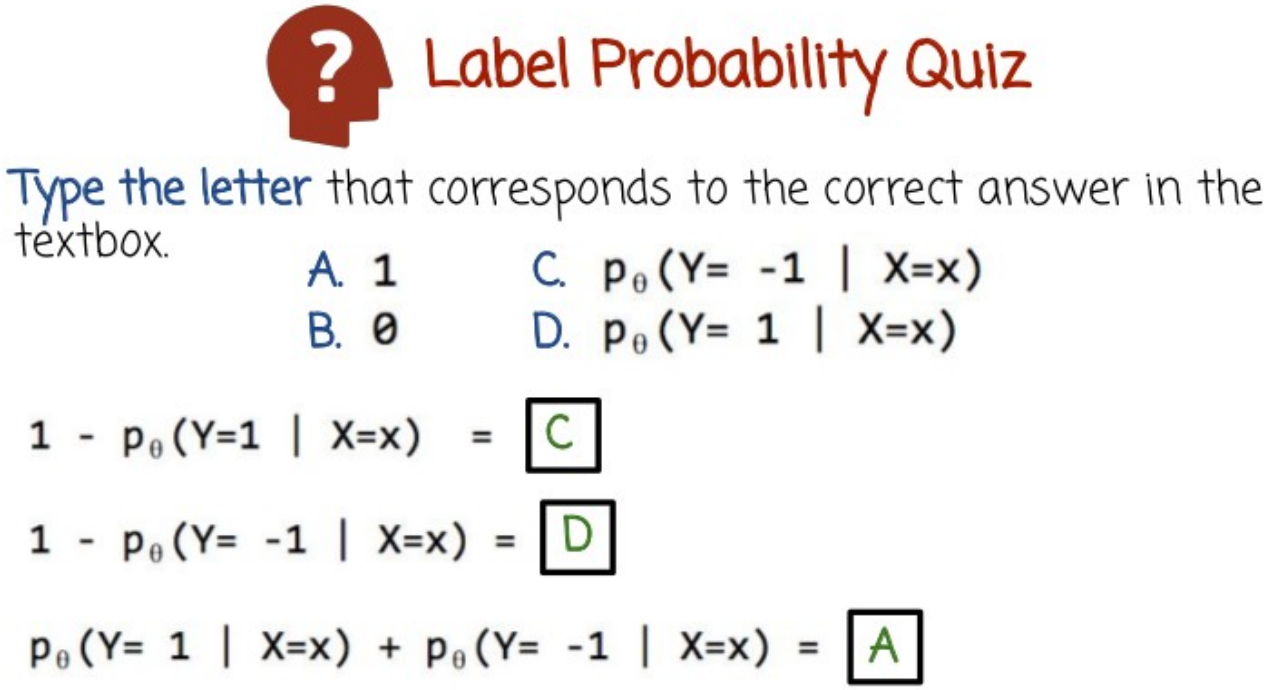
We know from probability theory that a probability of y being +1 given x plus probability of y being -1 given x is 1. Therefore, the probabilistic classifier should just give us a way of either measuring or expressing the first value or the second value. The other value we can get from the first, by doing 1- the first probability based on this formula right here. We also know that probabilities are greater or equal to 0.



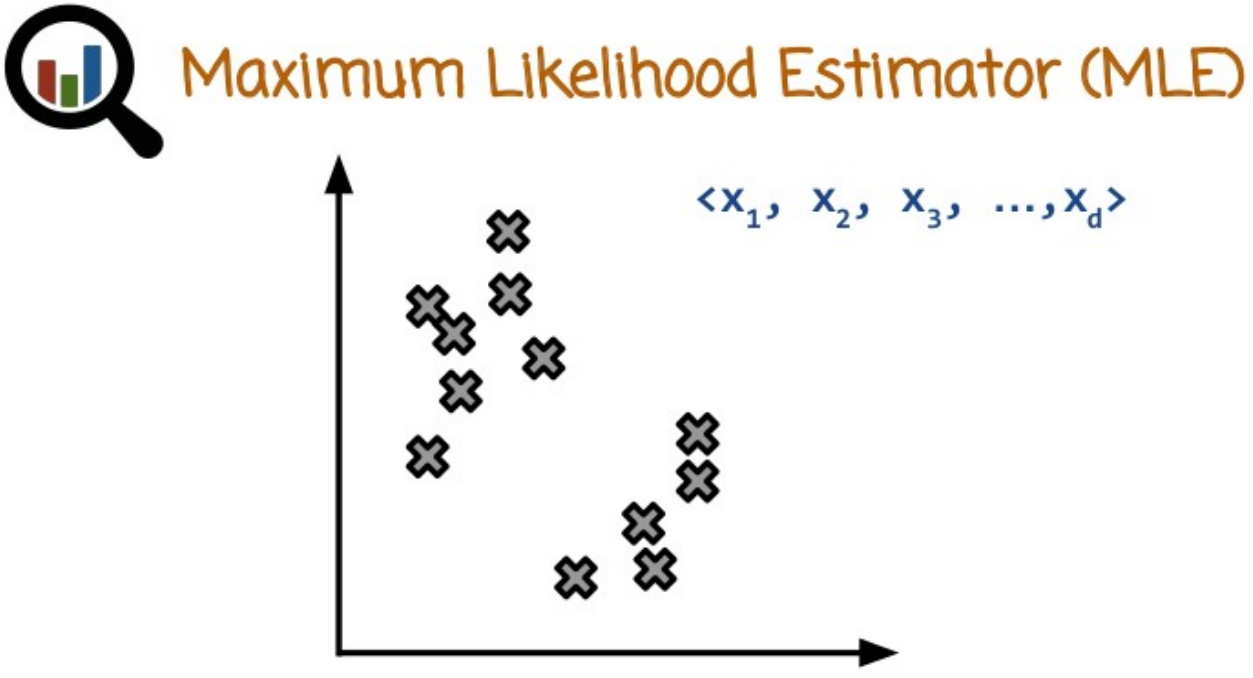
So the probability that a given vector X will be classified as 1, is this probability right here. And we denote using a subscript theta here, the fact that this model p is governed by the parameter of X for theta, that defines the classifier. We'll soon see how that definition happens, but for now we just going to write a little theta under the p. To show that the classifier is going to be responsible for the prediction rule, as well as the probability that the label is one. And based on that variable, we can also

come up with the probability that the label is -1, by just doing 1- this probability.

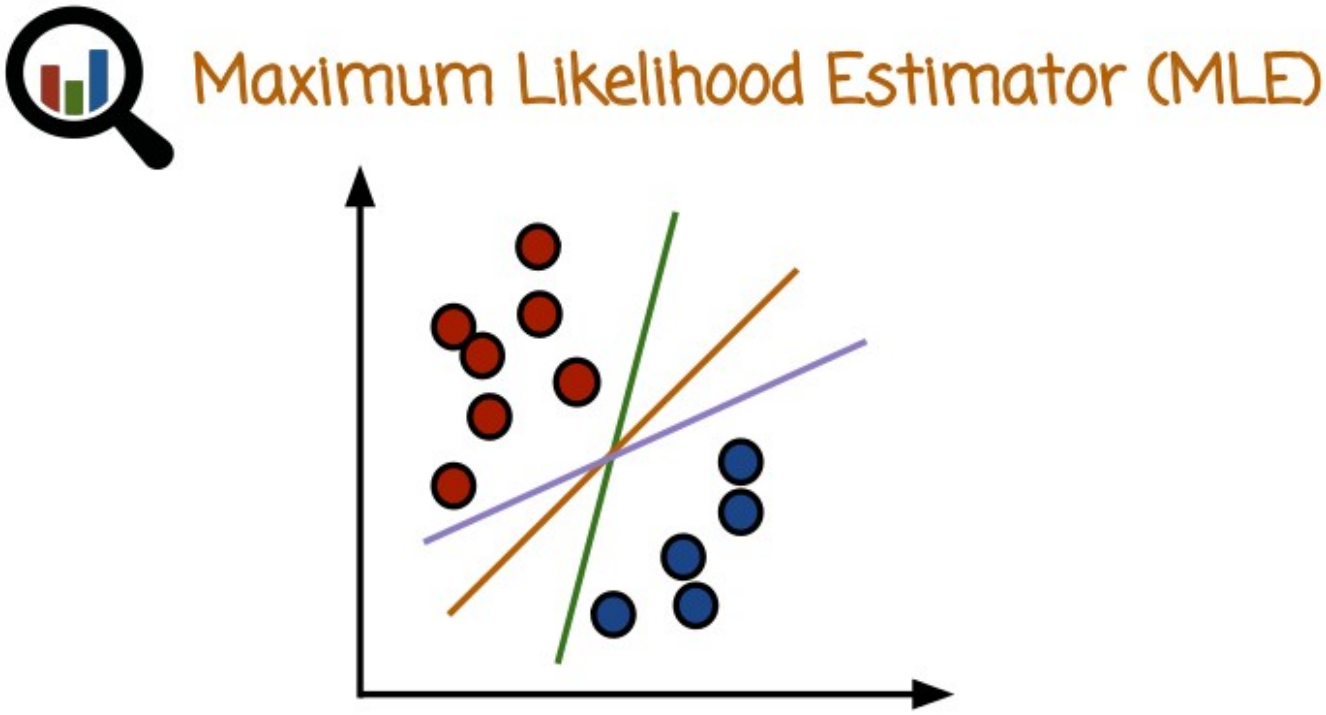
21. In this quiz, fill in the missing blanks with the letters a, b, c, or d



22. 1 minus the probability that y equals 1 given x equals x, equals c. And that's the probability that y equals minus 1 given x. Similarly, the probability that 1 minus p of y equals minus 1 given x is D. That's the probability that y equals 1. The sum of the two probabilities that y equals 1 plus y equals minus 1 has to be 1.

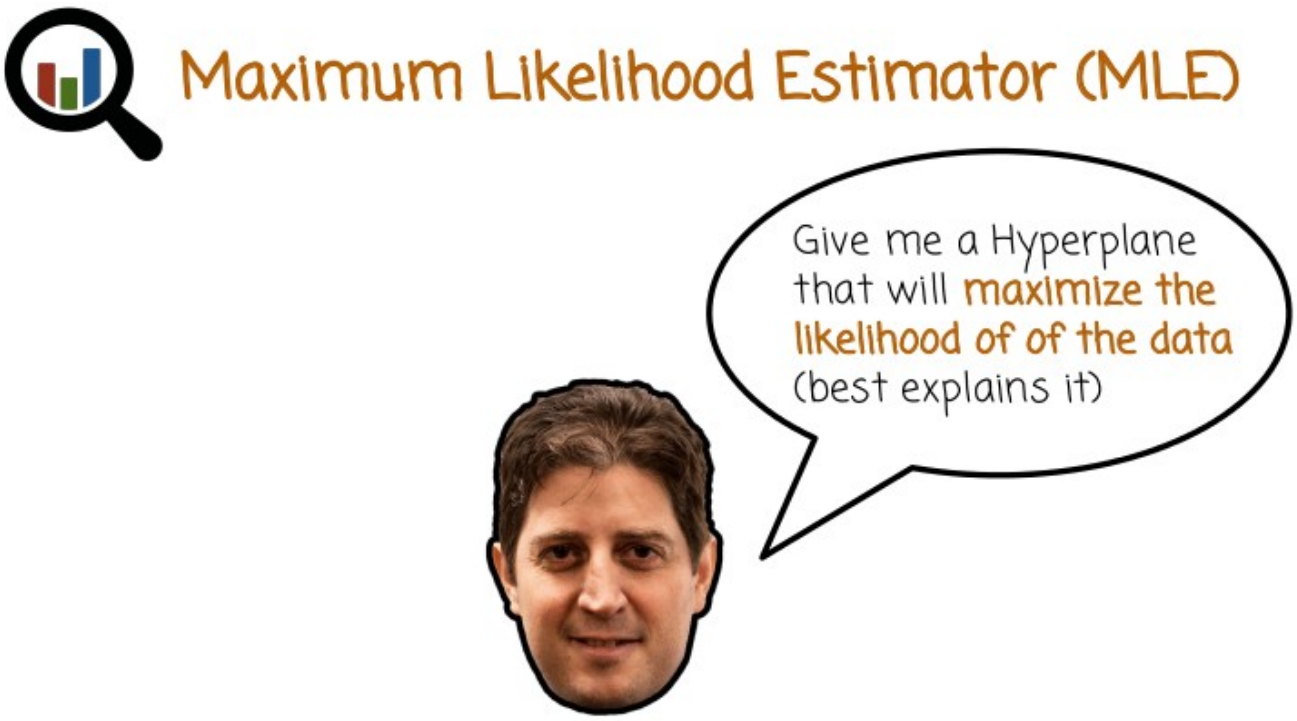


23. We're going to next learn how to get the probabilistic classifier that is known as the maximum likelihood estimator. This is a very well known method of training probabilistic classifiers. It goes many years into the past and it's still in use a lot even today in many places in industry. So as before we start with vectors that describe points in euclidean space. In this case for visualization purposes we assume the space is two dimensional.

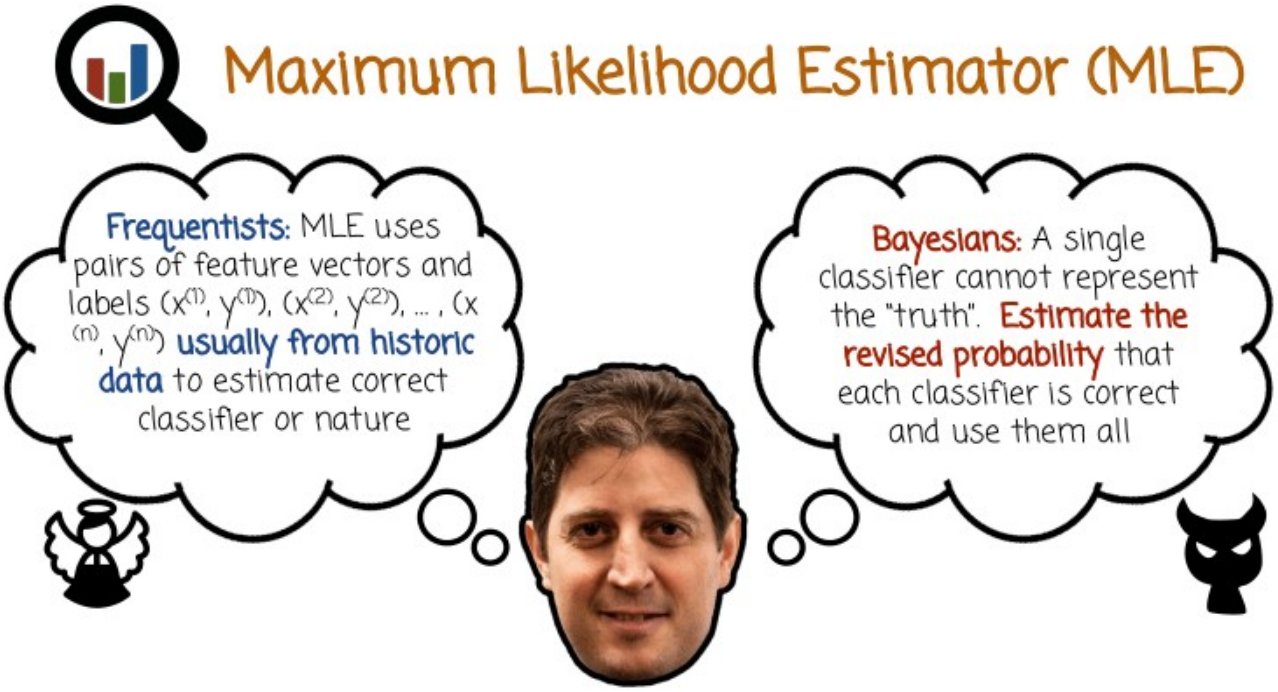


我的理解: logistic regression ⊂ maximum likelihood estimator ⊂ frequentists philosophy
其中⊂表示屬於的意思

We have a training set which we can visualize here using two clouds of points, one plus one, one minus one or red and blue. This can be one classifier, this can be another classifier, this can be third classifier. Each of the the classifier has a decision boundary, it also gives up probabilities, that the classification or the prediction is 1 or -1. It is also worth saying maximum likelihood to assume specific mapping from theta to the probability that y equals 1 or -1. We're going to use logistic regression as that mapping, although maximum likelihood is much more general and can apply to other mappings from theta to probabilities that are different from logistic regression. But still for concreteness we have to focus on one specific probabilistic classifier and so we'll choose logistic regression because it's a very general purpose and popular classifier.



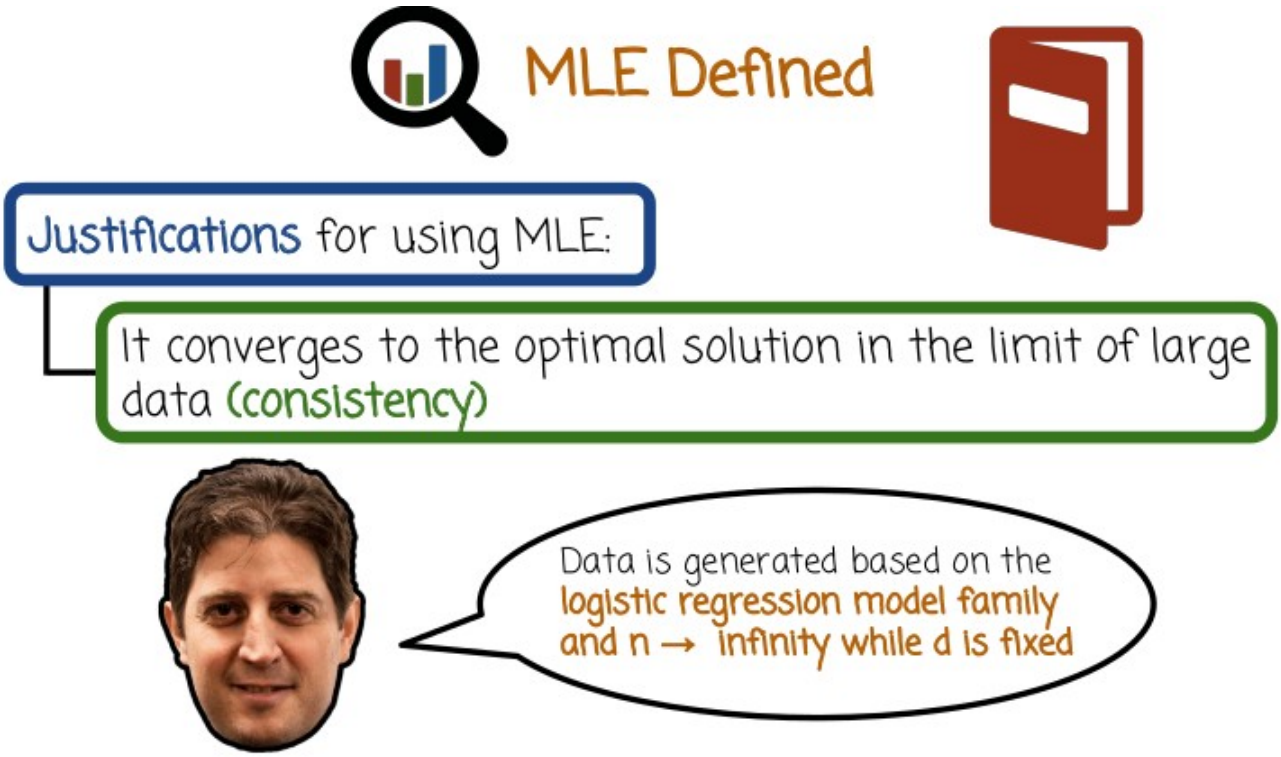
24. So the thinking behind the maximum likelihood estimator is, give me a hyperplane that will maximize the likelihood of the data. Or in other words, give me a hyperplane that best explain the data that I have. The data being the training data that I'm using to train the classifier. If we have a hyperplane that explains the data really well or maximize the likelihood of the data, then that's probably going to be a good classifier for future data.



Maximum Likelihood Estimator intuition that I've described in the previous slide is an example of Frequentists philosophy. Frequentists philosophy is a philosophy in statistics that talks about nature being something that we're trying to predict. There's one correct state of nature that we don't know. And the task of the statistical procedure is to predict that state of nature, that unknown state of nature using observations from nature. So, nature perhaps in the case of classification would be a specific model that generates the data. We don't know that model, but we see the data and we want to use that data to try to guess what is the process that generates the data. And there is one such correct process. And if we have more and more and more data, hopefully our estimator or our prediction of nature will converge to the true process. There's also an alternative philosophy in statistics called Bayesian, Bayesian statistics. Bayesian statistics, the assumption is that there is no single correct state of nature that generates the data. And therefore there's no point in thinking about a specific model that generates data that we're going to try to predict. We're going to try to predict what it's the correct model, there is no point of thinking about that, because there is actually a collection of different models each model being correct with some probability and what we try to understand or to measure is the probabilities that different models are correct and then we can use **all** the models together to inform our decision making. These are two philosophies. Earlier on in the much of the 20th century, frequently statistics has dominated, both in traditional statistics and more recently Bayesian trend has been gaining more popularity, in machine learning in particular. However, when you think of which procedures are being used in industry and in large dataset high dimensions, the procedures that are currently being used successfully are still predominately frequentist tools. Maximum likelihood estimator is one of the most widely used frequentist tools. You can still use the Bayesian, but then you will probably claim this is an approximation to what you really want to do. But you cannot do what you really want to do because of lack of computational resources, or lack of testing time. Therefore, you're going to resort to the frequentist maximum likelihood estimator as an approximation. If you're If you're Frequentists, you're not going to to worry about that and just use the Maximum Likelihood Estimator happily. One more word about this division between Frequentists and Bayesians. Frequentists is not necessary being obligated to use Maximum Likelihood Estimator and only Maximum Likelihood Estimator. There are many Frequentists paradigms that go beyond the MLE. This is something that is often misunderstood in that attacks against the MLE, and we see later in the course high dimensional situations sometimes cause problems for the MLE. So attacks on the MLE are sometimes considered as attacks on frequentists, or frequentist philosophy. But the frequentist philosophy actually can be generalized much beyond the MLE to account for these situations. So in this course, we're going to focus on the maximum likelihood estimator, which is in the frequentist camp or the frequentist philosophy. We're not going to focus much or talk much about the Bayesian Perspective. The main reason for that is that this is today what is mostly being used in practice.

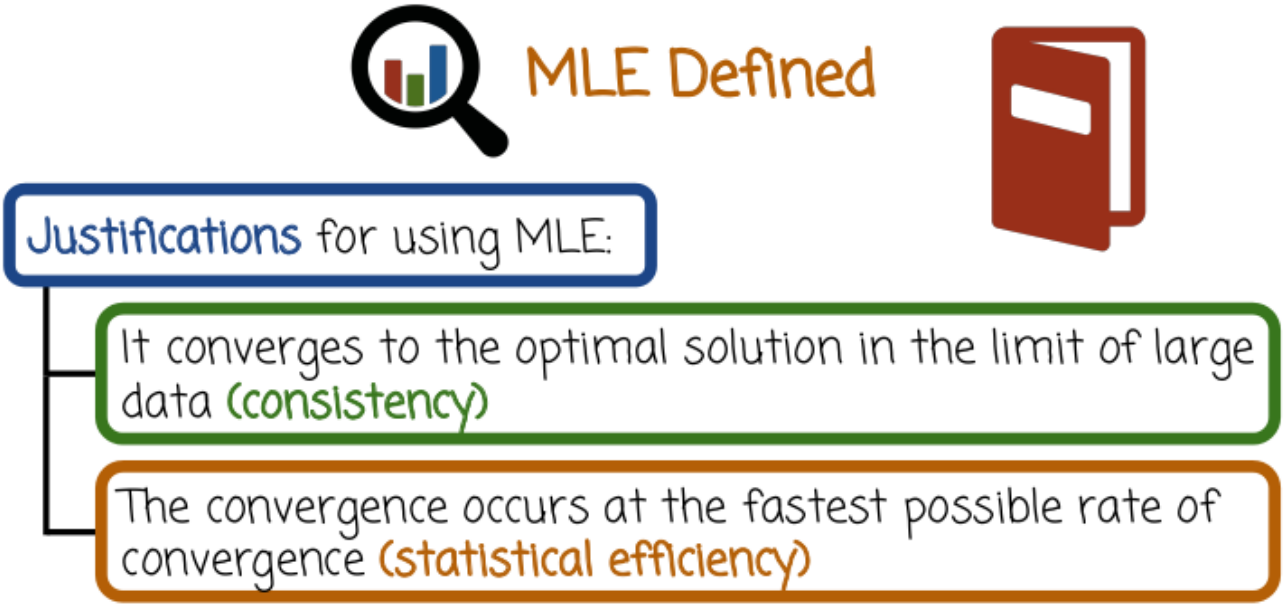
$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \quad p_\theta(Y = y^{(1)}|X = x^{(1)}) \cdots p_\theta(Y = y^{(n)}|X = x^{(n)})$$
$$= \arg\max_{\theta} \quad \log p_\theta(Y = y^{(1)}|X = x^{(1)}) + \cdots + \log p_\theta(Y = y^{(n)}|X = x^{(n)})$$

25. So, let's define the maximum likelihood estimator. The maximum likelihood estimator is the theta that maximizes the likelihood of the data. The likelihood of the data is the product of the conditional probabilities of the labels given the feature vectors where we take a product over all the probabilities of the training data pairs. So that product here is called the likelihood and it's a function of theta, because theta here describes the probabilistic classifier. Because it's a function of theta, we can investigate which theta maximizes it and take take that maximizer as the classifier that we're going to use. That is the maximum likelihood estimator. And that theta, we can use later in prediction time to classify new vectors or to predict new vectors. Because log is a concave function, the maximizer of the product will be the same as the maximizer of the log of the product. The product and the log of the product of the maximum will be different, but the maximizer. Meaning, the state of this maximizer's the product and the state of this maximizer's the log of the product will be the same. Think why would it be correct to say that the maximizer of an expression is the same as the maximizer of two times that expression. Think why this is true first and then try to see why the log would not change the maximizer. But the reason we take the log is that the log of a product is the sum of the logs that simplifies the expression here considerably, especially later when we want to compute derivatives. So the maximum likelihood estimator is the maximizer of the likelihood, which is the same as the a maximizer of the of likelihood log likelihood. Log likelihood being the expression that's written here. When we want to actually do the maximization, we're going to use that log likelihood expression, because it's going to be much simpler to compute and to maximize than the likelihood itself and the maximizer itself will be the same. So, it doesn't matter which one we maximize.



Why do we want to use the maximum likelihood estimator or the MLE? There's several motivations or justifications. One is known as consistency or the consistency property of the maximum likelihood estimator. The MLE will converge to the optimal solution in the limit of large data. As we get more

and more training data, remember, the frequency philosophy assuming there's one specific correct state of nature and that state of nature is within the family that we're considering. For example, the family of logistic regression classifiers. Then in this case, as we get more and more and more data, the maximum likelihood estimator will converge to the correct state of nature that generated the data. That's very appealing motivation. We certainly want our classifiers to be consistent. And as we get more and more data, we want them to become more accurate. And if they can get to the right solution or a nature, then that's even better. That would be great. In this case, we see the disclaimer and we get consistency, but assuming that data is generated based on a model within the logistic regression family. If we're doing MLE for logistic regression and you assume that n being the number of training pairs goes to infinity while the dimensionality is fixed.



The second motivation or justification is that the convergence specified here in the first bullet occurs at the fastest possible rate of convergence (看下下句更好懂). This is called statistical efficiency of the MLE. In other words, there is no classifier that will converge the truth at the faster rate (than the optimal classifier) and it also very strong motivation for why we want to use the MLE and I should probably give a few disclaimers here that when dimensionality is high, then some of these convergence results do not hold anymore, because they assume d is fixed and n goes to infinity. In many cases in practice, we can have d. Remember, I talked about millions or billions of features. D could be much bigger than n. In which case, this motivation here breaks down. Nevertheless, this is a very nice motivation and that has driven the use of MLE in much of the 20th century, including 21st century. And even today if you look at what big companies are doing whether they're using in production, it's some variation of MLE at least in many, many places. Maybe not every single place, but in many places.

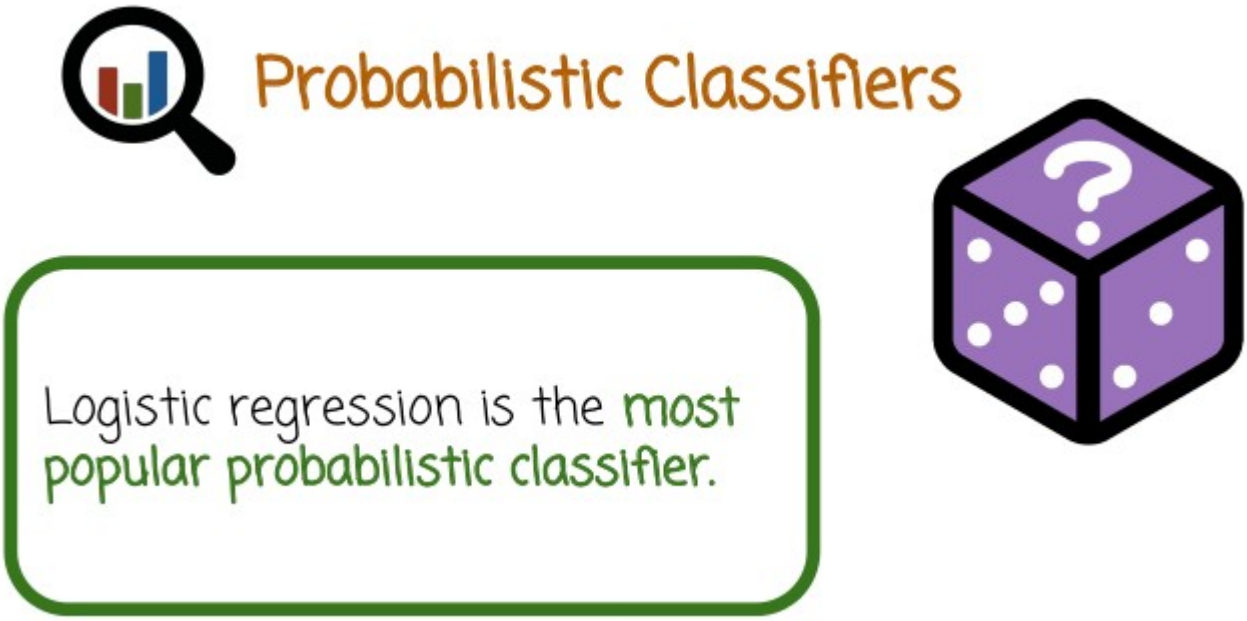
26. In this quiz, describe the computational procedure for reaching the value x for which the function f(x) is at the maximum. Does it scale to high dimensions x? So x is in this case is a vector. It's not necessarily a scaler. How can we find the computational procedure for maximizing f? This is a direct analogy or application to the maximization of the likelihood that the maximum likelihood estimator needs to do.

MLE Quiz

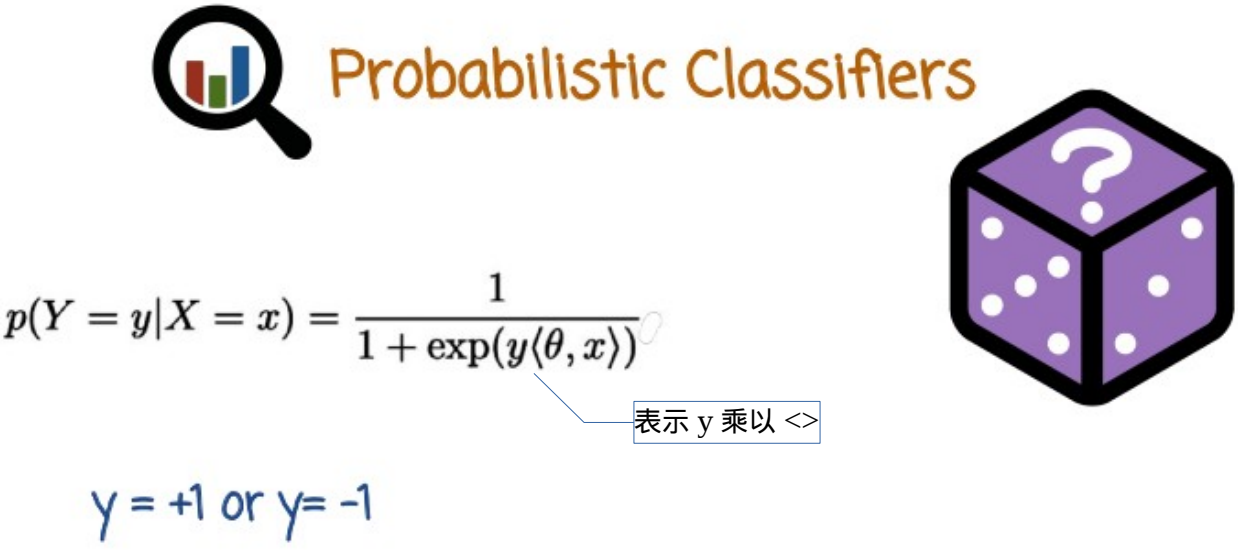
Describe a computational procedure for reaching the value x for which f(x) is at a maximum. Does it scale to high dimensions x?

- Compute f(x) on a grid of all possible values and find the maximum.
- Do this for scalars x or low dimensional vectors x.
- (It does not scale to higher dimensions. An alternative technique that does scale is gradient ascent.)

27. Here is one procedure: compute f(x) on a grid of all possible values, and find the maximum point. By just going over all possible points and checking which value is the maximum. We can do this for scalars x or low dimensional vectors x. The problem here is that this does not scale up to higher dimensions. Because as the dimensionality of x increases for example if we have two dimensions going on to 3, to 4, to 5, 20, 40, 100. The grid of all possible values would have more and more points. The number of points in that grid will grow exponentially with the dimension of x. If we have just one dimension we can say the grid can have ten points but if we have two dimensions the grid will have 10 by 10 or 100 points. If we have three dimensions the grid will need to be 10 by 10 by 10 or 1,000 points. Very quickly we'll get to a situation if d, the dimensionality increases, we'll get to a situation that we just cannot evaluate all points on that grid and the maximum using this computational procedure. So we'll need an alternative technique that does scale up with the dimensionality and that technique is called gradient ascent. We'll see it later.



28. So we'll make things more concrete by talking about logistic regression, which is one specific probabilistic classifier. When I say probabilistic classifier, that's a model or a mapping from a parameter vector theta into a probability for y given x. This is just one specific classifier, however, it's a simple classifier. The training scales up very well, both to high dimensions and to large data sets. The prediction is very simple, because it's a linear classifier. And it's very popular today, even though it's been around for many, many decades.



So let's see the definition of logistic regression. The probability of Y being either +1 or little y is the value that big Y, the random variable, takes. It can be either +1 or -1. Condition on a vector X taking on specific value little x. That definition, the formula is 1 / 1+e to the power of y, which, remember is +1 or -1, times the inner product of theta and x. So this is the formula of the logistic regression classifier. What is missing is the vector theta. Once we know the vector theta, logistic regression

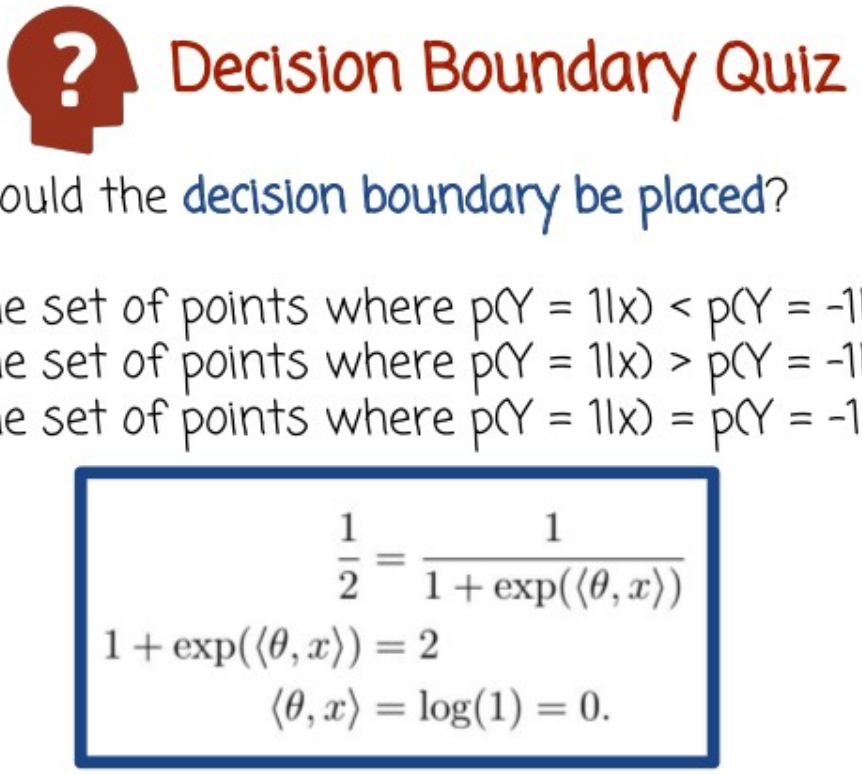
formula will give you a probability for y equals +1 or -1 given any vector of measurements x.

$$p(Y = 1|x) + p(Y = -1|x) = 1:$$

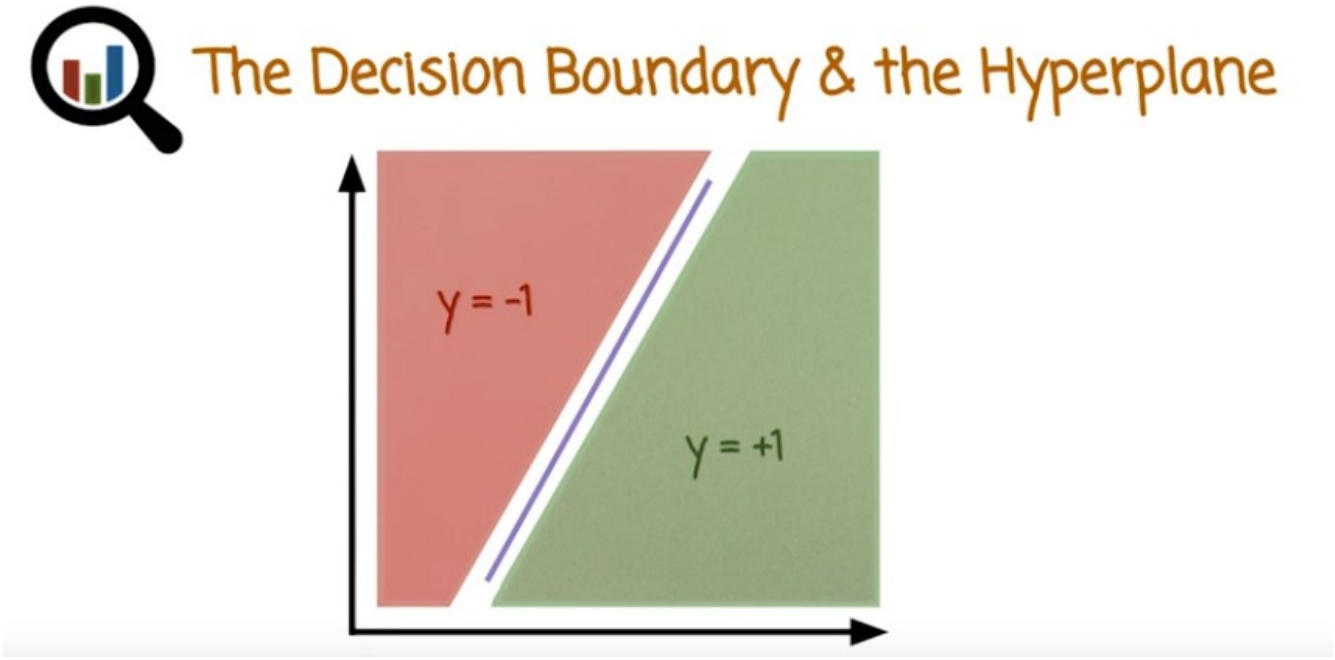
$$\frac{1}{1+\exp(\langle\theta, x\rangle)} + \frac{1}{1+\exp(-\langle\theta, x\rangle)} = \frac{1+\exp(\langle\theta, x\rangle)+1+\exp(-\langle\theta, x\rangle)}{(1+\exp(\langle\theta, x\rangle))(1+\exp(-\langle\theta, x\rangle))}$$
$$= \frac{1+\exp(\langle\theta, x\rangle)+1+\exp(-\langle\theta, x\rangle)}{1+\exp(\langle\theta, x\rangle)+\exp(-\langle\theta, x\rangle)+1} = 1$$

Let's first ascertain, or make sure that this is indeed a probabilistic classifier. The probability of Y equals 1 plus the probability of Y equals minus 1, both of them conditioned on x, we want to make sure it's indeed 1. This is the probability that Y is 1. Because Y is 1, it just gets absorbed, or we can just remove that. Plus the probability that Y is minus 1, we just, instead of Y equals 1, we write Y equals minus 1, in this case it's minus 1, and we just write minus the inner product. Instead of minus 1 times the inner product, we just write minus the inner product. We take the common denominator, and we put it all together using a common denominator. We do some algebra, and we get to the fact that the numerator cancels out with the denominator and we get 1. So that's indeed making sure that the two probabilities sum up to 1. The other requirement is that the probabilities are nonnegative. We can see that no matter what's the value of theta or x, this formula here, as well as this formula here, they're always going to be nonnegative. Therefore, this is indeed a legal probabilistic classifier without any definition issues.

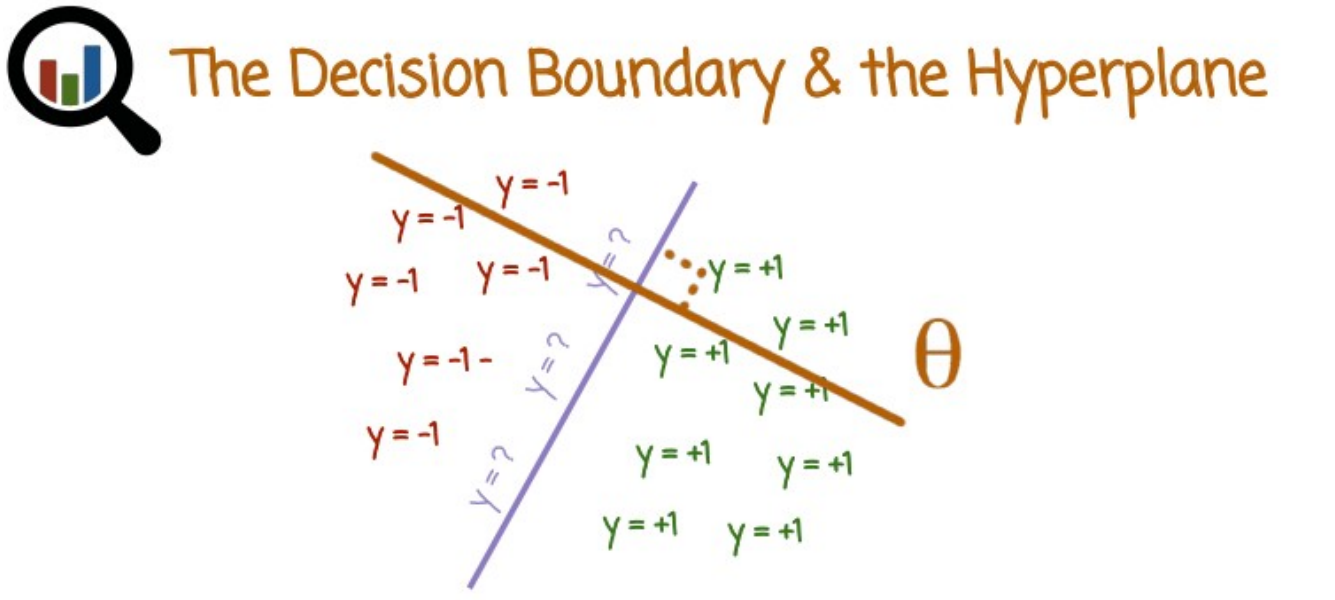
29. In this quiz, please answer the question. Where should the decision boundary be placed? The set of points where probability of Y equals 1 less than probability of Y equals minus 1? Set of points where the probability of Y equals 1 greater than probability of Y equals minus 1? Or the set of points where probability of Y equals 1 equals probability of Y equals minus 1 which equals one-half?



30. The correct answer is the third alternative. That's decision boundary is the point at which the prediction changes from positive to negative. That happens exactly where the probability that the label is 1 is exactly the same as the probability that it's minus 1, which is exactly one-half. In fact, if this probability is the same as that, then they have to equal both one-half, otherwise there is inconsistency. Let's explore this mathematically. So what happens when the probability that Y equals 1 given x is one-half? This is one-half, that's the probability of Y equals 1 given x, based on the logistic regression formula. We multiply by both denominators, we get this formula right here. We subtract 1 from both sides and take the log of both sides. We get that the inner product of theta with x is the log of 1, which is 0. What we see here is that the set of points for which the probability of Y = 1 given x is one-half. If that's the case, it has to equal the points at which probability of Y = -1 also equals one-half. The set of these points is also the set of points x for which the inner product between theta and x is 0. And that's the set of points which are orthogonal to the vector theta, describing the hyperplane, or the decision boundary, which brings us to where we were a few slides ago, where we talked about linear decision boundary and we said the vector theta is orthogonal or perpendicular to the decision boundary. That's exactly what we see here. The decision boundary or the point where the classifier doesn't know how to classify, it changes its mind from 1 to -1, is the set of points that are perpendicular or orthogonal to the parameter vector theta. And this is the linear decision boundary.



31. Let's see, what we saw earlier with formulas a little bit more pictorially. So we have a classifier described by a vector theta. That vector theta is perpendicular to the decision boundary here, the blue line. And the area where prediction is minus one, you see here red versus green area where the prediction is plus one. In this case, notice that decision boundary does not go through the origin, which means that there's an offset term. Alternatively, the the first or the last or one of the components of x is going to be forever defined to be one. That means we're going to have an off setter or biaster implicitly, even though we're still going to use the inter product formulas.



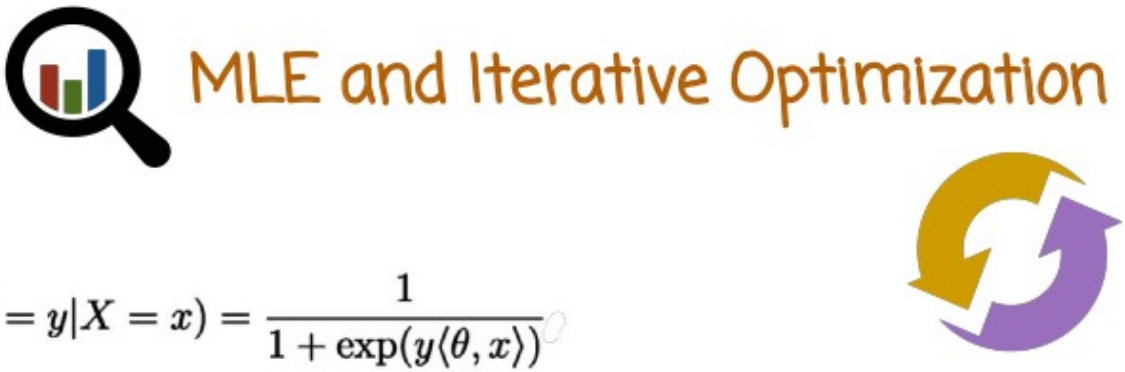
If we zoom in into the decision boundary area, on both sides we have different predictions. We zoom in exactly on that line which is infinitesimally narrow, the decision can be either plus one or minus one,

because the classification probability is exactly one half. And this is again showing that the parameter of vector theta is perpendicular to the decision boundary.

Task	Use
Predict the label associated with a feature vector x	prediction rule sign (θ, x)
Measure of confidence of that prediction	$p(Y = y \mid X = x) = 1/(1 + \exp(y\langle\theta, x\rangle))$

上表最後一行就是 logistic regression.

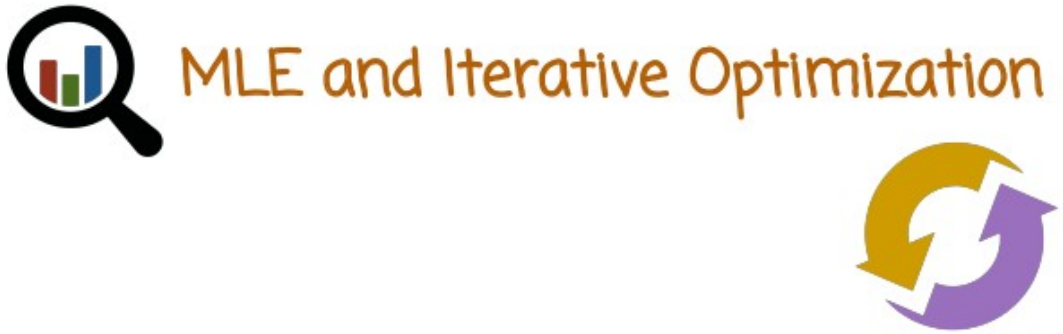
32. So let's see prediction confidence, summarize what we've seen before. If we want to just predict the label associated with a feature vector x, or a measurement vector x, we're just going to use the prediction rule, which is the sign of the inner product between theta and x. If we want a measure of confidence of that prediction, we can use the logistic regression formula assuming that we trained the model or gotten the vector theta using maximum likelihood for logistic regression. We can use that formula right here and not only will it gives us a confidence measure, that confidence measure has a direct interpretation as probability



$$p(Y = y|X = x) = \frac{1}{1 + \exp(y\langle\theta, x\rangle)}$$

$$\begin{aligned}\hat{\theta}_{\text{MLE}} &= \arg\max_{\theta} \quad p_\theta(Y = y^{(1)}|X = x^{(1)}) \cdots p_\theta(Y = y^{(n)}|X = x^{(n)}) \\ &= \arg\max_{\theta} \quad \log p_\theta(Y = y^{(1)}|X = x^{(1)}) + \cdots + \log p_\theta(Y = y^{(n)}|X = x^{(n)})\end{aligned}$$

33. So lets put the maximum likelihood formula (上圖最下面那兩行), as we've seen before, with the logistic regression formulas we've seen more recently and see how we can express the log likelihood, simplify it, and come up with a computational procedure for maximizing the likelihood. So, this is a reminder, the formula of logistic regression is given right here. The likelihood is the expression right here, the product of the conditional probabilities of the labels given the feature vectors or measurement vectors. And we want to maximize that product, but the maximizer is the same as the maximizer of the log of the product. The log of the product being the sum of the logs because of the property of the logarithm function.



$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \sum_{i=1}^{n} \log \frac{1}{1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)} = \arg\max_{\theta} \sum_{i=1}^{n} -\log\left(1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right)$$
$$= \arg\min_{\theta} \sum_{i=1}^{n} \log\left(1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right)$$

And so we're left with maximizing the sum of the log of the probabilities, of the labels given their data points. We plug in the logistic regression formula for the probability, and we get this expression right here. The first thing that we notice is that a log of 1 / a is the same as minus log of a. And it's not very convenient to keep a denominator and a numerator right here, so we're just going to say log of 1 over that expression here equals minus the log of that expression. Then we don't have to deal with the fraction anymore. The second thing we notice is we have here, basically find the maximizer of this expression right here. But there's a minus here, so the maximizer of the minus of some expression is the same as the minimizer of the expression without the minus. And we want to simplify so we want to get rid of the minus. So instead of saying maximize minus of an expression, just find me the minimizer of that expression. That will be exactly the same point as the maximizer of this. That's why the equality sign here reflects the fact that the minimizer of this is the same of the maximizer of this.

Gradient Descent

a. initialize the dimensions of θ to random values

b. for j = 1, ..., d update

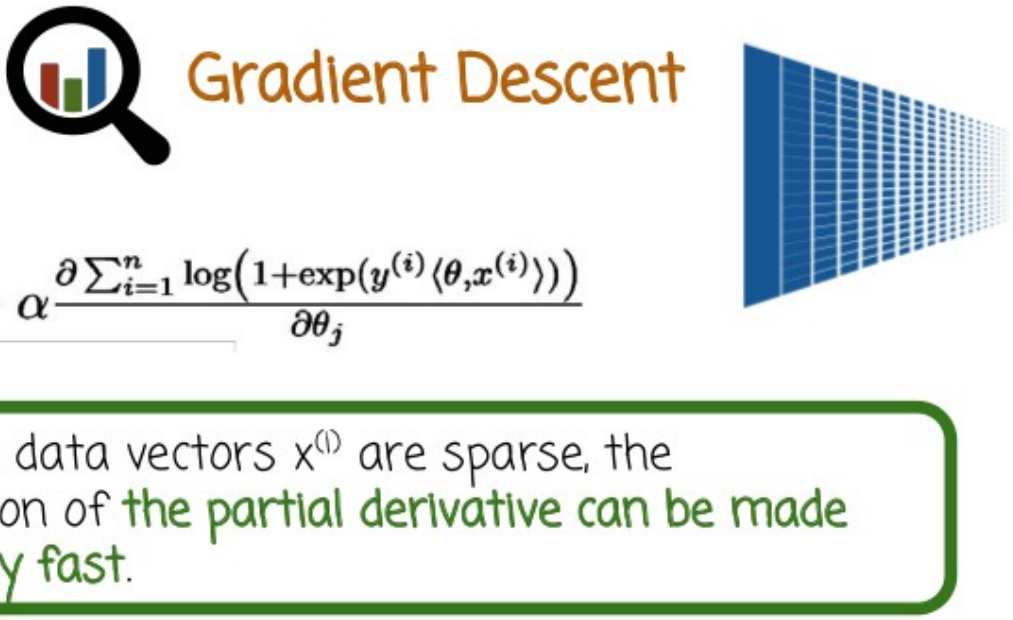
$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial \sum_{i=1}^{n} \log\left(1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right)}{\partial \theta_j}$$

c. repeat the update (step b) until the updates becomes smaller than a threshold

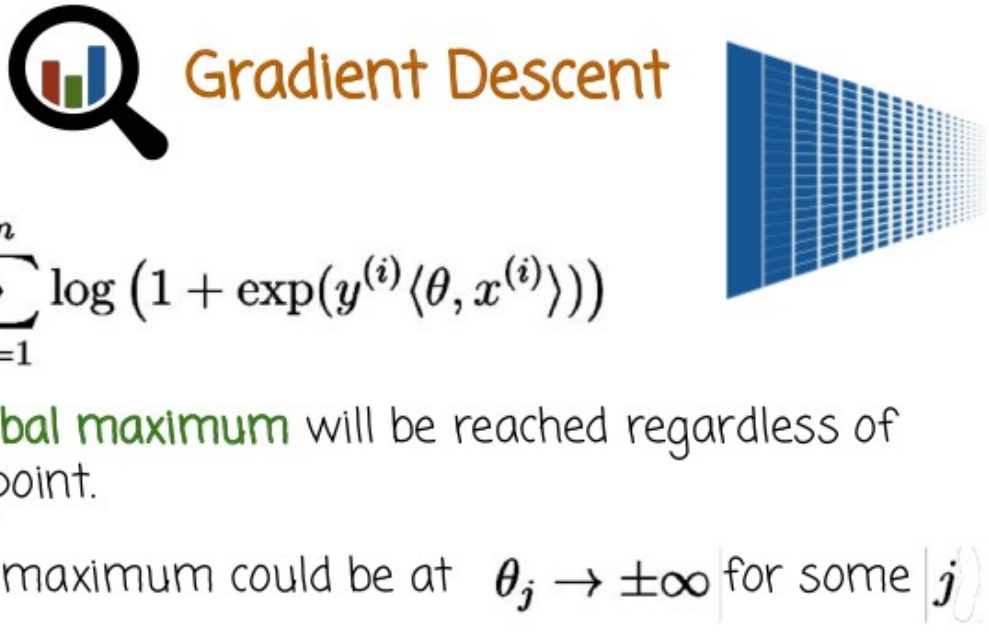
Let α decay as the gradient descent iterations increase

34. So we saw before that we still have a problem of maximizing or the case we had before was minimizing, finding the minimizer of an expression. And in one of the quizzes before, we saw that one procedure for doing that is to create a grid of all possible values. Evaluate the log likelihood or the function we want to minimize or maximize on all the values and find the minimum or maximum point, but that approach does not scale up to high-dimensional parameter vector theta. So, let's see a different method called gradient descent for finding the minimum of a function where the minimum of a function is computed for vectors that may grow to higher dimensionality and the procedure is relatively scalable. Meaning, it can work pretty well, even when the dimensionality is pretty high (這就是為何 machine learning 中老愛用 gradient descent 之原因). So, the first step is to initialize the dimensions of theta to random values. Second step is for every dimension of the vector theta, update the component theta j according to the rule right here. The new theta j is the old theta j minus alpha, which is some number that's called a step size times the partial derivative of the log likelihood or the negative log likelihood, in this case, with respect to theta j. The expression right here is a function of the vector theta. And when we want to take the derivative with respect to theta j, that's going to be a partial derivative. We're going to fix all the other components and take the derivative only with respect to theta j. That tells us how much we should decrement the current theta j and we do that for all dimensions. We're going to repeat step b until the updates that we get here becomes smaller and smaller. And eventually, becomes smaller than a certain threshold that we think is too small to care about or maybe we don't have enough time. We want to finish and be done with it and then we're going to terminate the procedure and the resulting vector theta j that we have is our maximum likelihood estimator or the minimizer of the negative log likelihood or maximizer of the log likelihood, which is the same as the maximizer of the likelihood. The number alpha is the step size and that typically, either we keep it constant or either better let alpha go down as we have more and more gradient descent iterations. So the beginning of the gradient descent procedure, alpha maybe relatively larger. And then as we iterate more and more, we slowly decrease alpha. The rate at which we decrease alpha matters and there is some practical algorithm, as well as theory to try to reason or say something about at what rate we should let alpha go down. For example, it could be the square root of the number of iterations

or whatever. There are different ways of doing that. I don't want to get into that at this point, but this is the grade in the sand procedure. And at the end, if you choose the alpha to go down appropriately and the termination procedure or the threshold to be sufficiently small. The resulting vector theta would be very close to the likelihood of the logistic regression. That also applies more generally to any other function you want to minimize or maximize. Now the reason this works and the reason it's called gradient descent is the step b, basically, means that the vector theta is adjusted with the gradient. The gradient is the vector of partial derivatives. And basically, if we want to write this step b in vector form, it will be vector theta is the old vector theta minus alpha times the gradient. And the gradient, if you remember from calculus is the direction of steepest ascent. And here, we have minus. So, minus of the gradient would be direction of steepest descent. So the direction at which, if we were go in, the function would go down the fastest. And so if we keep following the gradient or the negative gradient if we want to minimize and we keep decreasing the step size alpha, we'll keep going down and down and down until we reach a local minimum. Or in this case, because we are minimizing the negative of likelihood, we're going to maximize the likelihood or the likelihood. We're going to reach the hilltop or some sort of maximum point that has a gradient of zero or sufficient close to zero to indicate this is the local maximum. If we go from that point to any other neighboring point, we're only going to make the likelihood lower not bigger. Gradient descent is relatively easy to compute and to scale up both with the size of data as well as with the size of dimensions.



One of the nice properties is that, especially for the case of logistic regression, because of the linearity property, if the data features x or the parameter vector theta are sparse, then that inner product right here can be done very, very fast. Fast even if the dimensionally is very high, that means we can scale up gradient descent to very high dimensions assuming that the vectors x where the data are mostly sparse. We can even scale them up to millions or more than millions without much problem. The computational load depends on the sparsity pattern rather than the dimensionality.



35. So let's review what we have before. This is the minus of the log of the probability of the label given the training point. We take the sum of all such logs and then we want to minimize that because this is minus of the log of the probability. In the case of logistic regression, that's another nice properties that the likelihood function is concave, meaning that there are not more than a single global maximum. So if we use gradient descent, we slowly or fast will converge to the maximizer of the likelihood regardless of where we start. Specifically, we will not be captured in a local maximum while there's another local maximum that is even higher, but it's further away and we will not get to it with gradient ascent. That's something that could happen, for other maximum likelihood estimators for other models, not logistic regression if the likelihood is not concave. But in the case of logistic regression, the likelihood function is sufficiently nice in some sense. You can think of it as there being only one hill rather than being multiple hills. If you think about a topographic map, there is a single hill on the topographic map and we try to get to that hilltop as opposed to a more complex landscape of multiple hills. And then if you do gradient descent you'll converge to the nearest two point depending on where you start. You remember our starting point is random. So it's kind of nice that no matter where you start, you'll converge to that one global hilltop. Although one caveat is that it could be that the hilltop, the maximizer of the likelihood could be when theta j goes to infinity or to plus infinity or minus infinity for some dimensions. In that case if you want to think of it graphically imagine a topographic landscape of the likelihood function where there is no single hilltop but rather the hill or the mountain keeps going up and up and up and up. The more you go to the left or to the right, North, South, East, West, the more you go in a specific dimension, you keep climbing up and up and up. And so there is no single maximizer. So that could happen. And in which case,we need determination criteria to tell us that, okay we've explored enough. We keep going to infinity, we better terminate, and use the current values of the vectors Theta. We'll see later that when this happens quite possibly we have a situation of overfitting. And this is a possible problem and we'll see some ways to address that later on. But for now before we learn about both overfitting and regularization, let's just keep in mind that's a possibility in gradient descent for logistic regression. That the maximum will be when one of the theta j's will go to plus or minus infinity. In which case, we don't need to consider it a problem. Rather, we just terminate at some point and use the model that we have.

Stochastic Gradient Descent

Amount of Data	Preferred Technique
non-massive data	Gradient descent
massive data	Stochastic gradient descent

36. Gradient descent, like I mentioned earlier, scales up pretty nicely with the dimensionality and also with the size of the data. But in cases where we have a real massive dataset, it may converge relatively slowly and there is an alternative called Stochastic Gradient descent, which works faster and better. Stochastic Gradient descent is now the method of choice in many places, when they have to do maximum likelihood estimation. Or maximize some other function for a different reason when there's a lot of data involved and gradient descent just does not work fast enough. The reason for the difficulty of gradient decent to handle large datasets is that every iteration, where you iterate over all the dimensions you have to go over every single data point. In contrast Stochastic gradient descent takes a different view point, and what it does is it's updating the parameter vector theta, as it traverses the dataset it gives more updates continuously as opposed to waiting to give an update to the parameter vector until doing the full loop of the entire data set, like gradient descent. And that means we can see updates to the parameter vector of theta faster. In fact, by the time we do a complete loop over all the training set, we will have already updated theta many, many times. And when we compare the accuracy of the estimator or the performance as a function of a specific amount of computation time, Stochastic gradient descent, we just converge much faster because of the fact that it takes many small steps. A small step after seeing every data point as opposed to gradient descent to do one step it needs to look at the entire training set. So we see here preferred technique from massive data is Stochastic gradient descent, and let's see how that works.

Stochastic Gradient Descent

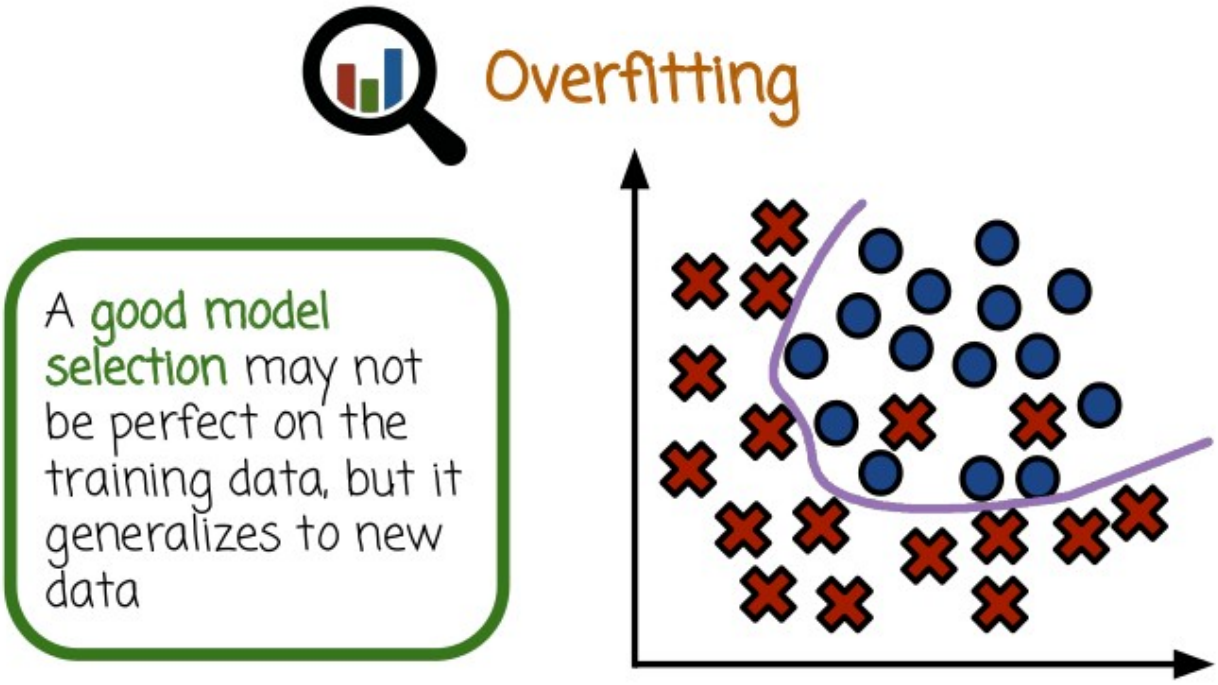
a. initialize the dimensions of θ vector to random values

b. pick one labeled data vector $(x^{(i)}, y^{(i)})$ randomly, and update each

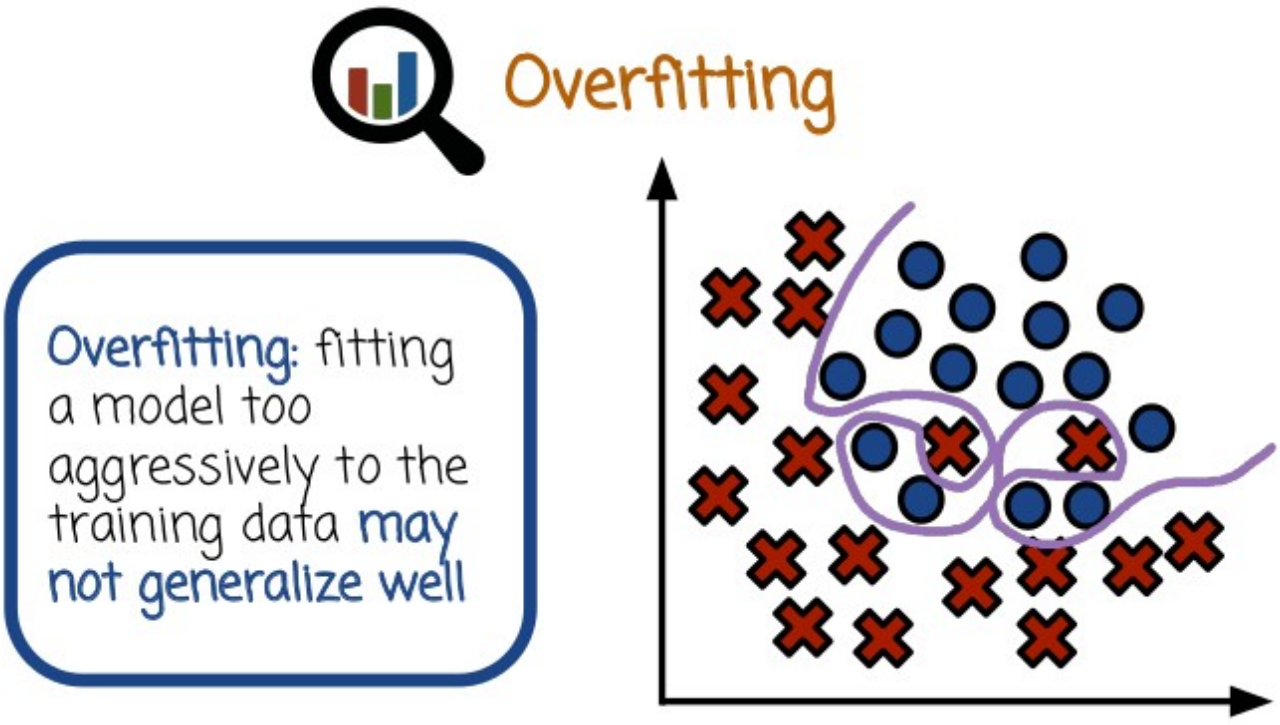
$$j = 1, \ldots, d: \ \theta_i \leftarrow \theta_j - \alpha \frac{\partial \log\left(1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right)}{\partial \theta_j}$$

c. repeat step b until the updates of the dimensions of become too small (reducing alpha as the number of iteration increases)

So we start by initializing the dimensions of theta to random values just like in the case of gradient descent. But here is the difference, the second step of the iteration, we pick a labeled pair of measurement vector and label, we pick that randomly. And then we update each of the dimensions doing a gradient, like a gradient descent step but based on just one single example. So you can think of it as computing the log likelihood, and taking gradient of the negative log likelihood, based on just a single example and updating all the dimensions. We do that after seeing one example, chosen randomly, and then we choose some other example randomly, and we do another update. Then we choose another example, we do another update. We repeat step b until the updates of the dimensions of the theta become too small, or basically, under some threshold. As before, we have a step size alpha that needs to be decreased typically as the iterations increase. So, as you see here step b would happen, the computational load of step b is much lower than in gradient descent because it's just based on a single data point. And specifically, it could be that the dataset is partitioned to, and resides across multiple computers, for example, in a cluster or in the cloud. In which case different computers can compute different Stochastic gradient descent iterations because they hold different parts of the data. As opposed to if you wanted to do gradient descent, you need to first get all the data to specific place where you compute all the components of the gradient. Then you add them all together then you do the step which could take a long time. In contrast to Stochastic gradient descent, computation load here is much lower.



37. A good model may not always perform well on the training data, but it will perform well on new data, or at least as good as can be expected. Here is an example we have training data of, plusses are positives examples the circles are negative examples. This is the training set and one particular classifier is expressed here using that purple line. And now the question is this a good classifier or a bad classifier? Well, it misclassifies two examples in the training set, because it predicts them to be on that side of the decision and it predicts that these two are negative whereas we know that they are positive. So it makes some mistakes on the training set, but that's not necessarily a bad thing, because sometimes it's just some noise that we cannot avoid, some sort of variation in the data. And if we try to perfectly classify all the pluses on one side and all the circles or negatives on the other side, we'll get a classifier that is a perfect fit for the training data, but that classifier is going to make a lot more mistakes in future data points. In that situation is called over-fitting, where we try to fit the training data so badly, almost perfectly, very aggressively. We do whatever we can. We consider very complex classification rules, very complex decision boundaries. We do whatever we can to fit the training set and by doing so we get to classify that is very accurate on the training set, but it actually learns some artifacts, or noise that is not true signal. And that classifier is not going to perform well in the future, classifying future data. Which is a problem, because we don't need to classify the training set, this is already classified, already labeled. What we need is to classify future data.



So this is an example of an over fitting classifier. A classifier who's decision bounder is this purple line here. Perfectly classifies the training points, but very likely to not work well once we get future data. Because of its convoluted shape and the fact that it does whatever it can to fit the training data, even up to fitting some noise that is inherent in the data or some statistical overlap between the pluses and the circles.

Overfitting becomes particularly difficult when the dimensionality of the feature vector and the parameter vector is high, especially when compared to the size of the training sets. So when the dimensionality is high and the number of training pairs is relatively low over fitting could be a very big problem. Another way to think of it intuitively is there are a lot of dimensions, not enough data. There are too many parameters to estimate from a relatively small training set that does not have a lot of signal. We tried to estimate a lot of parameters from let's say the extreme case think about single example that's going to be a problem. So we basically will over fit random noise pattern rather than over fit the signal that it's going to describe how future data points arrive or are generated.

Example:

$d = 10^6$ and $n = 2$

(d = dimension of x vector,
n = dimension of the training data)

Here's an extreme example, dimensionality is a million, a million dimensions. And only two data points, one is positive, one is negative. So think about a Euclidian space that has a million dimensions, million dimensional space and there's only two points. One positive, one negative. How can you possibly learn a linear classifier that is going to do a good job for future data when you have a million dimensions and million parameters to learn in only two points. So, what do we do in these cases? There are several alternatives, one is to choose a classifier that is more resistant to over fitting. Linear classifiers are relatively resistant to over fitting. But if the dimensionality is very high, the data is not very big. And in particular, if dimensionality gets higher and higher by adding non linear transformations of the original features in a way that I described earlier to avoid the constraint of having a linear decision boundary. We add more, and more, and more non-linear transformations, then the number of able pairs still stays the same, a linear classifier can still overfit in this case. And there are some techniques, one of them is called regularization. Another would be to try to do some sort of feature selection. Try to figure out which features are not actually relevant and remove them from the data.

Regularization

Regularization terms added to the maximum likelihood cost function

$$\beta\theta_1^2 + \cdots + \beta\theta_d^2 \quad \text{or} \quad \beta|\theta_1| + \cdots + \beta|\theta_d|$$

High values of θ are penalized
MLE is tempered from achieving high values of $\theta_1, \theta_2, \ldots, \theta_d$

β is selected through experimentation
β should be close to the optimal value

38. So I mention earlier that regularization is one technique to address over fitting, which happens in high dimensionality usually. Regularization refers to adding another penalty term to the maximum likelihood cost function or objective function. This(左邊的) penalty right here is called L2 penalty. Beta is some number and that multiplies the squares of the components of the future vector theta. This (右邊的) penalty right here is called the L1 penalty, and that's beta times the absolute values of the different components. These are the two most common regularization penalty terms, although there are others. Both of these penalty terms basically penalize high values of theta (注意在 ML 課中講了, 對於 neural network, 很大的 weight 值容易導致 overfit, 這裡的 theta 應該差不多). So when you think about maximizing the likelihood, remember that gets converted to minimizing the negative log likelihood. So we want to minimize the cost function and then we add this penalty terms. And these penalty terms prevent the theta from becoming bigger and bigger, because as it becomes bigger and bigger, it's going to be a penalty and it's going to make the objective function increase, and so in some sense we're taking the negative likelihood function and we are adding a penalty for large status, you can think of it as telling the likelihood function be careful with your maximizer. The likelihood maximizer, if the likelihood maximizer has a high values of theta, either high positive or very low negative and be high in both sides. Notice we don't take theta, it's theta squared or theta absolute value because a very, very high negative is also bad like -1000 is as bad as +1000. We want it to be close to zero. You can think of us telling the likelihood function, be careful with your maximizer if it has high values of theta or some dimensions of theta are particularly high. If there is such a maximizer, it's going to get penalized. And so it needs to be really, really good at treating the data in order to justify being selected if it's big. It's going to need to overcome that penalty, and so the result of adding this penalty and depending on the value of beta, beta controls the size of the penalty. And when you add that penalty, basically the maximum likelihood estimator will, penalize maximum likelihood estimator will favor more values of theta that are not associated with some very high or very low components. It's not very easy to know how to select beta (不是 theta) without experimentation. There's some theory, but in practice, what is typically done is, the ML, the penalized MLE or the MLE with this penalty terms is computed for different values of beta. And then the best value is selected based on performance on future unseen data. One other comment I'd like to make is that when you add these penalty terms to the logistic

regression maximum likelihood, you prevent optimal theta from being achieved at infinity. You remember earlier I said that even though log likelihood is concave, and the maximizer is at most unique, it could be that the maximizer is at plus infinity for some component of theta, or a minus infinity. When you add these penalties, you prevent that from happening, because as you increase theta the penalties are going to become bigger and bigger and bigger. And at some point, the likelihood will start to get worse, or go down, even though you increase theta and you get higher and higher and higher in the hill. Once you add the penalty, the hill will all of a sudden have one single hilltop. So that's sometimes practical, even if the dimensionality is not high, to add a regularization to avoid that situation where the maximizer of data is at plus or minus infinity.

Notes:

The Shiny App (Web App embedding R code) shows how logistic regression can be used to define non-linear classifiers in the original space:
Interactive Logistic Regression
https://tonyfischetti.shinyapps.io/InteractiveLogisticRegression/

The tutorial: Pyhton Logistic Regression shows Python code implemeting logistic regression and visualizing the results and the training process.
http://nbviewer.jupyter.org/gist/vietjtnguyen/6655020

39. In this lesson we learned what are classifiers, linear classifiers, and probabilistic classifiers, and how to train probabilistic classifiers using the maximum likelihood paradigm. We saw, in detail, the equations and model behind logistic regression, and how to apply it in practice.