Data Science and Big Data Analytics: Making Data-Driven Decisions

Module 3 Classification Hypothesis Testing and Deep Learning

Lecture Notes

Overview

The previous module on regression for prediction has laid the foundations of input/output learning that constitutes the basis of machine learning. Professor Chernozhukov also covered confidence intervals, which provide error bars rather than a single prediction. In some sense, prediction and confidence intervals are complicated questions because they require data to speak for itself. In a much more directed setting where data is asked simpler questions with yes/no answers, these answers may be answered much more accurately. This is the premise of hypothesis testing, anomaly detection, and classification.

Professors Gamarnik, Kelner and Moitra give us a unified overview of the techniques associated with such questions by describing techniques, ranging from statistics to machine learning. Specifically, this module covers:

1. The basics of anomaly detection and classification: for these tasks there are methods coming from either statistics or machine learning that are built on different principles.

2. Fundamentals of hypothesis testing, which is the formalization of scientific inquiry. This delicate statistical setup obeys a certain set of rules that will be explained and put in context with classification.

3. Deep learning: this recent technique has been the driving force behind the rise of artificial intelligence. Professor Ankur Moitra will demystify this method by describing its underpinnings and limitations.

Goals

1. Understand the framework of hypothesis testing in the context of the Neyman-Pearson paradigm

2. Survey basic machine learning methods for classification

3. Understand the fundamentals of deep learning and more generally of representation learning

Objectives

Students should be able to:

1. Understand the principles behind anomaly detection
2. Formulate a hypothesis testing problem
3. Understand the concept of p-value
4. Describe several machine learning algorithms for classification
5. Understand the concept of deep learning
6. Describe backpropagation

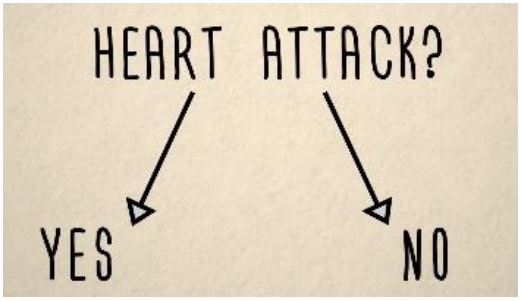
3.1 Introduction

The goal of classification is to be able to place a given object into the appropriate category based on its attributes, typically given by data associated with this object.

3.1.1 What are anomalies? What is fraud? Spams? - Part 1

In this module, we'll introduce statistical methods that assist companies in assessing these types of risks. It'll be our first example of a very common category of questions which we'll refer to as binary classification.

3.1.2 What are anomalies? What is fraud? Spams? - Part 2



All of the examples above fall into the category of so called binary classification, where the goal is to place the given object into one of two possible classification groups.

One artificial model based on how animals solve such a binary classification problem is called the neural network.

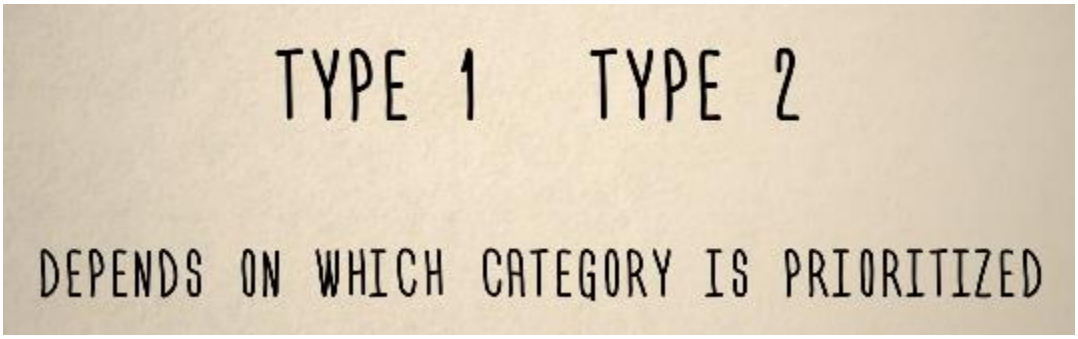
3.1.3 What are anomalies? What is fraud? Spams? - Part 3

The implications of misclassification actually may have very different consequences with different levels of severity. This brings us to the subject of type one versus type two errors.

When solving a binary classification problem, we encounter two types of errors. Consider again a medical fraud example. The two categories associated with this example are legitimate and fraud. The two types of errors we're about to discuss relate the ultimate classification decision made by a classifier to the crack but unknown to the classifier category.

Suppose for example an insurance company receives a claim which happens to be a legitimate one. Though of course company has no way of knowing that for sure. Suppose further that an automated system designed to make a preliminary determination declares the claim to be a fraud clearly making a mistake, somewhat arbitrary and somewhat noncreatively. We call this type 1 error to contrast it with the following mistake naturally called type 2 error. A claim which happens to be a fraud though unknown to the classifier is received but the classifier determines it to be a legitimate one.

Consider our example of a patient with severe symptoms who is rushed to an emergency center. Suppose this patient is not having a heart attack, even though he has severe symptoms that make the heart attack hypothesis very plausible to doctors. Again, this is clearly not something the doctors know a priori. Suppose they make an incorrect inference and decide that the patient is having the heart attack, and they conduct the treatment appropriate for a heart attack. This is a type 1 classification error. Conversely if a patient is indeed suffering a heart attack but the medical team fails to see this we're dealing with a type 2 error.

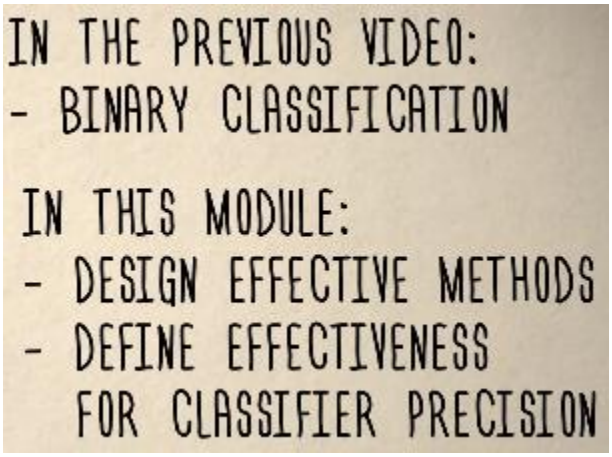


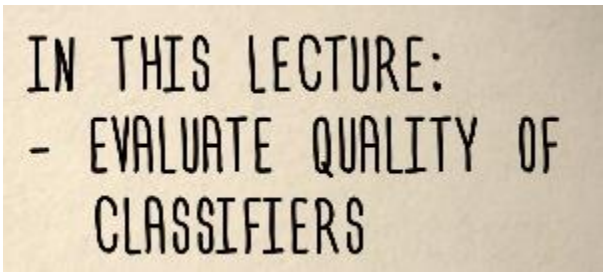
Deciding which type of the classification is type 1 versus type 2 is clearly somewhat arbitrary. And depends on the category which is declared to be the first category and which one is the second one. This is really up to the classifier, and simply is a matter of labeling the classes. Typically, in statistics the classes are called null hypothesis and alternative hypothesis, and we will talk about this in our future lectures.

What is important to keep in mind though, is the potential difference in severity of making type one versus type two error.

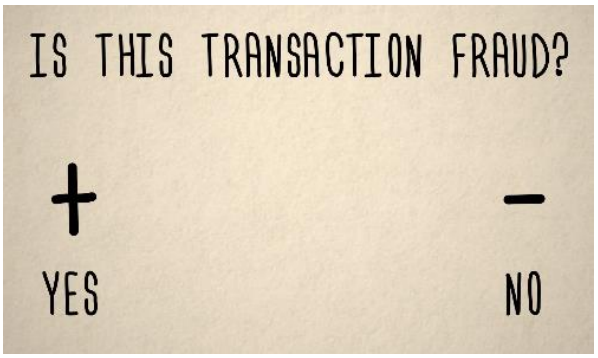
3.2 Binary Classification: False Positive/Negative, Precision / Recall, F1-Score

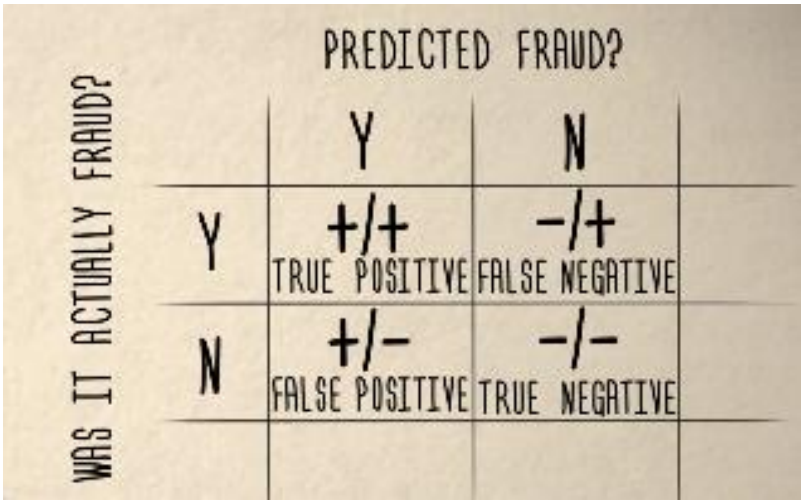
In the previous section we introduced binary classification, in which the goal is to put an object into one of two possible categories. Our aim later in this module will be to design effective methods for performing this task. But in order to do this, it's important to have a well-defined notion of what it means for a classifier to be effective.





When presented with a transaction, the classifier has to either label it fraud or not fraud. We discussed two different ways that the classifier could make a mistake. A type I error, where we identify a legitimate transaction as fraud, and a type II error, where we label a fraudulent transaction as legitimate.





It's useful to think of the classification problem as answering the yes or no question, is this transaction fraud? So that we can then describe the categories as positive, corresponding to the answer, yes, this is fraud, and negative, corresponding to the answer, no, this is not fraud.

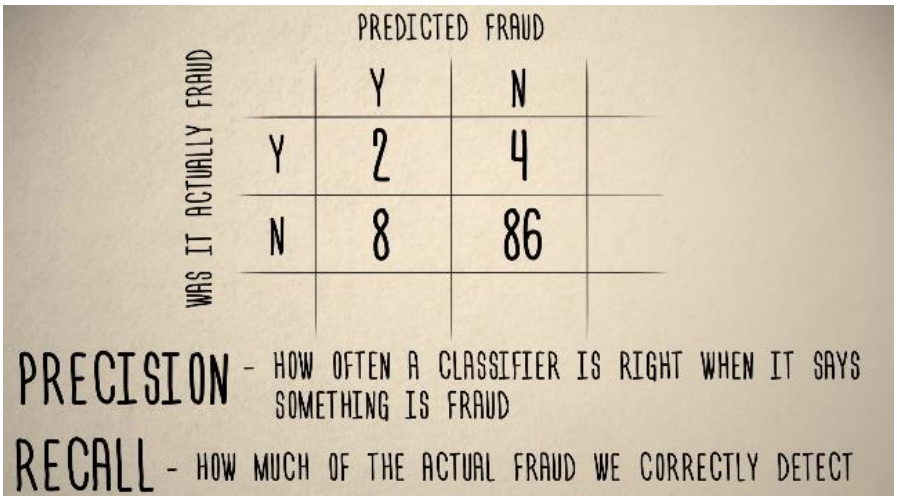
It's often helpful to arrange the possibilities as a table, where the columns correspond to the answer the classifier gives, and the rows correspond to the true underlying answer. We thus have four possibilities.

The first is that the transaction was fraud, and the classifier correctly labelled it as fraud. In this case the true category was positive and the label was positive as well. We'll call this a true positive.

The next possibility is a type I error. Here, the transaction was not fraud but the classifier incorrectly said that it was. In this case, the true category was negative, but the classifier falsely labeled it as positive. So we call these false positives.

The third possibility is a type II error, where the transaction was fraud, so the right answer was positive, but the classifier falsely labeled it as negative. So we call these false negatives.

The final possibility, which hopefully is the most common, is that the transaction was legitimate. So in this case, the true answer was negative, and the classifier correctly labelled it as such. We call these true negatives.



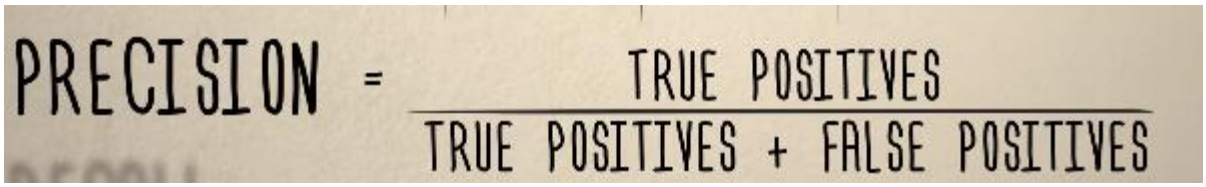
We can then visualize the performance of the classifier by writing in each box the number of times that the corresponding possibility occurred, forming what is sometimes referred to as the confusion matrix.

Note that the upper left and lower right numbers correspond to correct answers and the upper right and lower left ones correspond to mistakes.

To get a better evaluation, we need an approach that provides more nuanced information by the different ways in which the classifier can be right or wrong. A common way to do this is to look at two numbers, precision and recall.

Roughly speaking, precision will tell us how often the classifier is right when it says something is fraud.

And recall will tell us how much of the actual fraud that occurs we correctly detect.

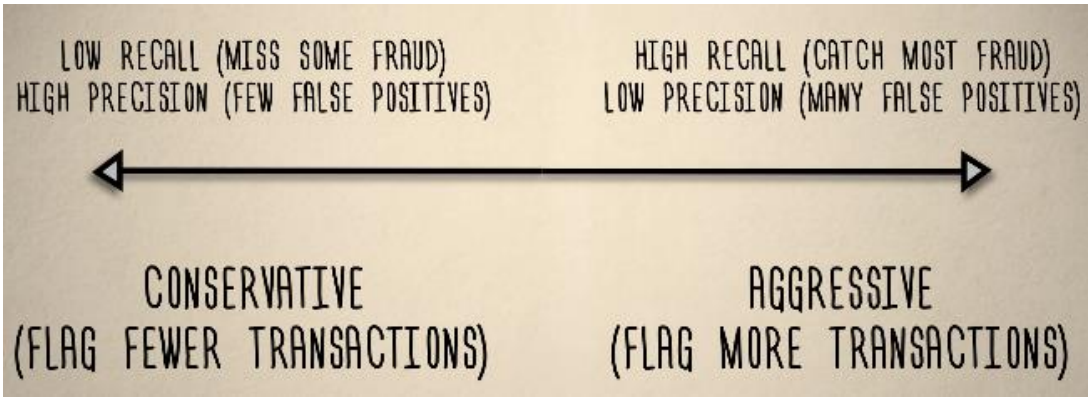


We define the precision to be the number of true positives divided by the number of true positives plus the number of false positives. Note that the denominator equals the total number of examples that the classifier labeled as positive. This number tells us what percentage of the time an instance labelled positive was actually positive. In our fraud example, the classifier labeled 2 plus 8 equals 10 instances as fraud. Of these, 2 were actually fraud, so the precision was 2/10 = 0.2.

We then define the recall to be the number of true positives divided by the number of true positives plus the number of false negatives. Here, note that the denominator equals the total number of examples in which fraud actually occurred. This number tells us what percentage of the actual positive examples our classifier detects. In the fraud example, there were 2 plus 4 equals 6 instances of fraud, and the classifier correctly caught two of them. So the recall here was 2/6, which is 1/3.

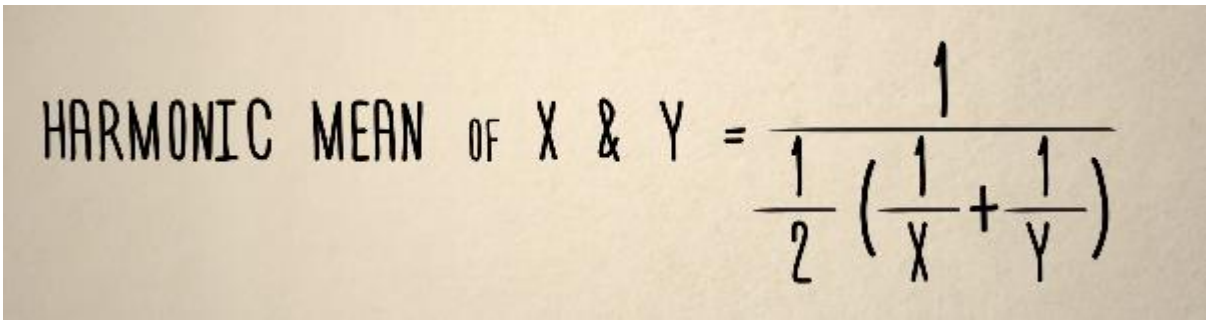
This number measures how sensitive our classifier is to indicators of possible fraud, so it's sometimes called the sensitivity.

Note, this would actually catch the problem with the lazy fraud detection algorithm that we described before. The one that describes everything as legitimate, since in this cases the sensitivity would be zero.



Know that there's often a trade-off between these two quantities. A more aggressive fraud department that flags a huge number of things as fraud, sometimes incorrectly, would have a high recall, since it would catch a lot of the fraud that occurs. But it would have a low precision, since it would also flag a lot of legitimate transactions as fraud.

On the other hand, a very conservative department that only flags the most obvious cases would probably have high precision. But it would miss the subtler cases of fraud, and would thus have lower recall.

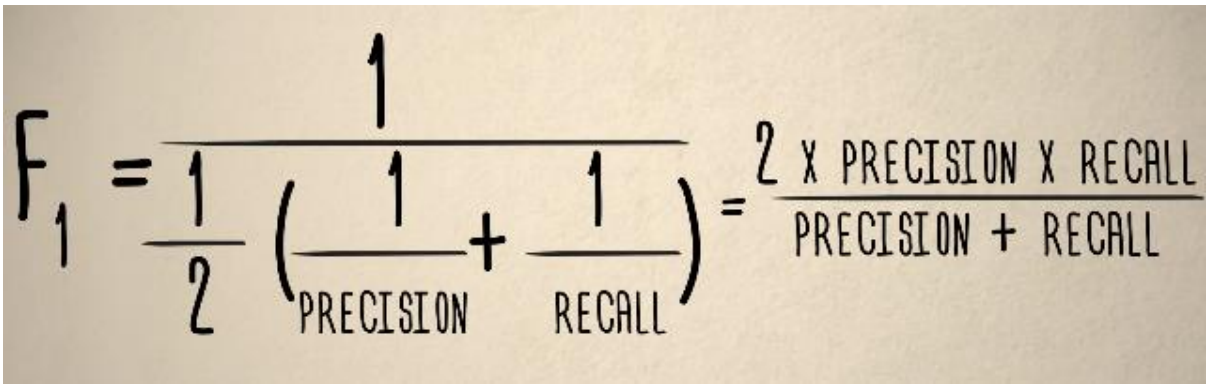


It is often most informative to look at both of these numbers separately, since they measure different properties of the classifier. However, if you want a single number to evaluate the quality of a classifier, there's a statistic called the F-score, or sometimes called the F1 score, that combines the two.

Our first attempt at combining the two numbers would be to just take their average. However, this often isn't quite what we want. In particular, suppose we are in the case where the classifier labels almost everything as positive. Here, the recall would be great, that is very close to one, but the precision would be quite small, and thus close to zero. This would probably not be a very good classifier but averaging the two numbers produces something close to a half.

Instead, we would like a way of combining the two numbers that puts more emphasis on the smaller value. A good choice for this turns out to be what's known as the harmonic mean.

The harmonic mean of two numbers, x and y, is given by 1 over the average of 1/x and 1/y. This will always be in between x and y, but it will be small if either of the two numbers is small. So it's closer to the smaller of the two numbers, in the arithmetic means.



The F-score is then defined to be the harmonic mean of the precision and the recall. So it equals 1 over the average of 1 over the precision, and 1 over the recall. Simplifying this gives a slightly nicer expression. F1 = 2 x precision x recall over precision + recall.

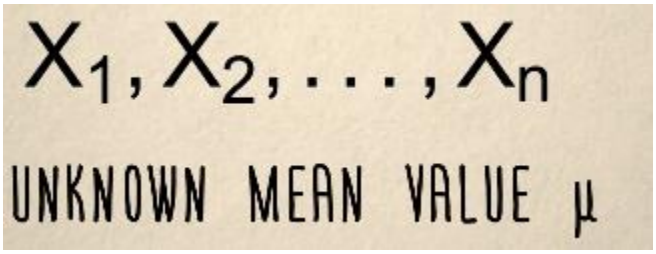
So in summary, we now have a couple of measures for evaluating how good a job a classifier does. We defined precision and recall, two numbers that capture different facets of the quality of classifier. And then we showed how to combine them into an F1 score. This gives us a set of statistics we can use so that later we can decide, when we have some classifier, how good a job it's doing.

3.3 Hypothesis Testing

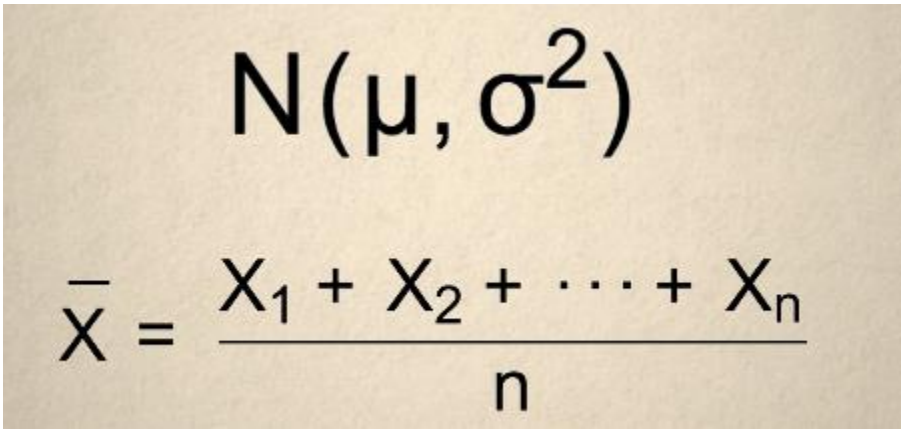
The meaning of the term hypothesis testing is precisely what it says. Using a data observed from a distribution with unknown parameters, we hypothesize that the parameters of this distribution take particular value and test the validity of this hypothesis using statistical methods.

3.4 Confidence Intervals

The confidence intervals are intended to provide us with probabilistic level of certainty about our conclusions, regarding parameters of some probability distribution.



Let us introduce it somewhat more formally. Suppose you have several observations, X1, X2, and so on, Xn from the probability distribution. Suppose you have a prior knowledge that this is a normal distribution, but with some unknown mean value, mu.



Supposed at the same time you happen to know the standard deviation, sigma, for this normal distribution. This is not as unreasonable as it may sound. As you may recall, we denote such distribution as follows.

Our goal is to estimate mu, a natural estimation for mu is simply the average of our observations, which we denote by x bar, like this. Naturally, the actual value of X bar, practically, will never equal to the mean value mu. Our hope is, though, that the estimation is not too far.
But, how far is too far? This is where the concept of confidence interval helps. Let us use the property of the normal distribution, which says, that in fact, X bar itself

also has a normal distribution, with the same mean value, mu, and standard deviation, equal to sigma over square root of n. Namely, the standard deviation of X bar is this.

$$\frac{\sigma}{\sqrt{n}}$$

Using our notational convention, X bar then has the following distribution. It is very important to understand, that while X bar is intended to estimate the unknown mean value, mu, it is actually a random variable with this distribution.

$$N\left(\mu, \frac{\sigma^2}{n}\right)$$

For such a random variable, we can compute the probability that it falls in a certain range. For example, we can compute the probability that it does not exceed the actual, but unknown to us, mean value of mu by, say, two units.

$$P(\bar{X} \leq \mu + 2)$$

Namely, our estimation is now two units greater than the actual mean value.

And now we rewrite it like this, and now we rewrite it further, in the following, admittedly, weird way.

$$P(\bar{X} \leq \mu + 2)$$

$$P(\bar{X} - \mu \leq 2)$$

$$P\left(\frac{\bar{X}-\mu}{\sigma/\sqrt{n}} \leq \frac{2}{\sigma/\sqrt{n}}\right)$$

$$\frac{\bar{X}-\mu}{\sigma/\sqrt{n}} \qquad N(0,1)$$

What we have on the left-hand side, namely this expression, is actually a standard normal random variable. That is, it is the normal random variable, with mean 0 and standard deviation equal to unit.

Let us do an example. Say you have collected a data set consisting of 40 observations. In this case, the sample size, n, is 40. Suppose you happen to know that the standard deviation of this distribution, from which the sample is generated, is 6.8.

$$N\left(\mu, \frac{\sigma^2}{n}\right) \qquad \frac{2}{\sigma/\sqrt{n}}$$

$$n = 40 \qquad \sigma = 6.8$$

From this, we can immediately find that, from the standard normal distribution table, we'll find that the probability that the standard random variable does not exceed this value, is approximately 0.968. You have computed the average of these 40 observations, and let us say you found it to be 4.7.

$$\frac{2}{\sigma/\sqrt{n}} = \frac{2}{6.8/\sqrt{40}} = 1.8602$$

0.968 $\quad \bar{X} = 4.7$

Then according to our derivation, the true, but unknown, mean value mu, is at least 4.7 minus two, which is 2.7, with probability 0.968. We also say that our confidence level of this statement is 0.968, or roughly 96%.

P(Mean value μ is at least 2.7) = 0.968

In this example, we have assumed the knowledge of the standard deviation sigma. Usually in practice, we do not have it, but we can still estimate it from data using the following expression, which we denote by sigma bar.

$$\bar{\sigma} = \sqrt{\frac{(X_1 - \bar{X})^2 + (X_2 - \bar{X})^2 + \cdots + (X_n - \bar{X})^2}{n - 1}}$$

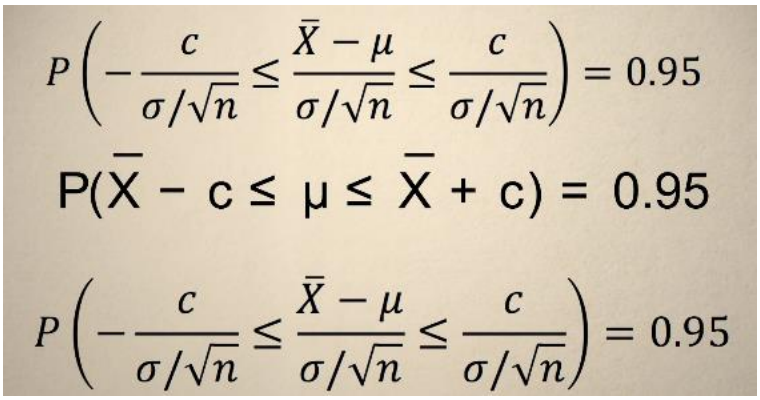
Here, X1, X2, and so on, Xn are your observations. Just like X bar is perhaps the most reasonable estimation of the actual, but unknown, mean value mu. This formula gives perhaps the most reasonable estimation for true, but unknowns standard deviation value.

SHOULDN'T WE HAVE CONFIDENCE INTERVALS FOR $\bar{\sigma}$ FIRST BEFORE WE USE IT FOR OUR COMPUTATION OF CONFIDENCE INTERVAL FOR THE MEAN VALUE μ?

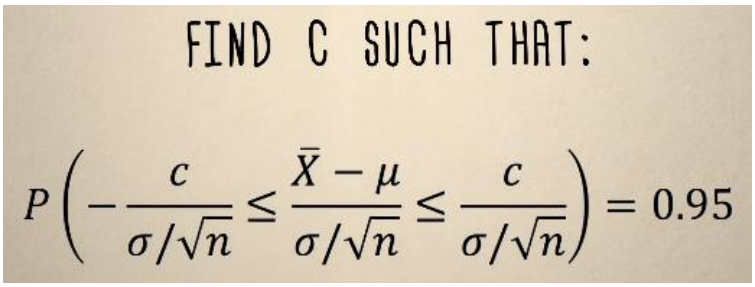
WHEN SAMPLE SIZES ARE LARGE, ESTIMATION OF $\bar{\sigma}$ IS MORE ACCURATE

The answer is that, when the sample sizes are reasonably large, usually at least 30, the estimation for sigma bar is far more accurate than the level of error we will encounter for mu. So, this substitution is a reasonable one. There is a deeper theory behind this, which relies on the so-called, t-distribution, which we'll skip today.

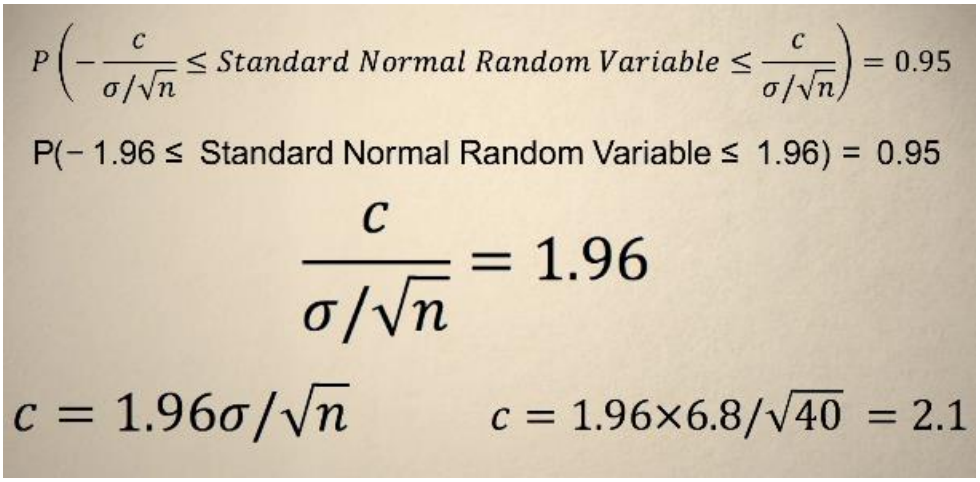
In the example above, we have obtained an estimation from mu, which is only one sided, namely, it is at least 2.7. Ideally, we would like to have an estimated from above and from below. Additionally, often we want to have a target level of confidence level, such as 0.95 or 0.99, rather then whatever comes out here, like 0.968.
This can be done by, sort of, reverse engineering our interval, so that the resulting certainty equals the target we need. Say we want to have confidence level of 0.95.



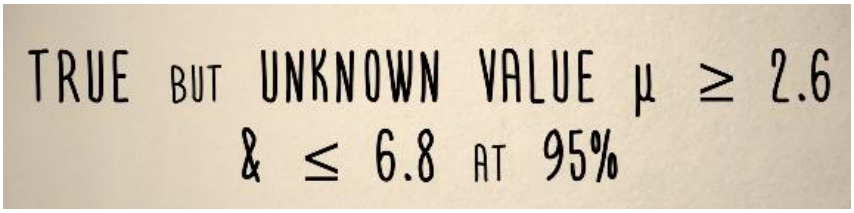
So our goal is to find a value c, such that the following is true. We can rewrite this equivalently as follows. And, if you find the right value for c, for this to be the case, our 95% confidence interval would be of the form X minus c, to X plus c.



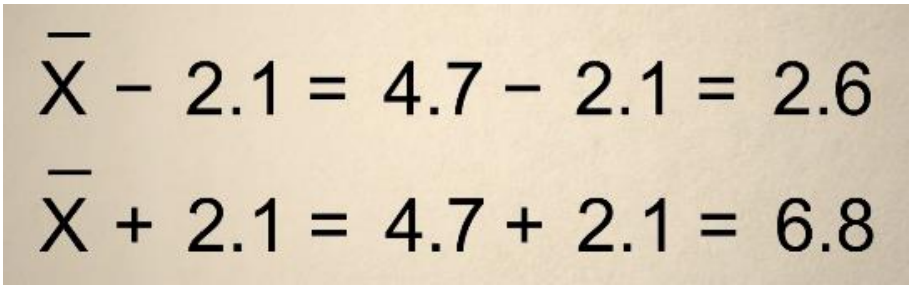
Now saying that this relationship is true, is the same as the following being true, which is the same as the following being true. From the standard normal distribution table, I can find the magic number, 1.96, which satisfies the following property.



We obtain that the following must be the case, from which we find the formula for c as follows. Since we know the standard deviation, which in our example was 6.8, and since we know the sample size, which in our case, is 40.

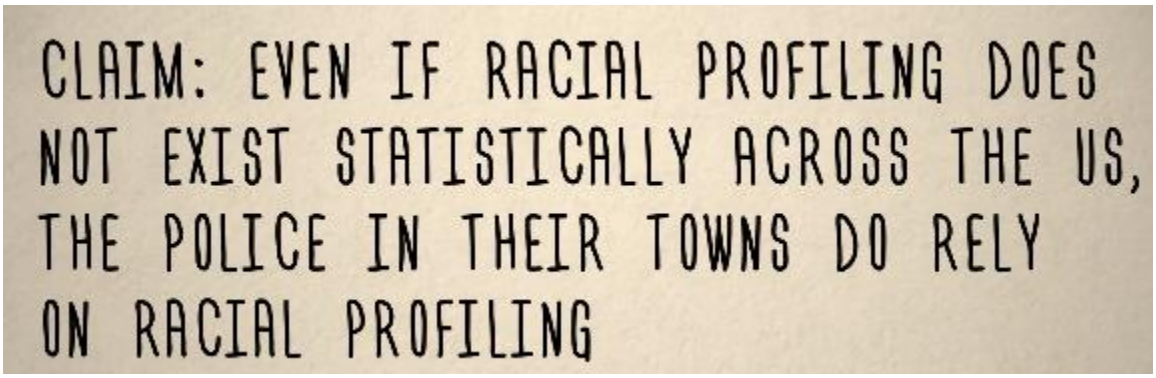


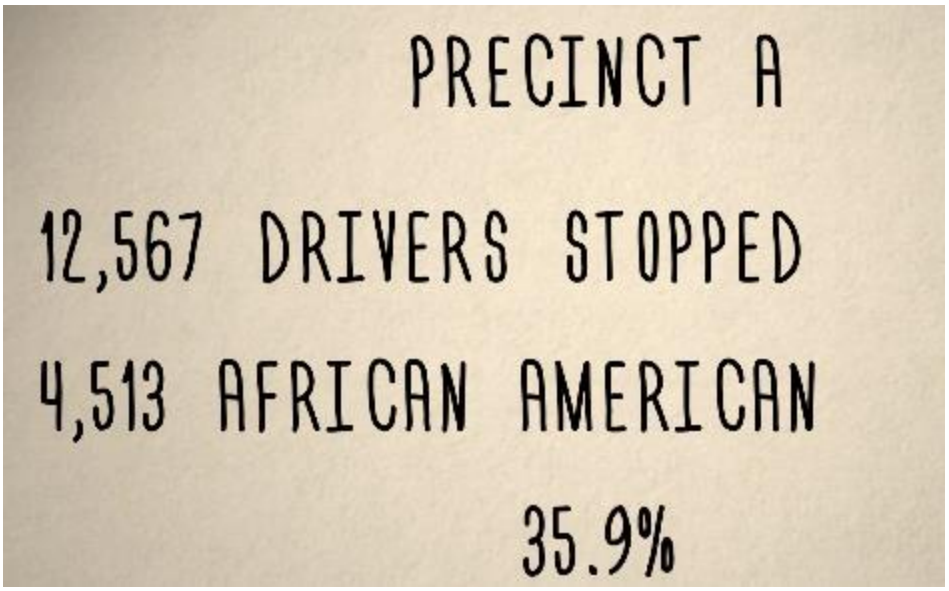
We can compute c as follows, we conclude that the true, but unknown, mean value mu lies within approximately 2.1 units from X bar, which is 4.7 with confidence 0.95, namely 95%. This interval is in fact defined by 2 values, the lower value here, and the upper value here.

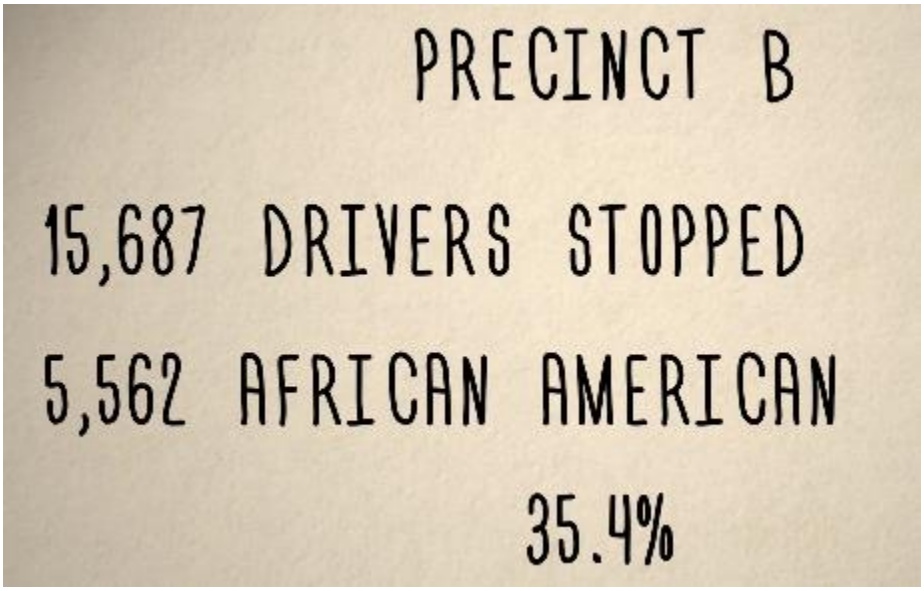


We can summarize our conclusions as follows, the true, but unknown, mean value mu is at least 2.6, and at most 6.8, with 95% confidence. The interval length might appear as too wide. This is perhaps true, but sometimes even a very wide interval can provide an extremely useful information.

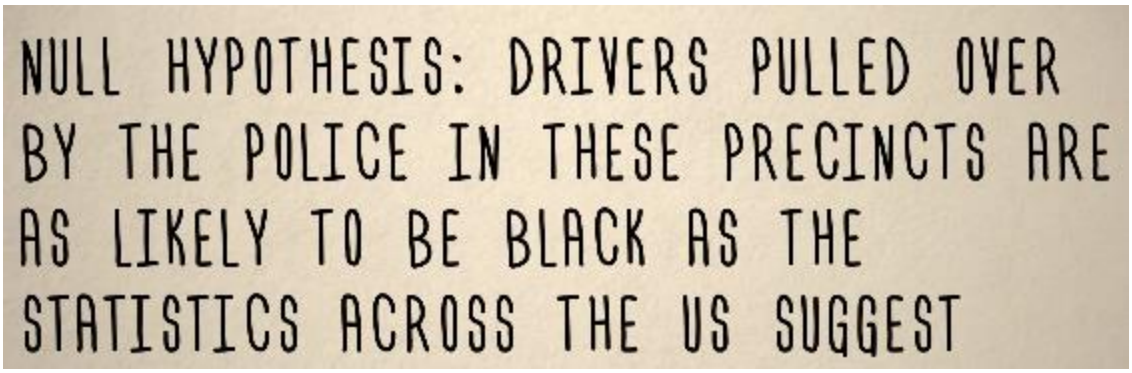
3.5 Hypothesis Testing: Validity of Binomial Distribution



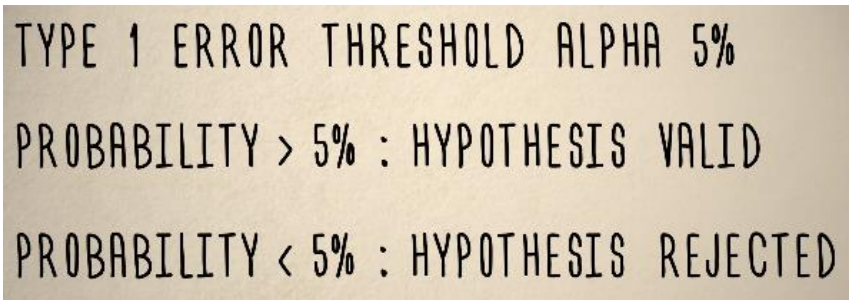




The data shows that during the year of 2015, out of approximately 12,567 drivers stopped by the police, in fact 4,513 were black. 4,513 divided by 12,567 is approximately 0.359, which is larger than 35% and hence the claim of racial profiling. For the second Precinct B among 15,687 drivers stopped 5,562 were black. Again, 5,562 divided by 15,687 is approximately 0.354, which is also larger than 35% and the claim of racial profiling is made again. How can we asses the statistical validity of these claims?



We do this by conducting the hypotheses test.
We take as our null hypothesis, namely the hypothesis to be tested, that the drivers pulled over by police in these precincts are as likely to be black as for the statistics across US suggest. Namely, a driver pulled over by police in these precincts is black with 35% likelihood, as is the national figure.



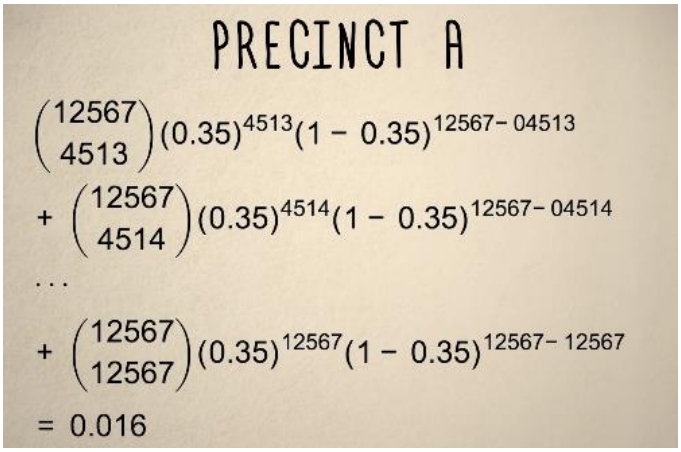
We set Type 1 error threshold alpha to be 5%, this is up to the person conducting the test, of course and different error rates may be considered, but let us accept if for now. Assuming the validity of this hypothesis, we will compute the likelihood that among 12,567 drivers in Precinct A at least 4,513 were black. Namely at least as many as was the number of black drivers actually pulled over.

If this probability is greater than 5%, we accept the hypothesis as valid with 5% error rate and say that the exceedance of 35% rate in this precinct is just a statistical fluctuation.

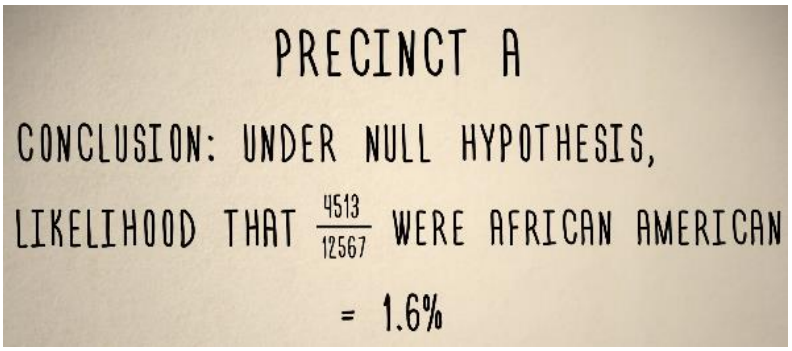
If it is less than 5%, we will reject this hypothesis and say that the number of black drivers actually pulled over was unusually high. We will conduct a similar analysis for Precinct B.

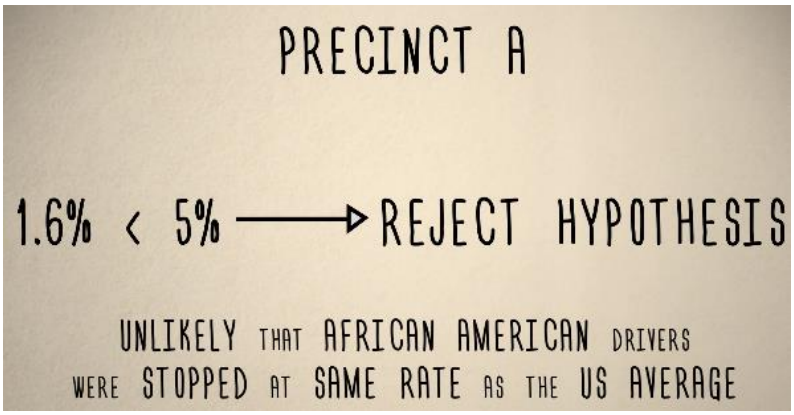
How can we compute these probabilities?

If the 35% hypothesis is true, then according to the binomial distribution, the likelihood that out of 12,567 drivers at least 4,513 were black can be computed using the following formula for binomial distribution as follows.

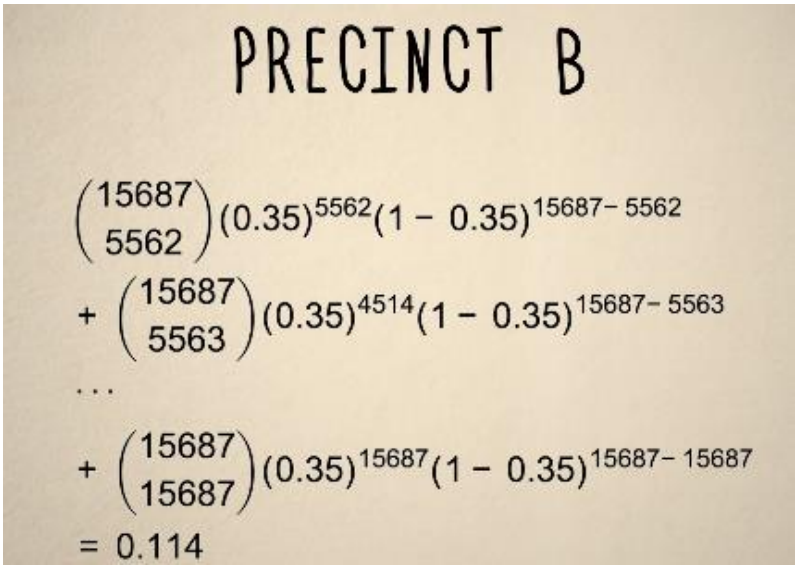


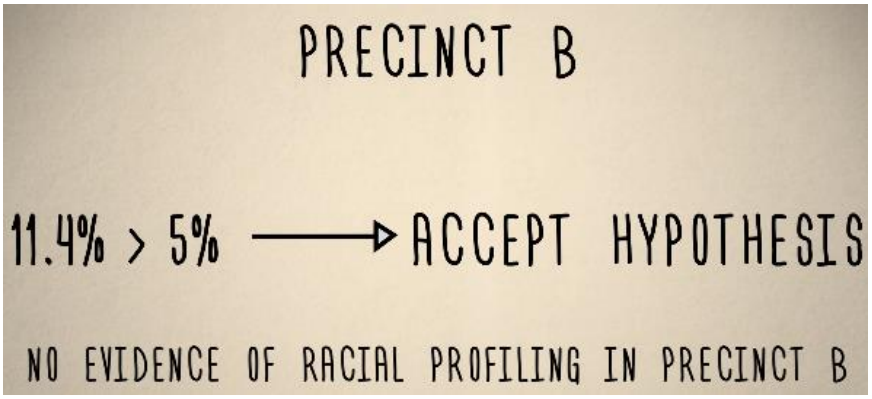
The answer in this case is 0.016 or about 1.6%. We conclude that under the null hypothesis, no profiling. The likelihood that out of 12,567 stopped drivers 4,513 were black is approximately only 1.6%. Since this is below 5% selected for Type 1 error tolerance, we must reject the hypothesis.



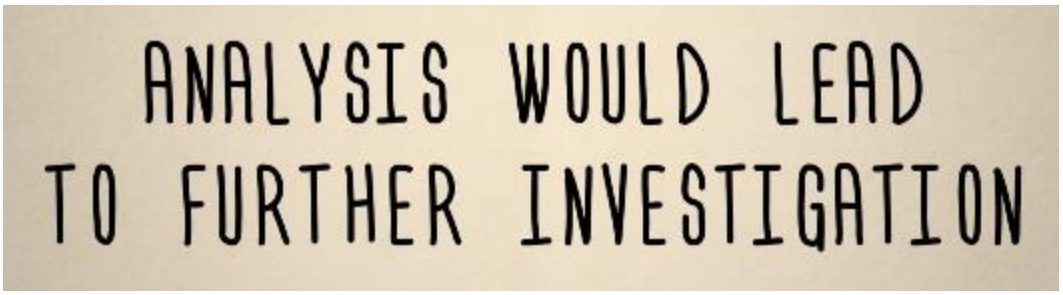


Thus, according to the data, it is unlikely, only 1.6% chance, that the black drivers were stopped at the same rate as in the US on average. This of course does not legally prove the presence of racial profiling, just indicates the statistical evidence for it.





What about Precinct B? Using a similar analysis the likelihood that out of 15,687 stopped drivers at least 5,562 are black is given by the following expression, which is evaluated to be approximately 11.4%. Since 11.4% is higher than 5%, we must accept the null hypothesis. Namely, according to the statistical analysis, there is no evidence of racial profiling in Precinct B.



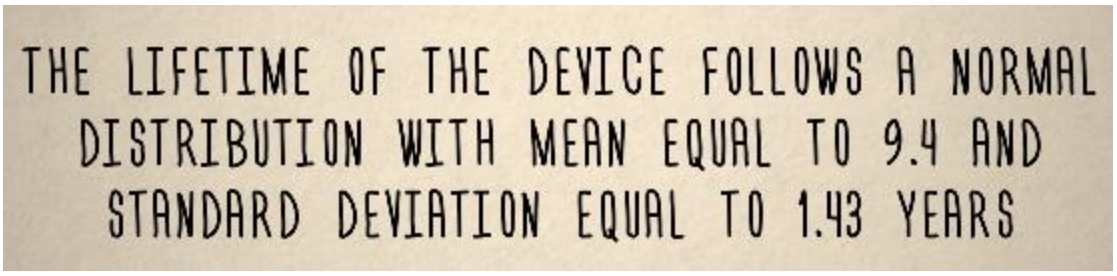
In the reality, such an analysis might suggest a further investigation where profiling appears to take place according to the statistical evidence. As far as the analysis can either confirm the presence of profiling or not find any evidence of it. The statistical tools can simply assist in the identifying, relatively quickly, precincts where there is some statistical evidence of profiling.

From the methodological perspective, we just gave an example of testing your hypothesis regarding the validity of a certain distributional assumption, binomial distribution being the case. One can naturally apply it to other types of distributions, such as uniform, geometric, Poisson distribution and many others. The method is very general.

3.6 Misuses of Statistics

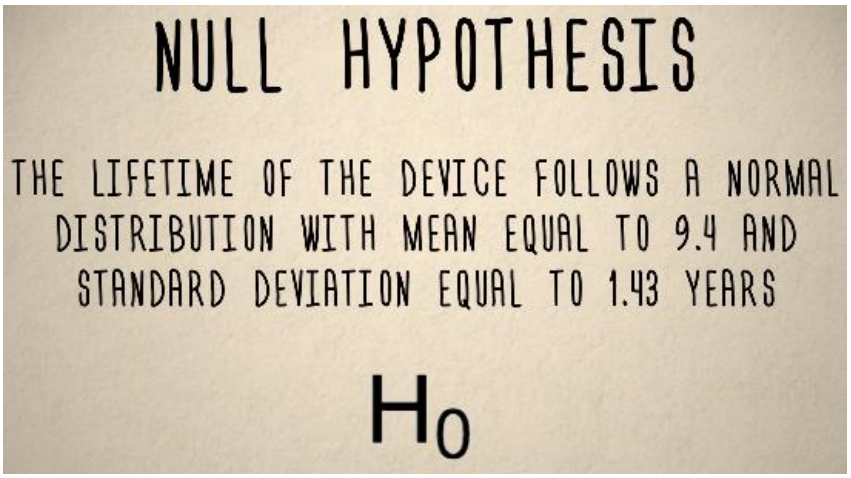
Let us say you are managing a manufacturing firm which produces a certain electronic device. And the industry standard for the lifetime of this device is to have let's say, lifetime of at least nine life years. Your engineer claims the device produced by your company fits the standard and in fact has average lifetime of 9.4 years. And a standard deviation of 1.43 years. To support his claim he gives you sample of lifetimes for 50 devices with various actual lifetimes.

You have computed the average of the projected lifetimes for the sample provided by your engineer, and found it to be 9.6 years, which is above the industry standard of 9 years, which is good. Your goal is to test the following hypothesis. The lifetime of the device follows a normal distribution with mean equal to 9.4 and standard deviation equal to 1.43 years.

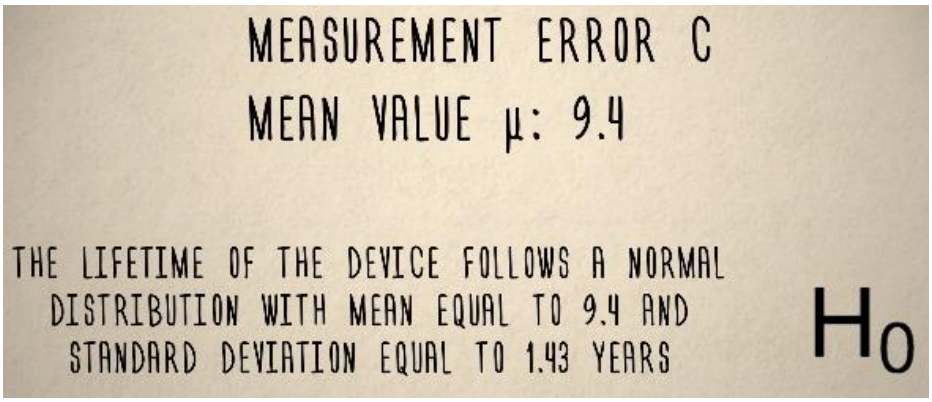


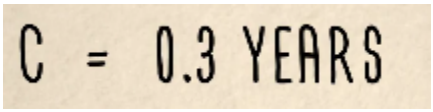
Since the mean value in this hypothesis is 9.4 which is greater than 9, confirming this hypothesis would be good news as far as meeting the industry standard is concerned. So, our hypothesis is as follows. How can you test this hypothesis?

When you introduce and test some hypothesis, a convention is to call it a null hypothesis, and denote it as follows.

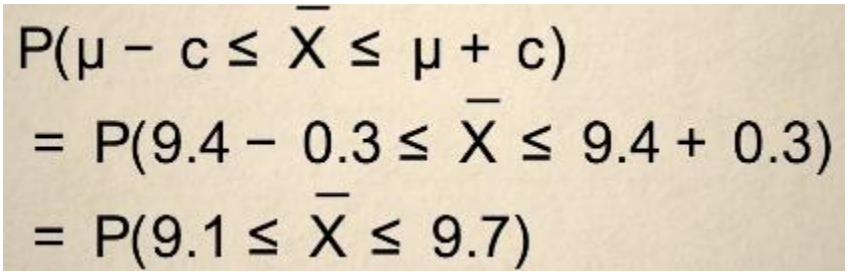


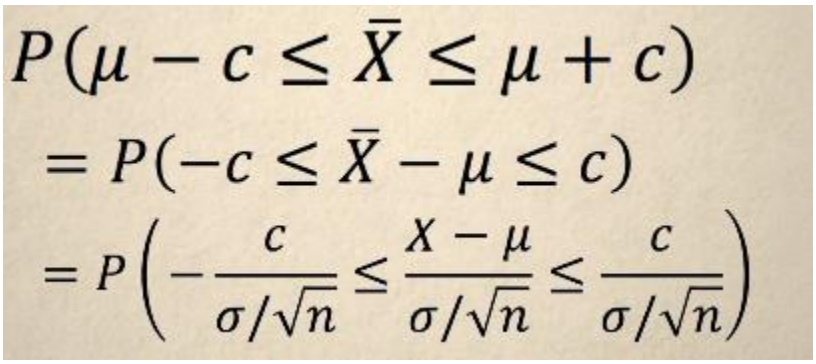
Now let's do actual testing. This is done by first assuming that the hypothesis is true, selecting a certain measurement error value C. And computing the probability that the outcome, namely the average of these 50 observations falls within C units of the claimed value of the mean value of mu. Which in our case, was claimed to be 9.4. If we choose our measurement error to be, say, 0.3 lifetime years, then we need to conclude the following probability.





How can we compute this probability? Well again, we can use the trick of manipulating x bar in the standard, normal random variable as follows. As in our discussion of confidence intervals, the random variable x bar minus mu over sigma divided by square root of n, is actually a standard normal random variable.





How can we compute this probability? Well again, we can use the trick of manipulating x bar in the standard, normal random variable as follows. As in our discussion of confidence intervals, the random variable x bar minus mu over sigma divided by square root of n, is actually a standard normal random variable.

$$P(\mu - c \leq \bar{X} \leq \mu + c)$$
$$= P(-c \leq \bar{X} - \mu \leq c)$$
$$= P\left(-\frac{c}{\sigma/\sqrt{n}} \leq \frac{X - \mu}{\sigma/\sqrt{n}} \leq \frac{c}{\sigma/\sqrt{n}}\right)$$

$$\frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$$

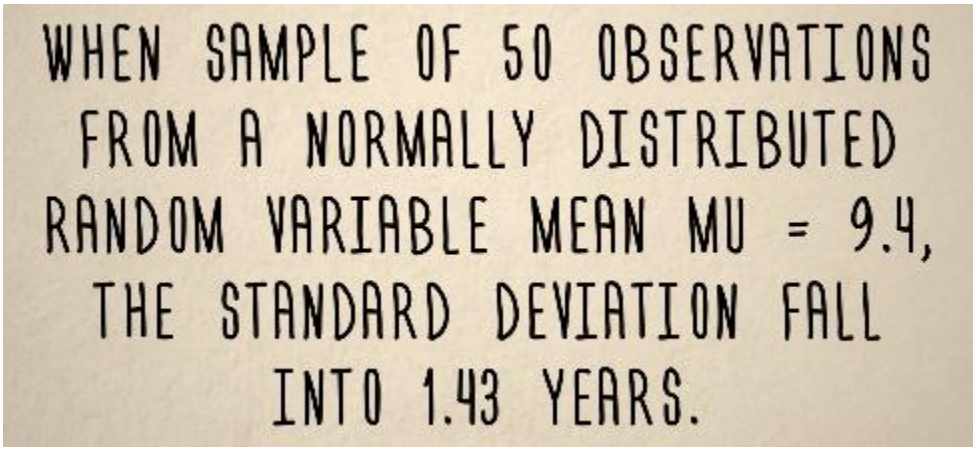
$$P(\mu - c \leq \bar{X} \leq \mu + c)$$
$$= P(-c \leq \bar{X} - \mu \leq c)$$
$$= P\left(-\frac{c}{\sigma/\sqrt{n}} \leq \frac{X - \mu}{\sigma/\sqrt{n}} \leq \frac{c}{\sigma/\sqrt{n}}\right)$$
$$P\left(-\frac{c}{\sigma/\sqrt{n}} \leq Z \leq \frac{c}{\sigma/\sqrt{n}}\right)$$

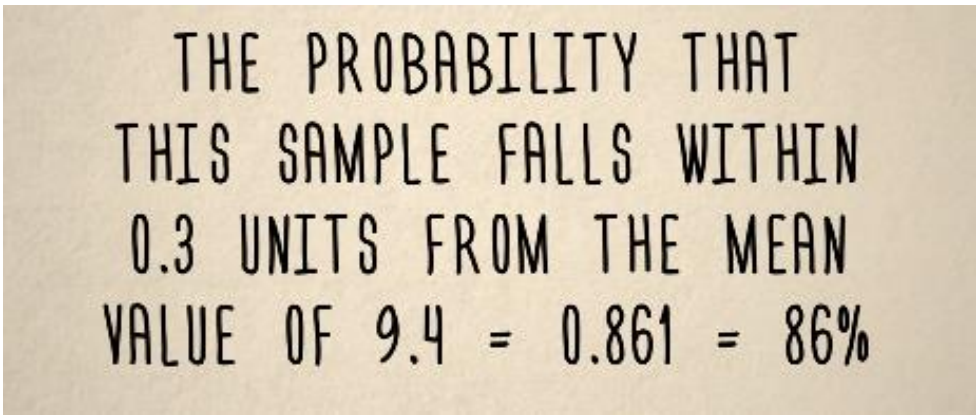
Let's denote it by Z for convenience. Then we have to compute the following. And we have everything we need to compute this probability. We have n, which in our case is 50. We have sigma, which in our case is claimed to be 1.43. And we have c, which we have chosen to be 0.3. So we need to compute the following probability which we can find in the table for standard normal distribution.

$$P\left(-\frac{0.3}{1.43/\sqrt{50}} \leq Z \leq \frac{0.3}{1.43/\sqrt{50}}\right) = P(-1.48 \leq Z \leq 1.48)$$

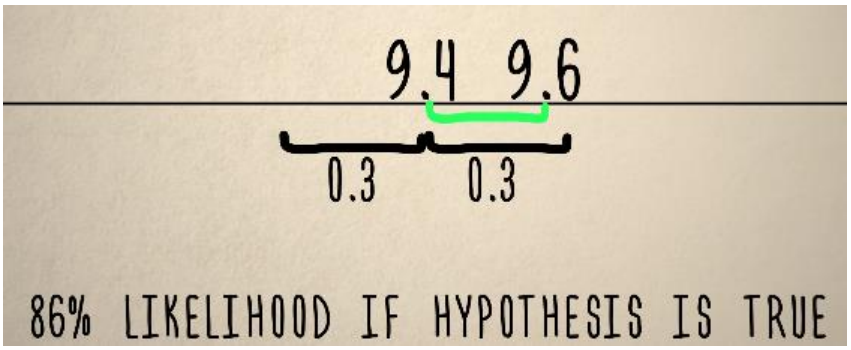
$$P\left(-\frac{0.3}{1.43/\sqrt{50}} \leq Z \leq \frac{0.3}{1.43/\sqrt{50}}\right) = P(-1.48 \leq Z \leq 1.48) = 0.861$$

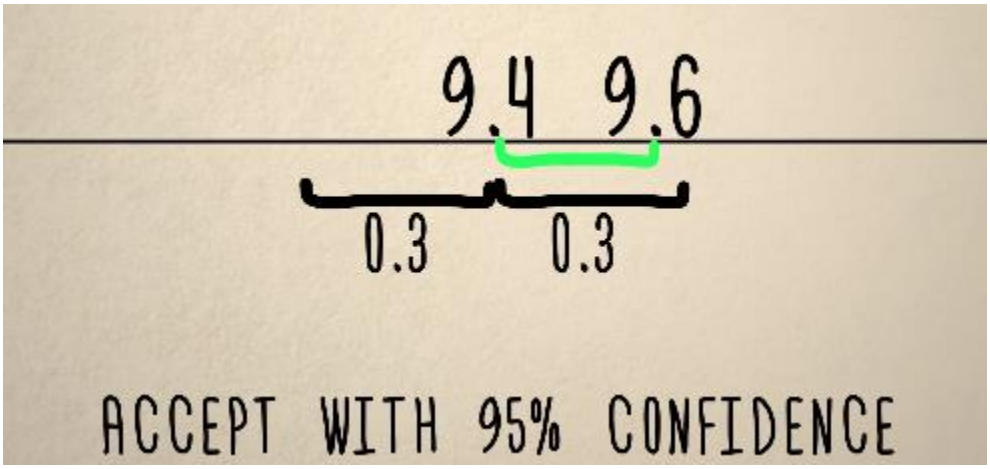
It is approximately 0.861 or roughly 86%.
Let's summarize what we found. We have computed that when a sample of 50 observations from normally distributed random variable with mean value equal to 9.4 years, and standard deviation equal to 1.43 years is drawn. The probability that this sample falls within 0.3 units from the mean value of 9.4 equals 0.861, namely about 86%.



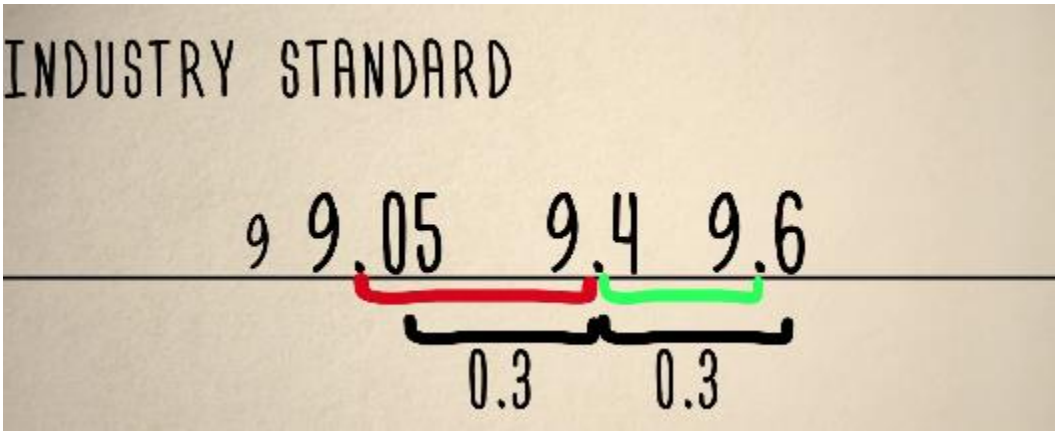


Wait a minute. Where did we use in this computation that our actual observed average of 50 estimated lifetimes was 9.6? This is the where we can use it to test the validity of the hypothesis. Falling within 0.3 units of the claimed mean value of 9.4 occurs with 86% likelihood if the hypothesis is true.





Our observed average is 9.6, which is in fact within 0.2 units from the claimed mean value of 9.4. Since 0.2 is in fact smaller than 0.3, we'll say that we accept this hypothesis and we do so with 95% confidence.



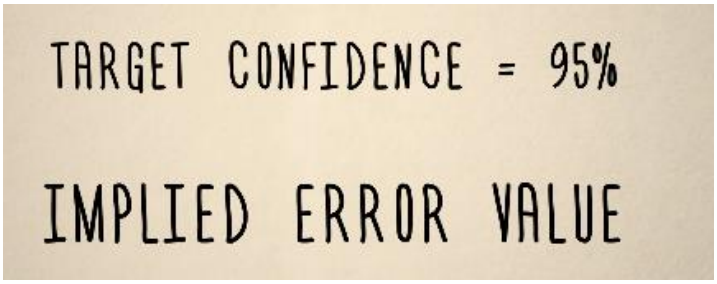
If you or your engineer collected a different sample with, say, average 9.05, and would want to use this average to test the same hypothesis, you would do the same thing. Compute the difference of 9.05 and the claimed mean value of 9.4, which is

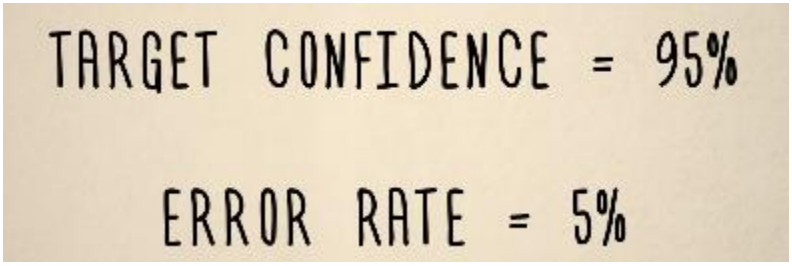
negative 0.35. Since negative 0.35 in absolute terms is larger than 0.3, the hypothesis should be rejected.

Notice that you have rejected the hypothesis even though your observed average was 9.05, which is higher than the industry standard of 9 years. You still reject the hypothesis because the hypothesis was about the mean of the distribution, and not meeting the industry standard per se. There's something unnerving about this method. Whether we accept or reject the hypothesis depends on the data we observe.

For different samples, we may either accept or reject the same hypothesis. But the samples come from data and this is the only thing we have access to. We can only draw our conclusion from observed data. There is no crystal ball to tell us whether the claimed value of 9.4 years of lifetime is correct or not. In fact, there is no such thing as the correct mean value.

There is something else which is perhaps annoying in this example. The confidence level in our hypothesis testing example was computed to be approximately 86%. This was an implication of choosing somewhat arbitrarily the error value of 0.3 years for the lifetime duration of the device. Well this is something we can deal with. We can choose the target level of confidence say 95% and compute the implied error value. This is very similar to what we have encountered in our discussion of confidence levels.





$$\alpha = 0.05$$

To achieve the target value of 95%, Namely the error rate of 5%. Let us first denote this error rate by alpha. Now we can compute the implied error value c as follows. From the table, as we recall, there is a magic number of 1.96 for the following idea. We conclude that the following is true. Looks familiar? It should. In our case, we use the fact that sigma is hypothesized to be 1.43 and the sample size is 50, and find that the value of c is 0.396 like this.

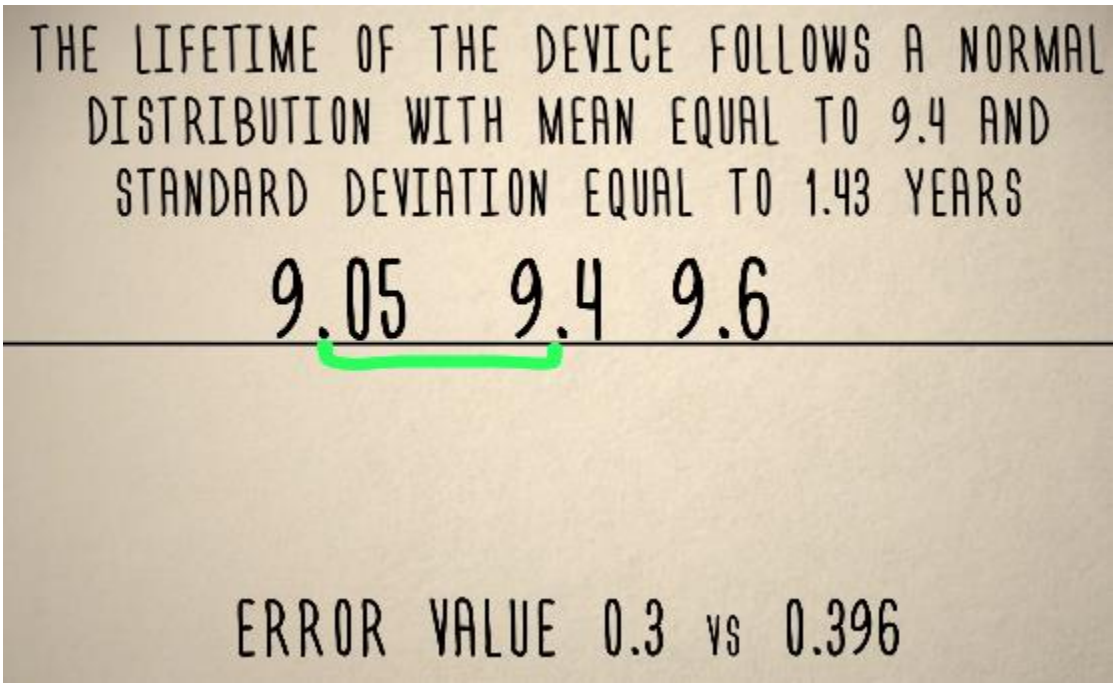
$$P\left(-\frac{c}{\sigma/\sqrt{n}} \leq \textit{Standard Normal Random Variable} \leq \frac{c}{\sigma/\sqrt{n}}\right) = 0.95$$

$$P(-1.96 \leq \text{Standard Normal Random Variable} \leq 1.96) = 0.95$$

$$c = 1.96\sigma/\sqrt{n} \qquad c = 1.96 \times 1.43/\sqrt{50} = 0.396$$

ACCEPT HYPOTHESIS WITH 95% CONFIDENCE LEVEL, IF OUR OBSERVED SAMPLE FALLS WITHIN 0.396 UNITS

We summarize this as follows. We accept our hypothesis of mean value of 9.4, and standard deviation 1.43 with 95% confidence level if our observed sample average falls within 0.396 units from the claimed value of 9.4.



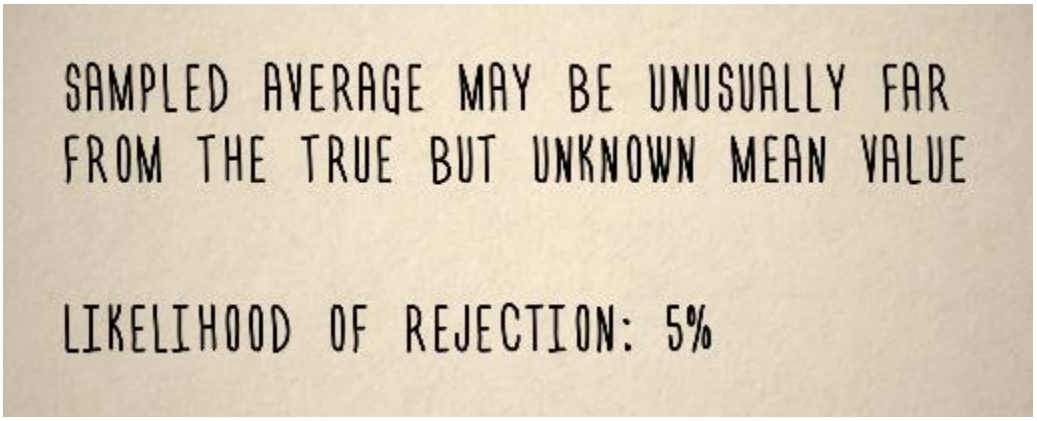
Recall that the sample brought to you by your engineer had average value of 9.6 units, which is within 0.2 units of the claimed mean value. 0.2 is smaller than 0.396 so the hypothesis should be accepted. If the observed average was 9.05, as in our second example, then since the difference of 9.05 and 9.4, which is negative 0.35 is still smaller than 0.396, the hypothesis should still be accepted. It was though, rejected before.

What has changed? You've guessed it. The error value of 0.3 chosen by us was more stringent that the error value of 0.396 implied by 95% confidence level. Thus, there is something else here which is perhaps unnerving in the discussion above. The same hypothesis with the same data and thus the same average value might be accepted or rejected depending on perhaps somewhat arbitrarily chosen confidence level. But maybe this is not so bad since it allows for choice of confidence level in different areas in different industries.

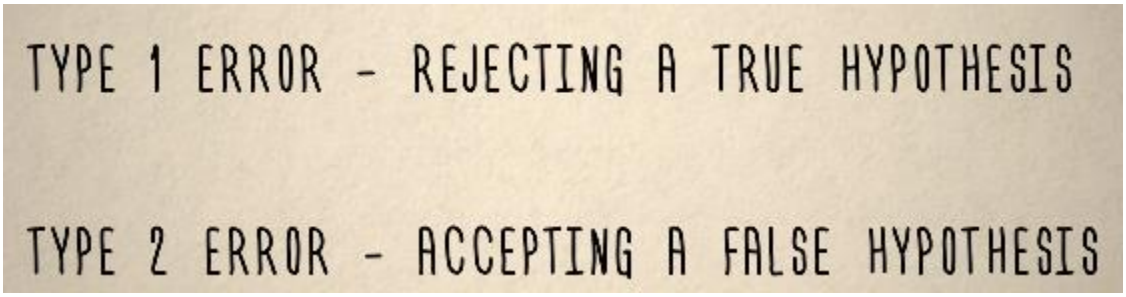
For example, when dealing with drug manufacturing or anything else which relates to human lives and well being, for example airline safety. One might want to desire a very high level of confidence of let's say, 99% or even 99.9%.

We have mentioned earlier the convention of calling the hypothesis, which is being tested, a null hypothesis. It is time to discuss another related concept called type 1 error, which has implicitly already surfaced in our discussion. Note than even when the hypothesis is true, say in our example the mean value was indeed 9.4 and

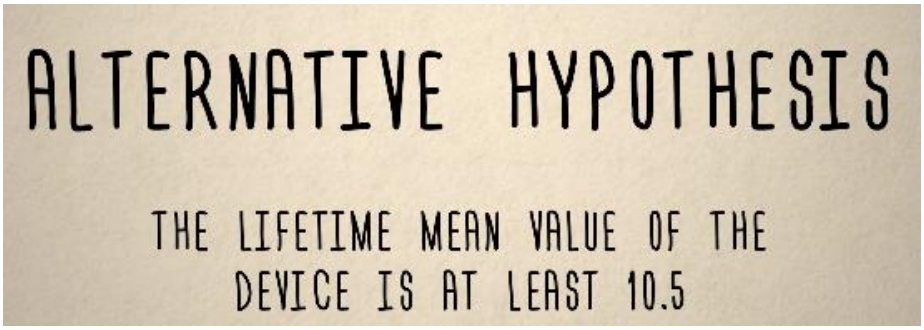
standard deviation was indeed 1.43. There is a chance that the hypothesis is rejected simply because the sampled average was unusually far from the true but unknown to us mean value of 9.4.



In fact, we have set up our testing framework in such a way that the likelihood of this rejection of the true hypothesis equals 5%. The error of rejecting a true hypothesis is called type 1 error, which in our case is 5%.



Type 2 error corresponds to the case of accepting by mistake a hypothesis when it is wrong.
For example, accepting the hypothesis that the mean lifetime of the device 9.4, when in fact, it is not. To estimate the probability for such an error, one naturally has to formulate the meaning of something like, 9.4 is not the right mean.

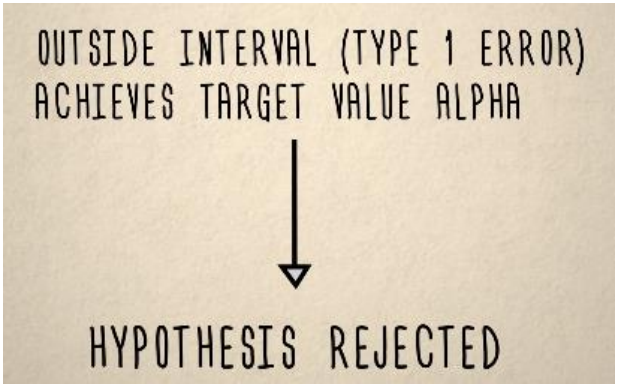


H_a

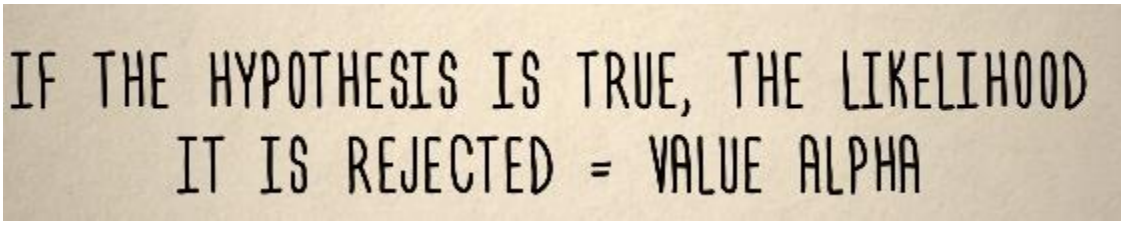
Namely, one has to formulate what is called an alternative hypothesis, for example something like this. The lifetime mean value of the device is at least 10.5. To contrast it with null hypothesis, such a hypothesis is called alternative hypothesis

3.7 P-Value

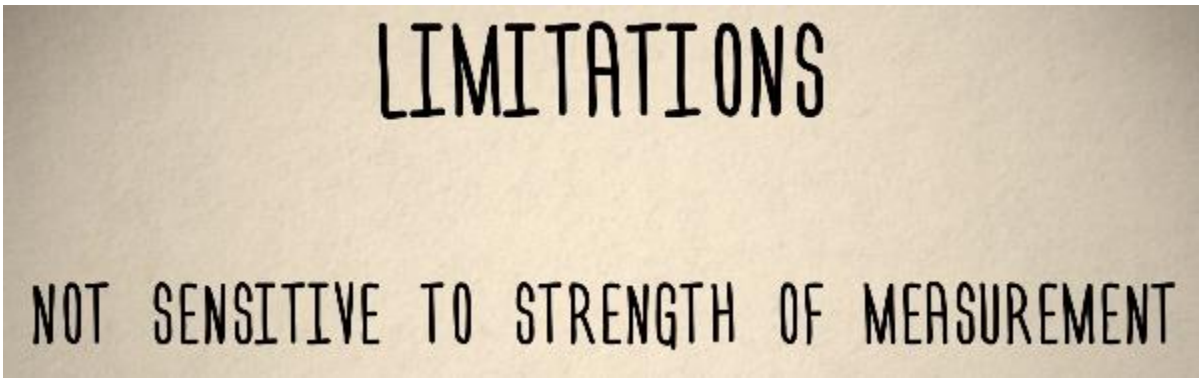
In discussing the hypothesis testing method, we have introduced in particular, a method for accepting or rejecting a hypothesis which is selected to be the null hypothesis. The method was based on introducing some measurement of the observed sample such as its sample mean and computing the probability that this measurement falls within some declared region such as for example, an interval around the mean which is claimed by the null hypothesis.



The life of the interval was engineered so that the probability of falling outside of this interval, which as you may recall is called type 1 error, achieves the target value alpha.
Usually taken to be, for example, 5% or 1%. In a case that this measurement indeed falls outside of the interval, the hypothesis is rejected otherwise, it is accepted.

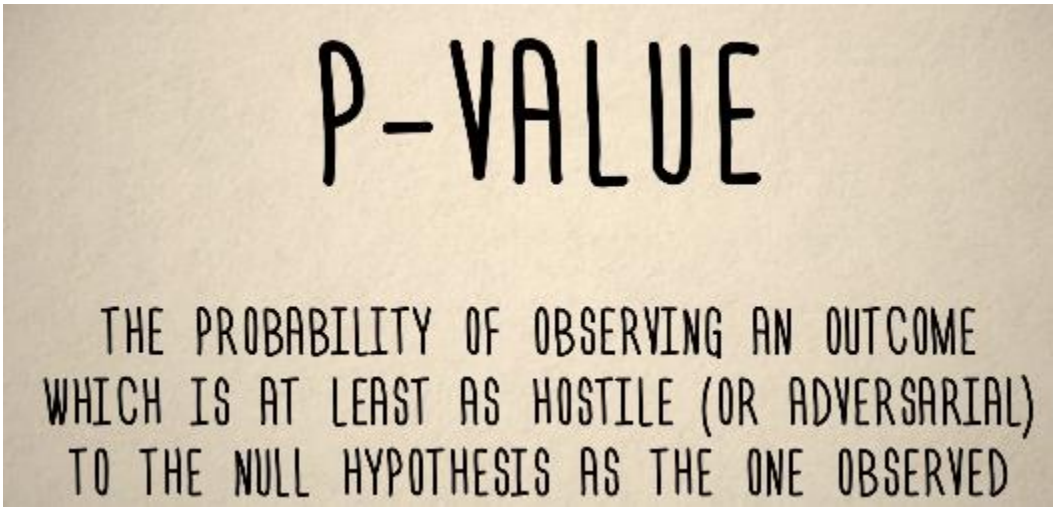


Thus, if the hypothesis is true the likelihood it is rejected equals value alpha. There is a certain limitation of this method. This mechanism provides an acceptance rejection rule for the hypothesis, but does not provide any sense of the strengths of the measurement in either supporting or rejecting the hypothesis.

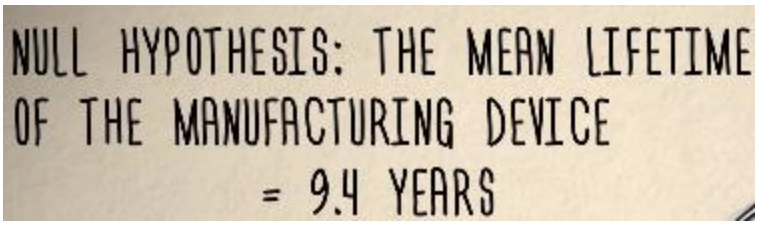


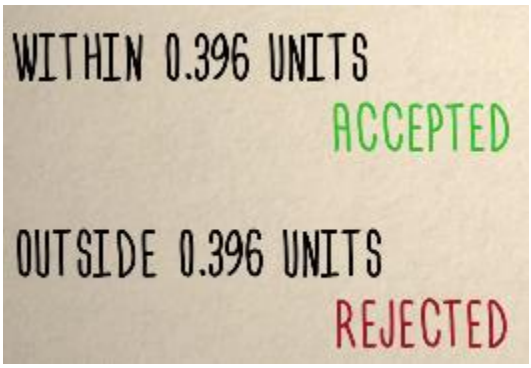
If we were to have two measurements of the sample and say based both of them the hypothesis should have been rejected, which one has stronger evidence for rejection?

To gauge the strength of evidence for either supporting or rejecting a hypothesis we introduce now a new and important concept of p-value.

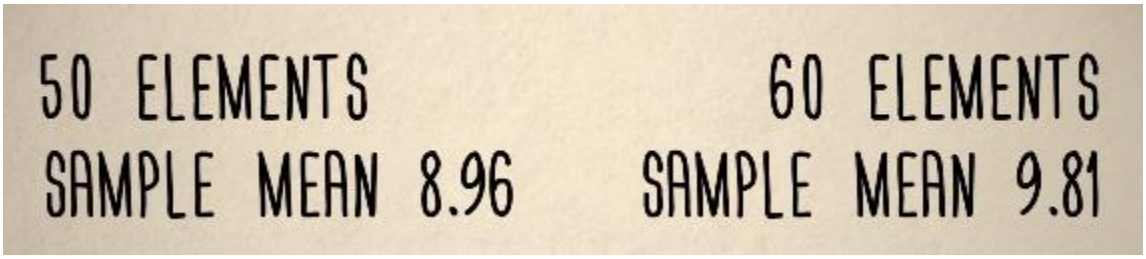


A formal definition of the p-value sounds something like this. The p-value is the probability of observing an outcome which is at least as hostile or adversarial to the null hypothesis as the one observed.
Did you get this? Probably not.

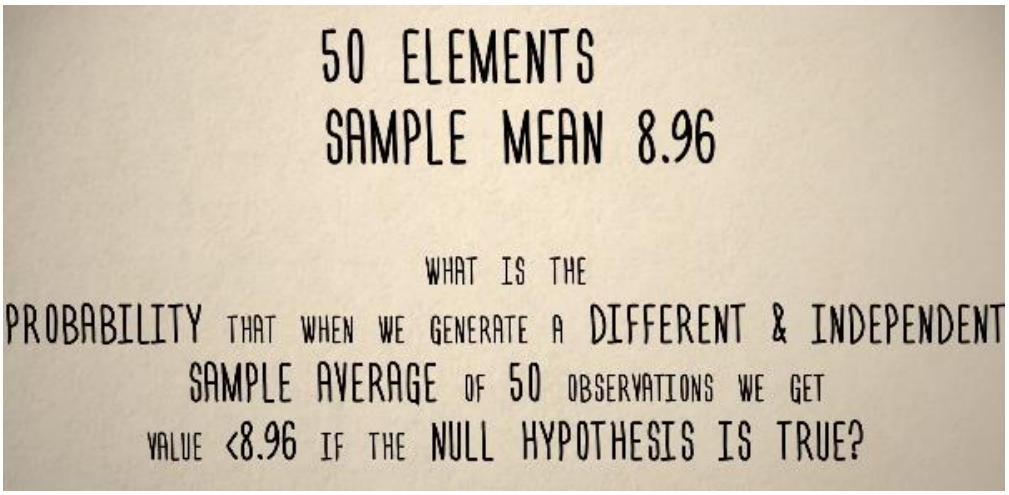




So let us discuss an example. Let us recall our example of a manufacturing device lifetime. Recall, that the null hypothesis in this particular example stated that the mean lifetime of the manufacturing device is 9.4 years. Then we computed a magic number 0.396 which had the property that if the computed sample mean is within 0.396 lifetime units from 9.4, then the hypothesis is accepted. Otherwise, it is rejected.

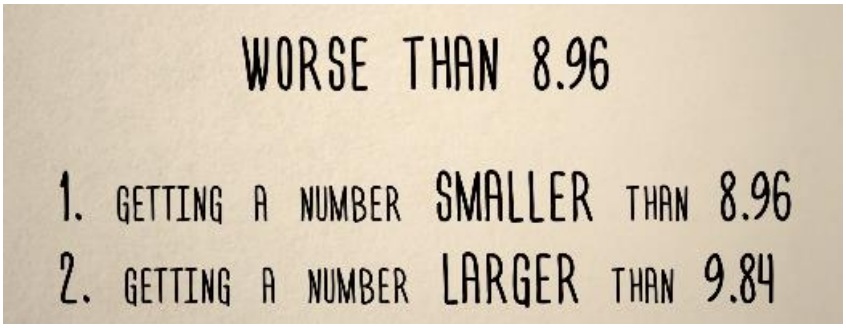


Suppose you have actually gathered two samples, one with 50 elements with a sample mean equal to 8.96, and another one with 60 elements, with sample mean equal to 9.81. First note that according to each of this measurement the null hypothesis should be rejected. Each of this measurement will be associated with some p-value which we now describe how to compute.



Take first 8.96 sample average. What is the probability that when we generate a different and independent sample average of 50 observations we get the value less than 8.96 if the null hypothesis is true. Before we answer this let us discuss why we want to compute something like this?

We want to have a sense of how unlikely it is to obtain a sample average of 8.96. Now the probability for observing exactly 8.96 is practically zero. So instead, we're asking to compute the likelihood of observing something even worse than 8.96. And this is our approximate measurement of the unlikelihood of seeing the number as far from the claim mean of 9.4 as 8.96.



Now worse can mean two things. One is natural getting a number smaller than 8.96. But there is a complimentary worse, the difference between 8.96 and 9.4 is negative 0.44. Then getting a sample of 9.4 plus 0.44, namely 9.84 is as bad as the one of 8.96. So getting a measurement above 9.84 in this sense is also worse than getting the number 8.96. Now let us compute the probability of getting a measurement either below 8.96 or above 9.84.

$$\mathbb{P}\left(Z \leq -\frac{0.44}{1.43/\sqrt{50}}\right) + \mathbb{P}\left(Z \geq \frac{0.44}{1.43/\sqrt{50}}\right) = 2\mathbb{P}(Z \leq -2.175)$$

I say that this probability is the probability that the standard normal variable which we denote by capital Z here. Takes at most the following value for the sigma equals to 1.43 is a standard deviation of the device under the null hypothesis.

From the standard normal distribution table we find that the probability that the standard normal rounding variable x value below a negative 2.175 is about 0.015 which is about 1.5%. So our answer is twice that much namely 0.03 or 3%. We conclude that the p-value for our sample is 3%.

Inwards, the likelihood of getting a sample which is as far from the hypothesis as the one observed is 3%. Since the p-value of 3% is kind of small, kind of. We will be inclined to reject this hypothesis.

Notice that there is a simple relationship between the p-value and our acceptance, rejection rule say based on the type one error alpha equal to 5%. The hypothesis is accepted if the p-value exceeds 5% and rejected if it stays below 5%.

A similar rule applies if the value of the type one error is said to be 1%. Often the p-value is defined as probability of being worse than observed only in one direction, in this case being below 8.96.

$$\mathbb{P}\left(Z \leq -\frac{0.44}{1.43/\sqrt{50}}\right) = 1.5\%$$

So in our example because the normal distribution is symmetric about its mean, the actual p-value would be only half of the 3% corresponding to this probability namely only 1.5%.

The one-sided version of the p-value is particularly useful when the distribution is not symmetric around its mean as we are about to see in our next example. It is important to be explicit in what the p value stands for when conducting a statistical test.

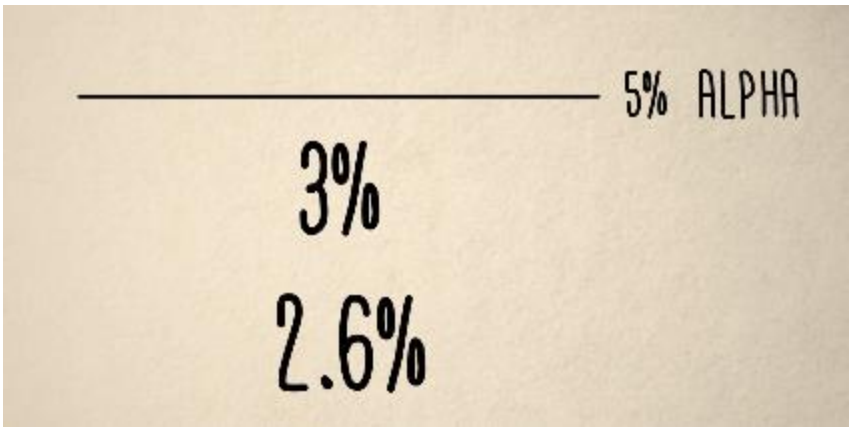
60 ELEMENTS
SAMPLE MEAN 9.81

$9.81 - 9.4 = 0.41$

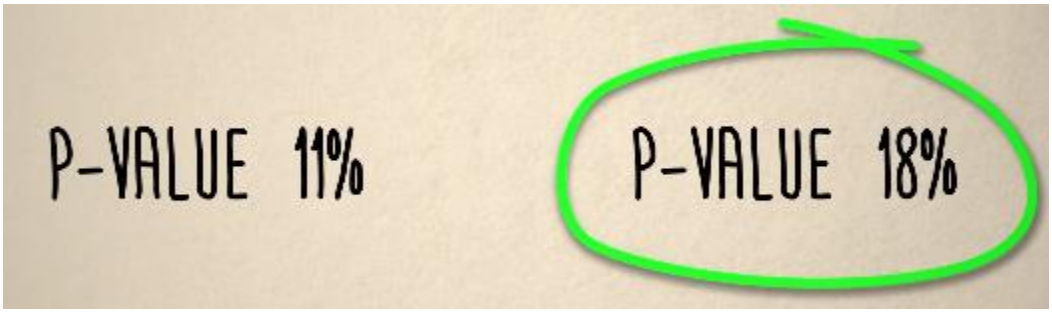
$$\mathbb{P}\left(Z \leq -\frac{0.41}{1.43/\sqrt{60}}\right) + \mathbb{P}\left(Z \geq \frac{0.41}{1.43/\sqrt{60}}\right) = 2\mathbb{P}(Z \leq -2.221)$$

$$= 2.6\%$$

We have the second sample of 60 observations with average 9.81. The difference of 9.81 and 9.4 is 0.41. As a result, the p-value is computing using the following formula. From the standard normal distribution, we find the answer to be twice 0.013, namely 0.026, that is 2.6%. We conclude that the p-value for the second sample is only 2.6%.



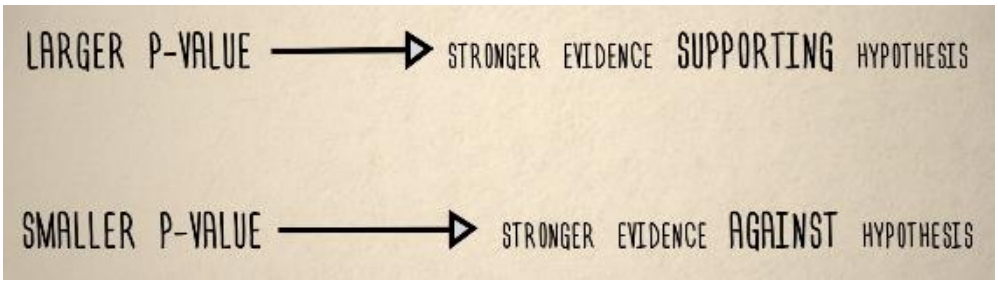
It is still below the 5% cutoff so the rejection is called for, but its value of 2.6% is even lower than the value of 3% for the value of the first sample. So the p-value of the second sample provides, so to speak, even stronger evidence against a null hypothesis.

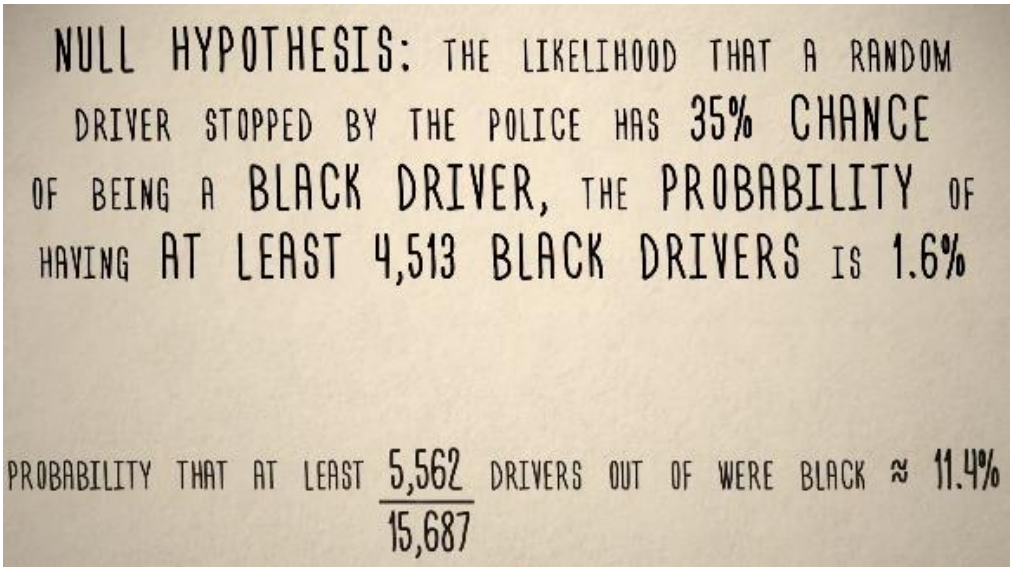


If we had two samples, one with say, p-value of 11% and one with say, 18%, the null hypothesis would be accepted based on both.

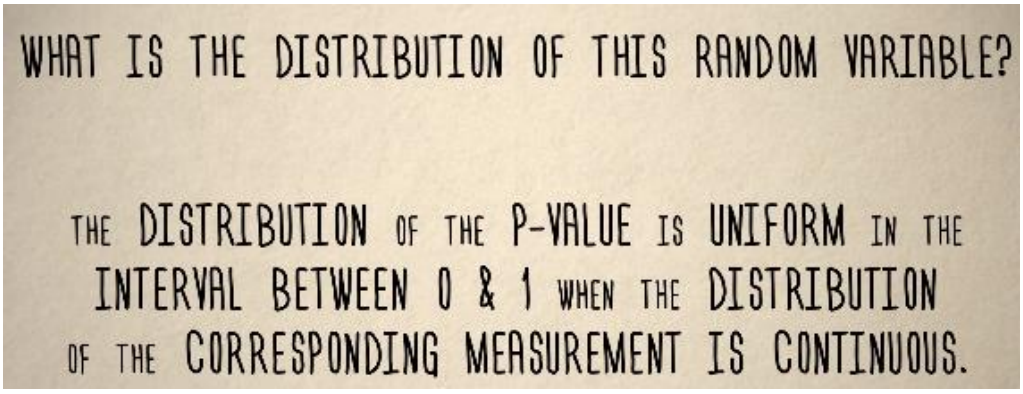
But arguably, the second p-value gives us stronger evidence supporting the hypothesis.
So it is natural to call p-value the strength of the test, a sample mean in our case.

The larger the p-value, the stronger is the evidence supporting the hypothesis. The smaller is the p-value, the stronger is the evidence against the hypothesis.





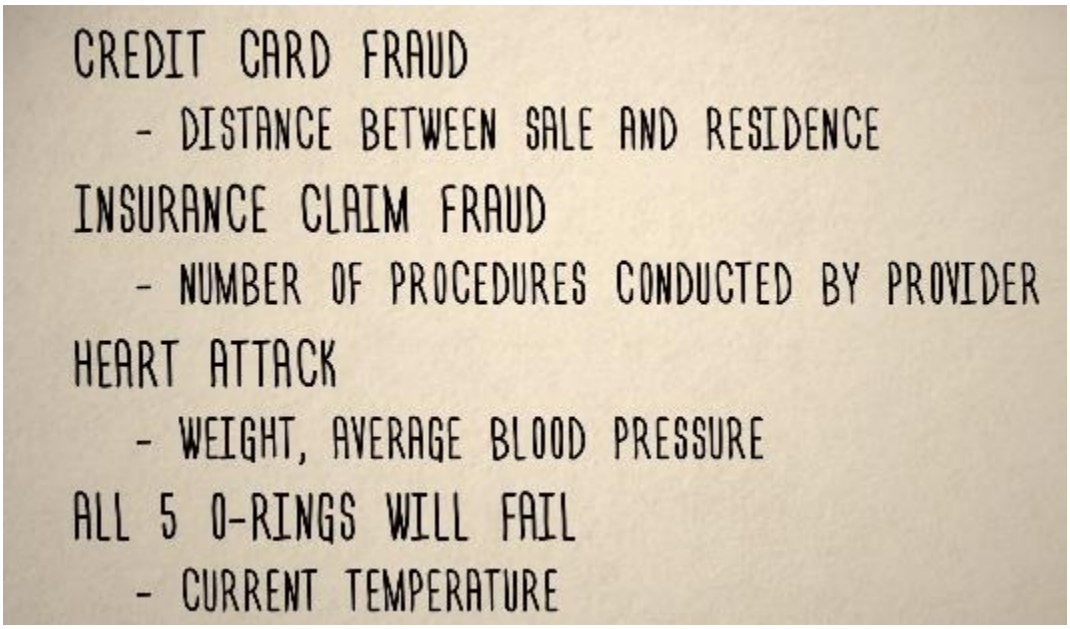
The p-value is naturally a random variable, as it is the probability of a observing a value smaller than the observed sample average. Since the simple average is a random variable so is the probability of seeing the value below it. What is the distribution of this random variable?

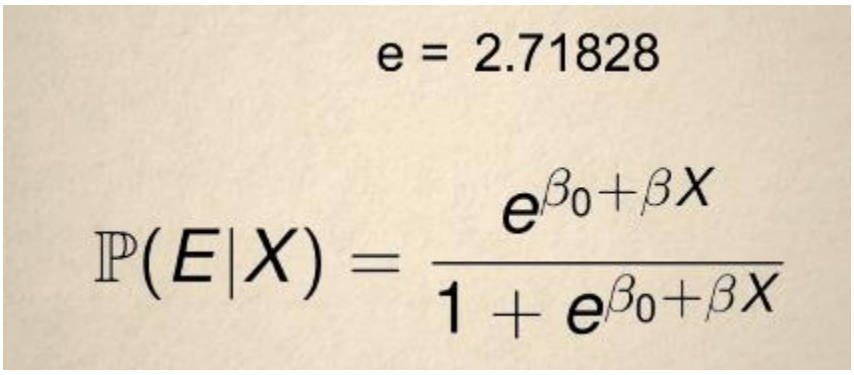


The distribution of the p-value is uniform in the interval between 0 & 1 when the distribution corresponding to the measurement is continuous.

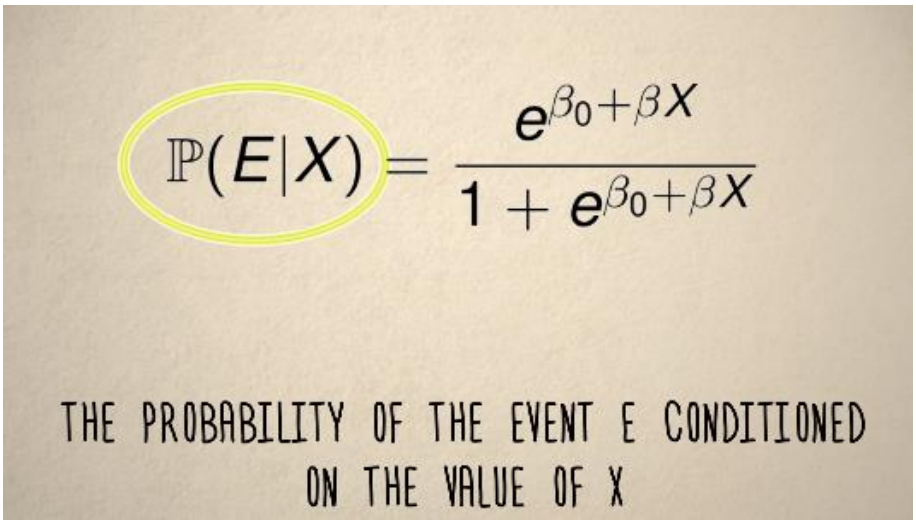
There is a lot of discussion as to what extent one should rely on p-values of say 5% or 1%.
Type 1 errors for accepting or rejecting hypothesis in various fields. These are just useful tools and they need to be used with extreme care and never applied blindly.

3.8 Methods of Estimating Likelihood

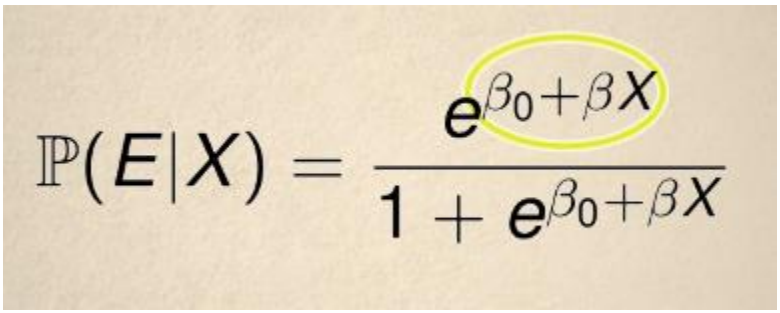




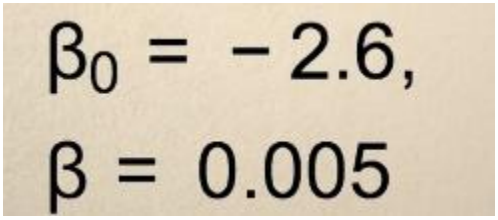
X - independent (explanatory) variable



The expression, which we read as the probability of the event e condition on the value x, is the quantity of interest.



The coefficients beta 0 and beta are called the regression coefficients, and are to be estimated from the data.



Consider our diabetes example. Suppose based on the past medical data, the coefficients beta 0 and beta are estimated to be as follows. Then according to this model, a patient with a body weight of say 135 pounds which has a chance of developing diabetes equal to namely approximately 13%.

Incidentally, approximately 9% of the US population are suffering from some form of diabetes.
So actually, this number is not too unreasonable.

$$P(\text{Diabetes}|\text{Weight} = 135) = \frac{e^{-2.6+0.005*135}}{1 + e^{-2.6+0.005*135}} = 0.1273$$

On the other hand, for a significantly obese patient with weight, say, 275 pounds, the corresponding likelihood is estimated to be as follows, namely approximately 23%.
According to this model, obesity is a significant risk factor for diabetes.

A more accurate way to express this conclusion of the model would be to say that within a group of people with otherwise similar characteristics, a patient with body weight 275 pounds has roughly 23% chances of having diabetes.

$$\mathbb{P}(E|X) = \frac{e^x}{1 + e^x}$$

Let us go back to the expression here. Since the numerator is smaller than the denominator, the ratio takes between 0 and 1. And therefore, it is a suitable expression for a probability value. The function of the form that you see here is called logistic function. And this is where the logistic regression derives its name.

$$Y = 0, 1, 2$$

$$P(\text{Diabetes}|\text{Weight} = X, \#\text{ of parents with diabetes} = Y) = \frac{e^{\beta_0 + \beta_w X + \beta_g Y}}{1 + e^{\beta_0 + \beta_w X + \beta_g Y}}$$

β_0

β_w

β_g

$\beta_0 = -2.7,$

$\beta_w = 0.0045,$

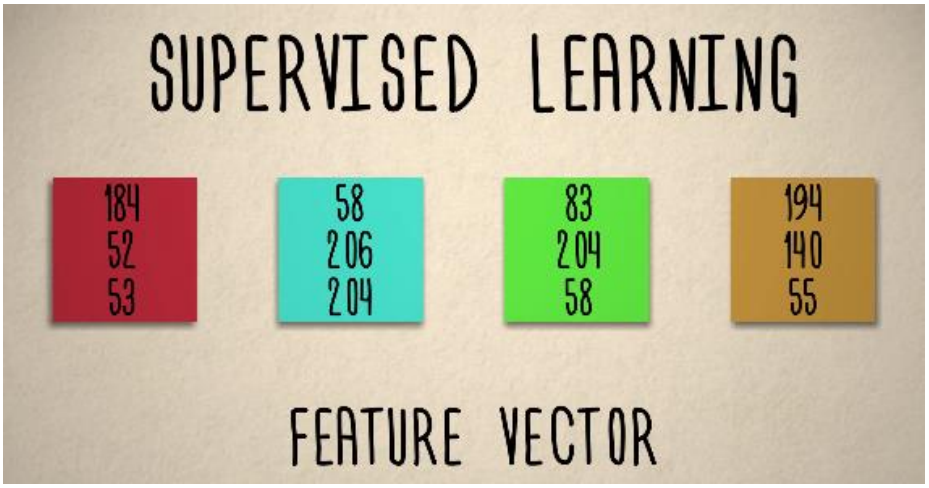
$\beta_g = 0.2$

$$P(\text{Diabetes}|\text{Weight} = 215) = \frac{e^{-2.7 + 0.0045*215 + 0.2*0}}{1 + e^{-2.7 + 0.0045*215 + 0.2*0}} = 0.1881$$

$$P(\text{Diabetes}|\text{Weight} = 215) = \frac{e^{-2.7 + 0.0045*215 + 0.2*2}}{1 + e^{-2.7 + 0.0045*215 + 0.2*2}} = 0.2568$$

3.9 Support Vector Machine: Non-statistical Classifier

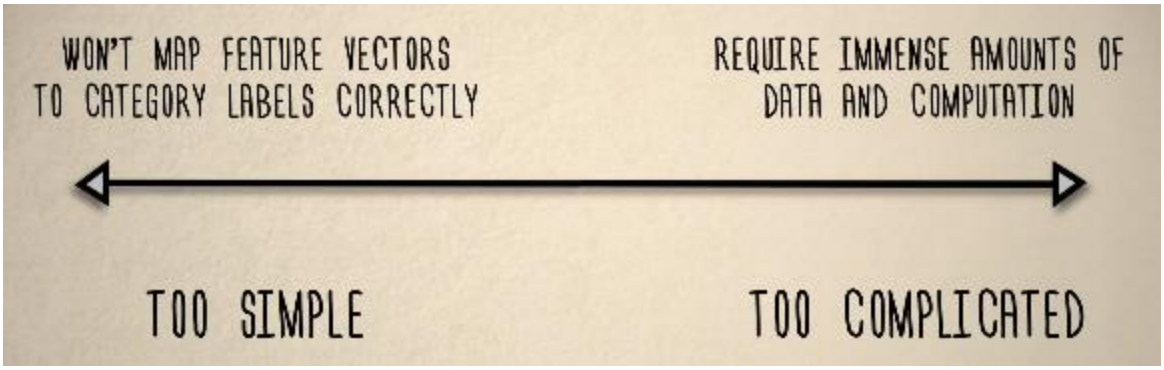
The methods for binary classification that we focused on so far, work in what I'd like to call a statistical setting. Given some input to classify, they postulate some class of probabilistic models, fit various parameters. And using these they give us an estimate of the probability that the input falls into one category or the other.



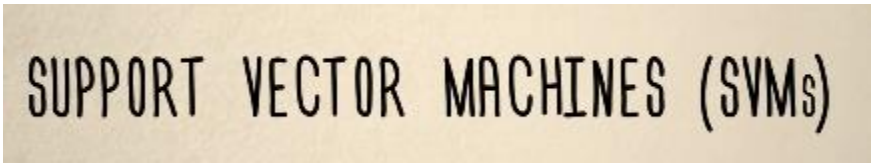
We'll set the problem up in the framework of supervised learning that was discussed earlier in the course. The first step here is to describe our data. To do this, we'll represent each object be classified with a list of numbers that describe it which we'll call a feature vector.

In supervised learning, the process is that we are given some training data which comes in the form of a collection of points that are correctly labeled as positive or negative. Using these, we try to construct a good classification function. We are then given new points, and we classified them by applying our function.

The question then comes down to how we choose this function, which is what distinguishes the different techniques for classification.



There's usually an important trade-off involved here. If the class of functions we look at is too simple, we won't be able to find the function in this class that does a good job describing the right way to map feature vectors to category labels. If we look at classes of functions that are too complicated we'll need immense amounts of data and possibly computation to figure out which one to use.

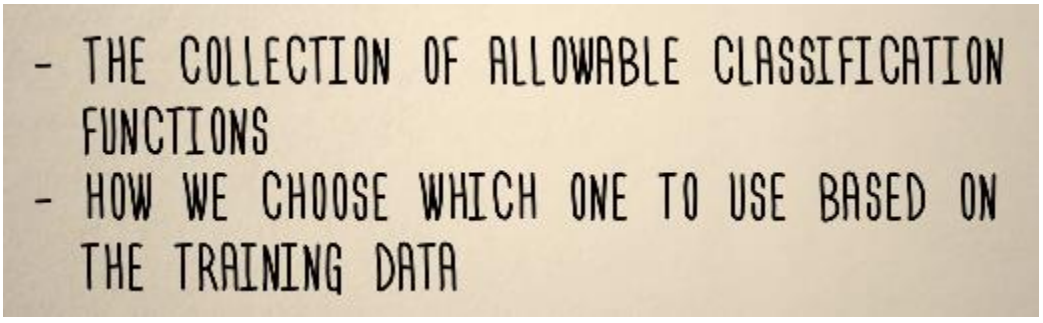


The first method we'll look at is called support vector machines, or SVMs. The category of functions they use will seem very simple, perhaps even too simple. But they're often very effective maps.

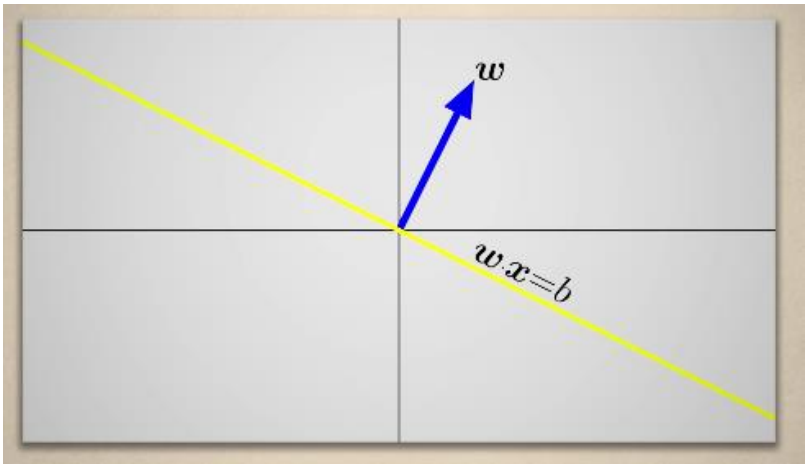
Moreover, we'll see that there is a beautiful and simple technique called the Kernel Trick. That lets them handle a much more complex classes of functions with very little additional work.

In addition, they'll be the fundamental building block out of which we'll construct deep networks, which for many problems give the president state of the art.

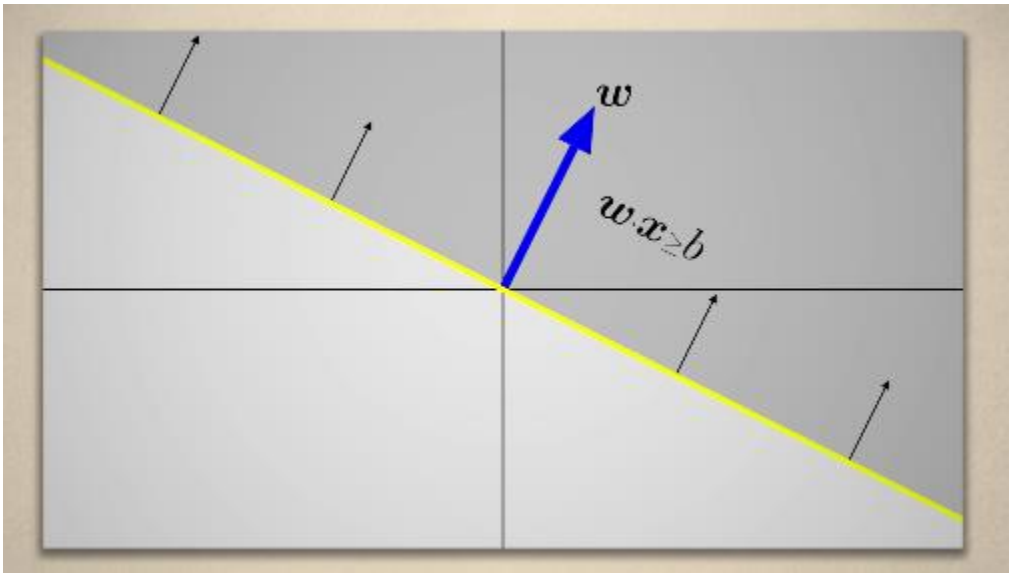
To specify SVMs we'll need to specify two things. The collection of allowable classification functions that we could end up producing. And how we choose which one to use based on the training data.



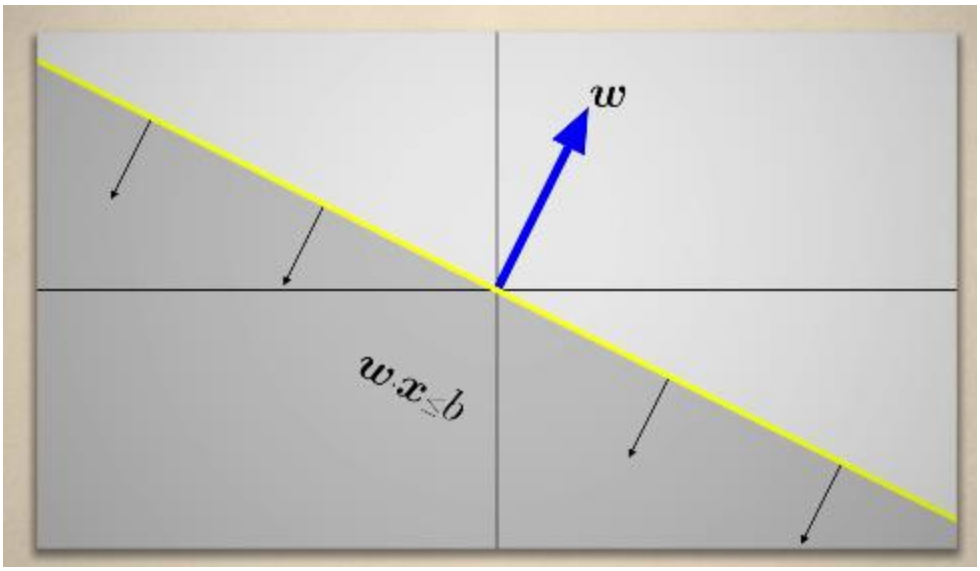
The classification functions we produced will be what we call linear classifiers. They'll work by applying a linear function and thresholding the result. More precisely, we will specify a classifier by giving a vector w of weights, w1through wn, where n is the number of coordinates in the future vectors. And a threshold b, given a data point x, with coordinates x1 through xm.



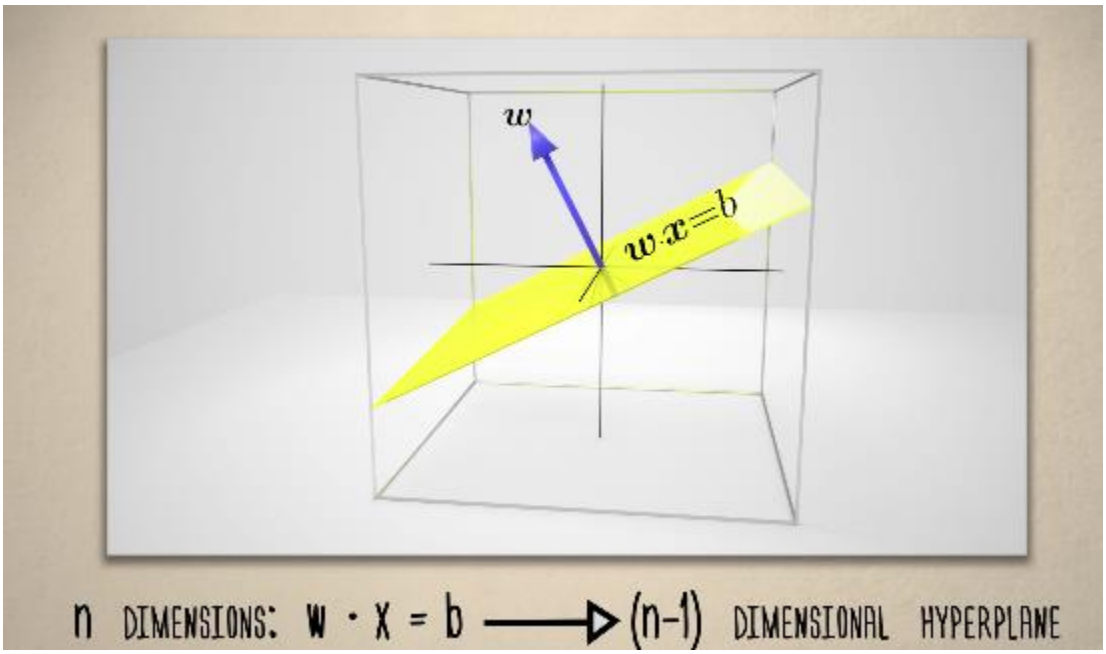
They first compute the dot product, w.x. Which you'll recall means that we add up w1 times x1, plus w2 times x2, etc., up to wn times xm. They'll then compare the resulting number to the threshold b.



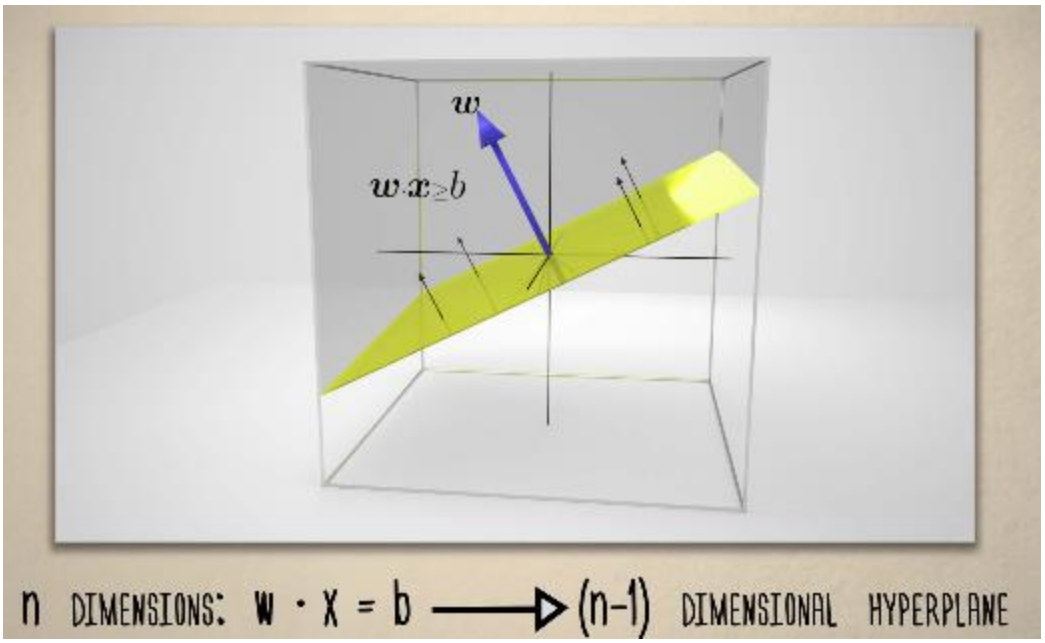
If it's larger than b, they'll put the point in the positive category, otherwise they'll put it in the negative category.

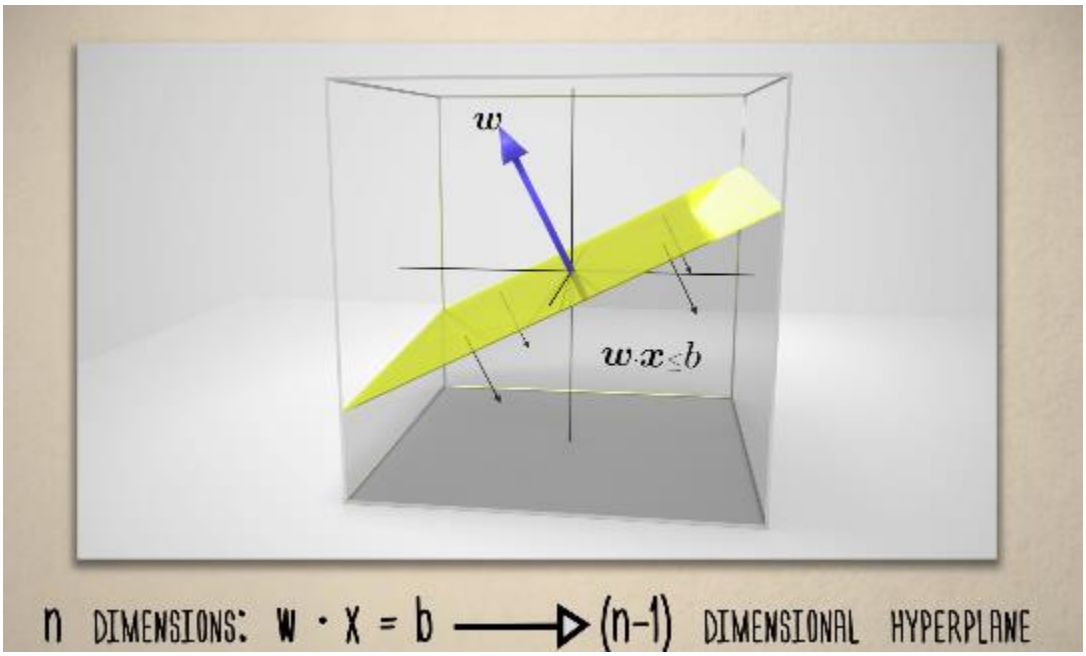


This has a nice geometric interpretation. If say the feature vectors are two dimensional, instead of points with w.x exactly equal to b, it's just the line in the plane. The points with positive dot product lie on one side and the points with negative dot product lie on the other. The classifier is thus just cutting the plane into two pieces with a straight line.



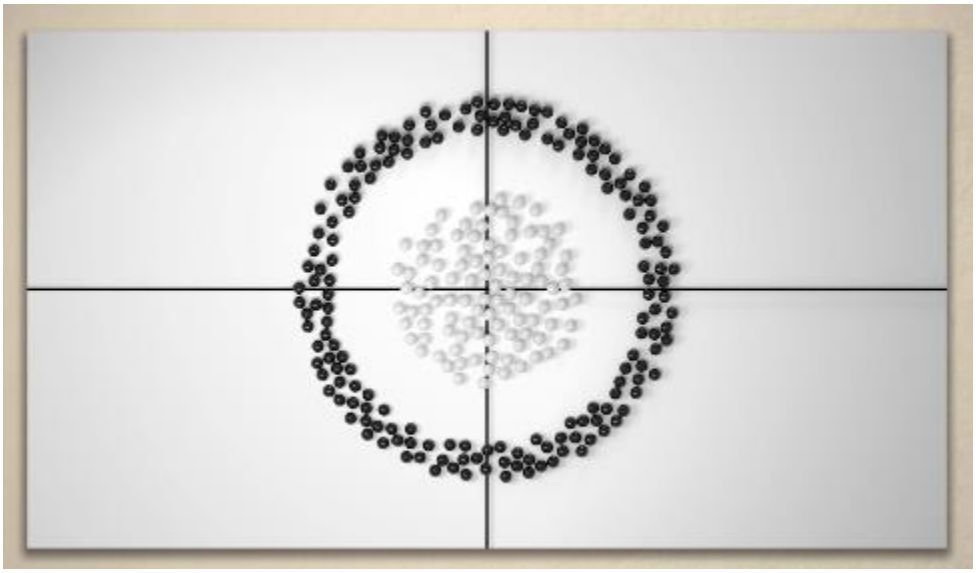
In three dimensions, the set of points with w.x equal to b is a two-dimensional plane, and our classifier uses it to divide the three-dimensional space into two regions.





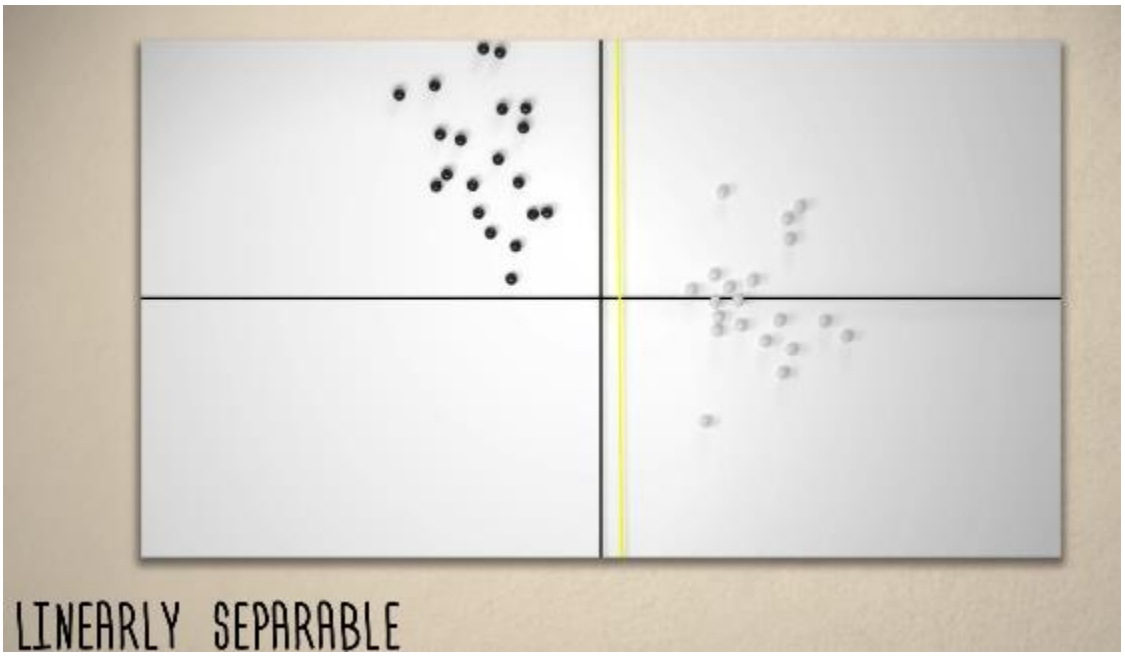
When the dimension is larger, it's a little harder to visualize, but the picture is analogous. The points with w.x = b form an n minus 1 dimensional hyperplane which divides un-dimensional space into two regions. The classifier calls something in one region positive, and something in the other region negative.

Now, let's see how we should choose such a hyperplane given a set of label points of training data. It's not always possible to separate the positive and negative points for the hyperplane.

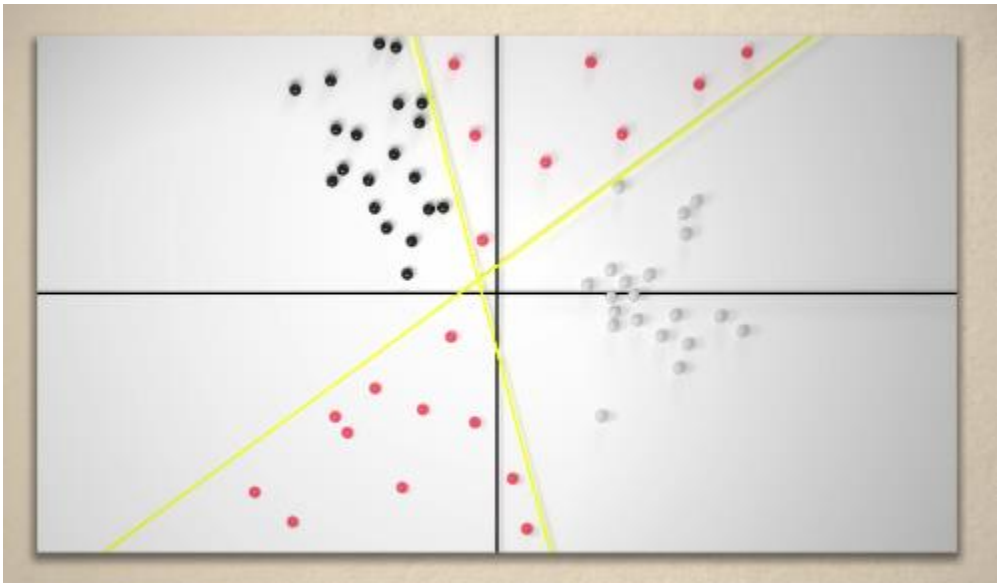


For instance, we can clearly see that there's no way to do it with the points showing here.
However, let's suppose for now that we're in the good case where we can find such a hyperplane that perfectly separates the positive and negative training points.

In this case, we will call the points linearly separable, and it means that there is some linear classifier that correctly classifies all of the training data.

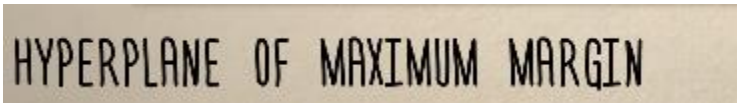


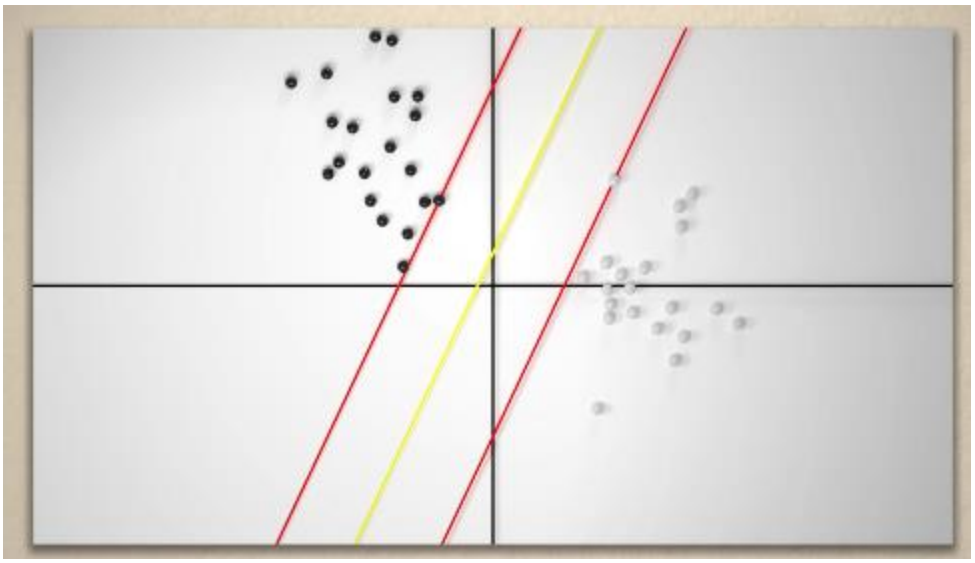
If you look at the picture, you'll see that there could be many such hyperplanes. While they all give the same answers on the training data, it would correspond to very different rules for classifying new objects. For instance, the two hyperplanes shown here both match the training data exactly.



However, they would give very different answers when asked to classify any of the points shown here. To choose among them, SVMs look for what's known as the hyperplane of maximum margin.
Given linearly separable training data, we can see how far we can move this hyperplane until it stops classifying the data correctly.

In the picture, this means that we'll shift in one direction until it hits something in the set of positive points. And then we'll shift it in the other direction until it hits something in the set of negative points.



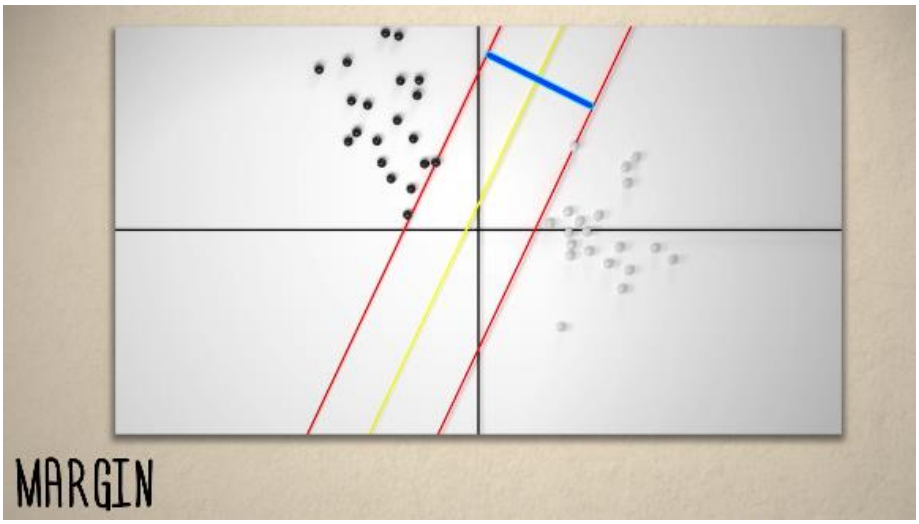


This gives us two new hyperplanes, both parallel to our original one. And they both correctly classify our training data. Intuitively, when these hyperplanes are very far apart, this means that the original hyperplane very robustly divides the two groups in the training data.

Because it worked with room to spare so to speak, this means that it's reasonable to guess that if we had some new training data the hyperplane would probably still work fairly well.

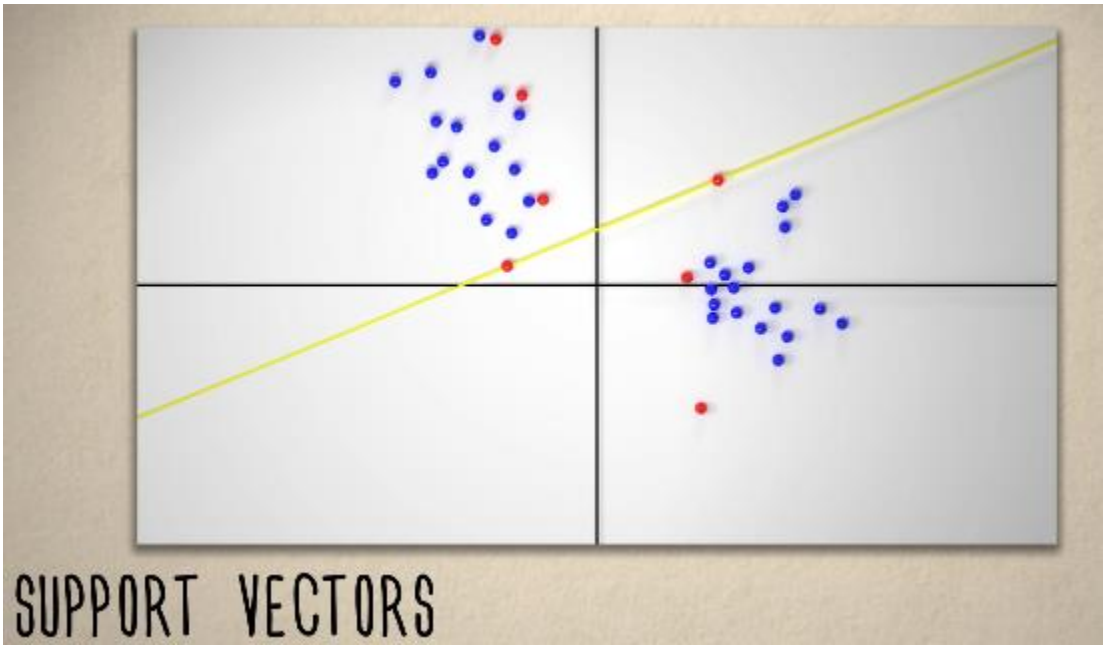
We'll call the distance between these two parallel hyperplanes the margin.

Now support vector machines work by looking at all the hyperplanes that divide the two groups in the training data. And then they try to find the one with the largest possible margin.



If we imagine shifting and rotating the hyperplane around to see what possibilities are, you'll see that it can hit the points that lie nearest to it, shown here in red.

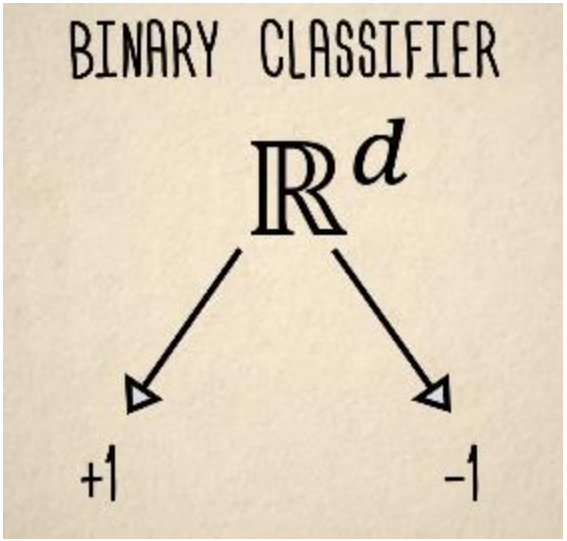
And that these are the ones that determine everything we need to know. Whereas the ones that are farther away, shown in blue, don't really matter. And moving them around a bit doesn't change the answer. The red points are called support vectors, which is why we call these support vector machines.

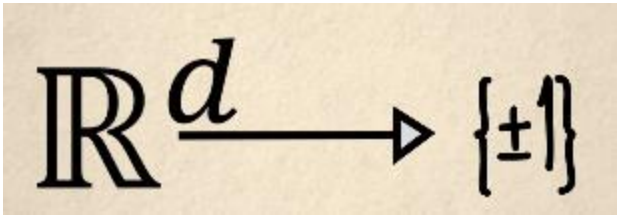


Going forward, there are two very natural questions that we'll need to answer.
The first is whether and how we can find this optimal hyperplane?
And the second is, what we should do when the points are not linearly separable?

3.10 Perceptron: Simple Classifier with Elegant Interpretation

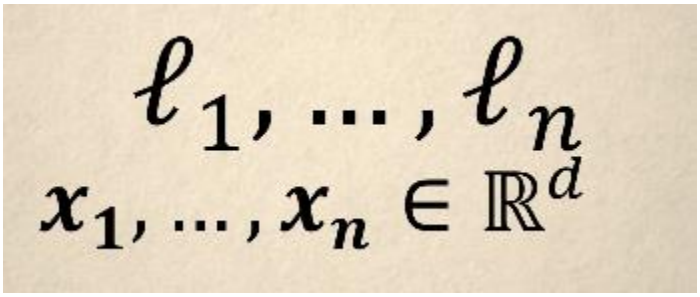
In the previous section, we looked at how to frame binary classification problem in the setting of supervised learning. Let's briefly review where we left off, and define some notations to use going forward.



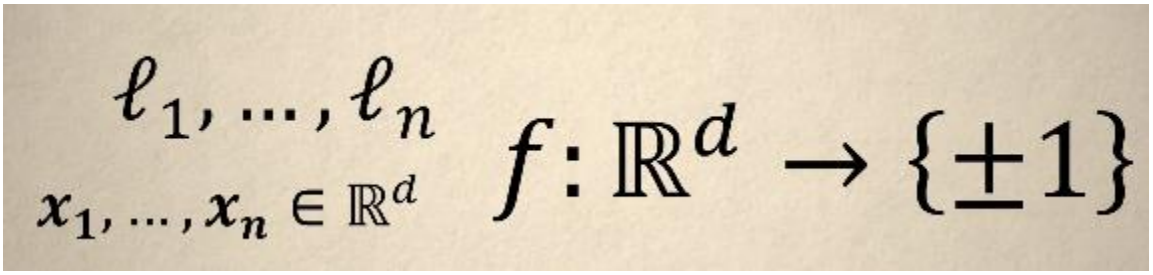


We described each object to be classified as a list of d real numbers, which we call the feature vector. We can think of these as points in d-dimensional space, which

we'll call Rd. A binary classifier is then a function that maps each point in Rd to one of the two possible categories.

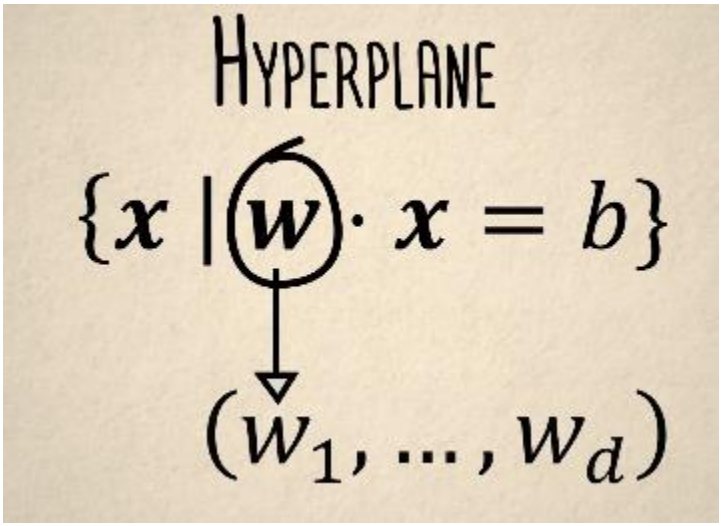


If we label the positive examples with plus 1 and the negative examples with minus 1, we can thus succinctly describe the binary classifier as a function from Rd to the set containing plus and minus 1.

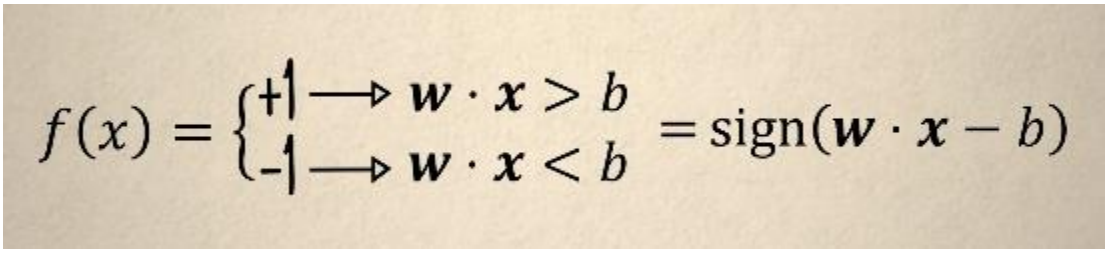


We're then given a bunch of training examples, which consist of points x1 through xn, in Rd, along with our correct labels, l1 through ln.We use these to decide what function f from Rd to plus or minus 1 to use as our classifier. Then, given a new point to classify, we get our answer by just applying the function f.

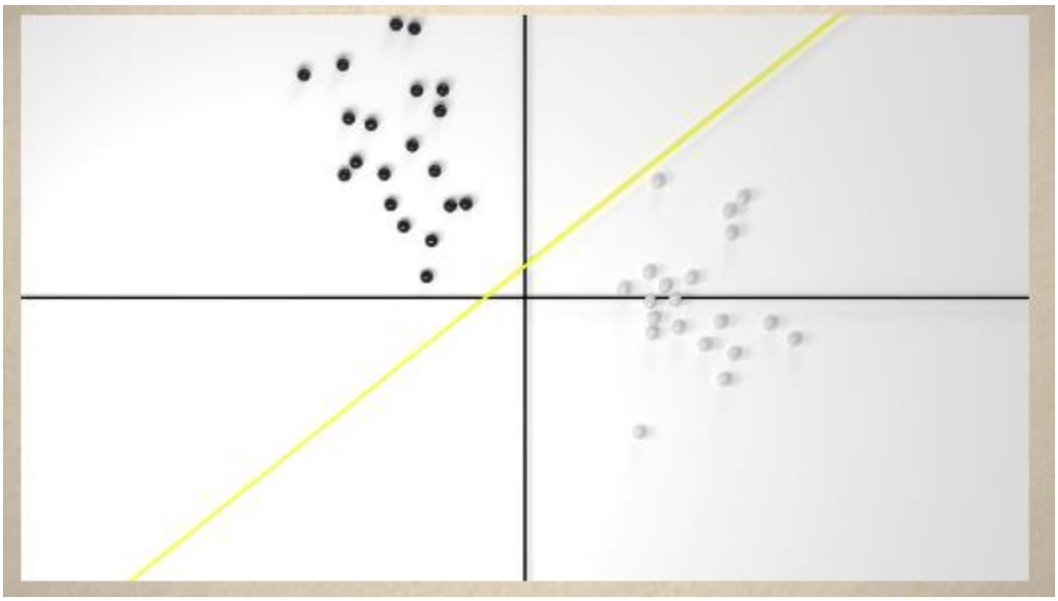
Deciding what class of functions to choose the classifier from is one of the crucial design choices to be made when setting up our classifier. And we decided to start with the case of linear classifiers, which choose a category by dividing space up into two categories using a hyperplane.



We can describe the hyperplane as the set of points with w .x equals b. Where w equals W1 through Wd is a vector in Rd and B as a number.



If we use plus or minus 1 to label our categories, we can write or classify very cleanly. We want to return plus 1 is w .x is bigger than b, and minus 1 if it's smaller. So, this just says that f of x is equal to the sign of w .x minus b.



We then looked at the case where there was a hyperplane that correctly classifies all of our training data. In which case, we said that the points were linearly separable.

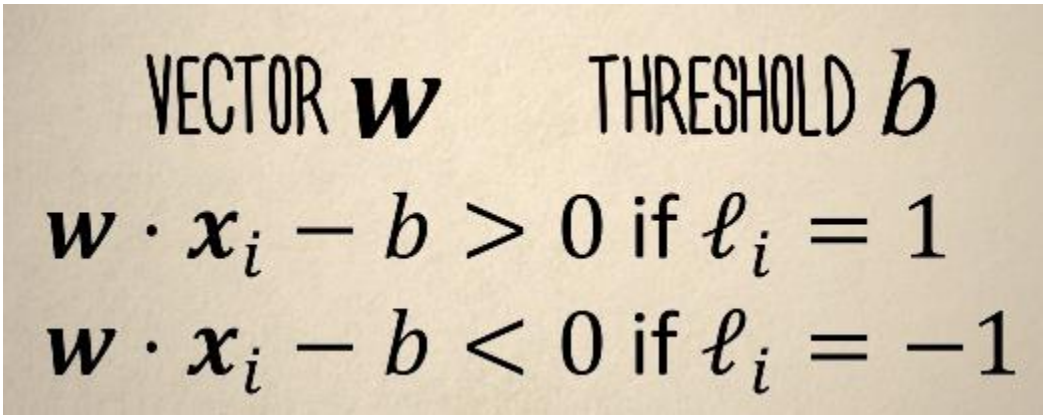
We define the margin of such a hyperplane to be the amount we can shift in either direction, while still correctly classifying all the training data.

Given linearly separable training data, there'll often will be many hyperplanes that separate the positive and negative examples, so we have some freedom in how to construct our linear classifiers.

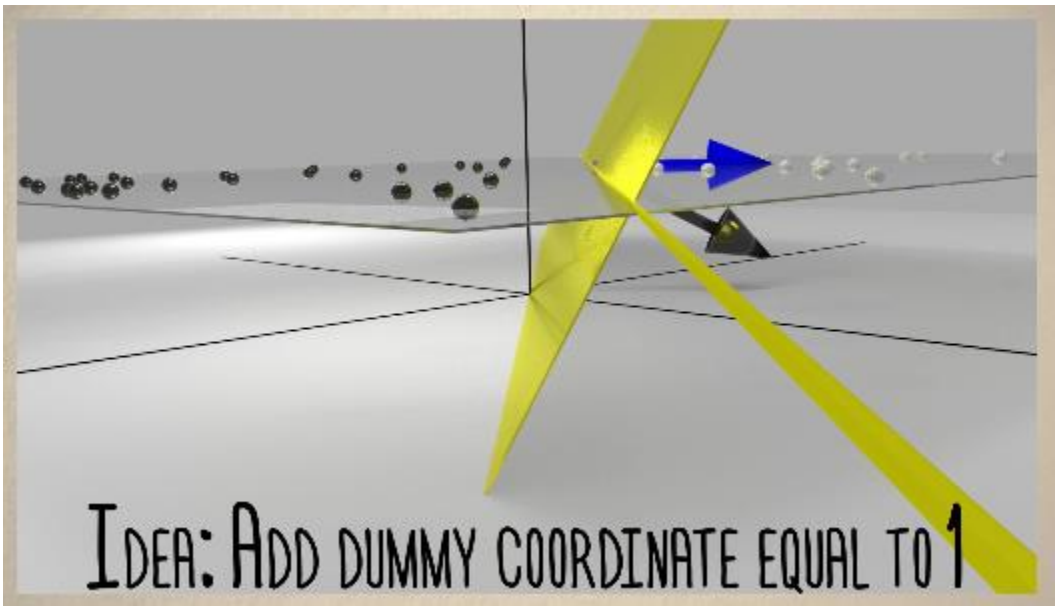
One role people often use to make this decision is to choose a hyperplane with the largest possible margin. The resulting linear classifiers called the Support Vector Machine.

Our classifier is specified by the vector w and the threshold b. For it to correctly classify the ith training example, we need w .x of i minus b to be greater than 0, if l sub i equals 1, and less than 0 if l sub i equals minus 1. Keeping the vector b around

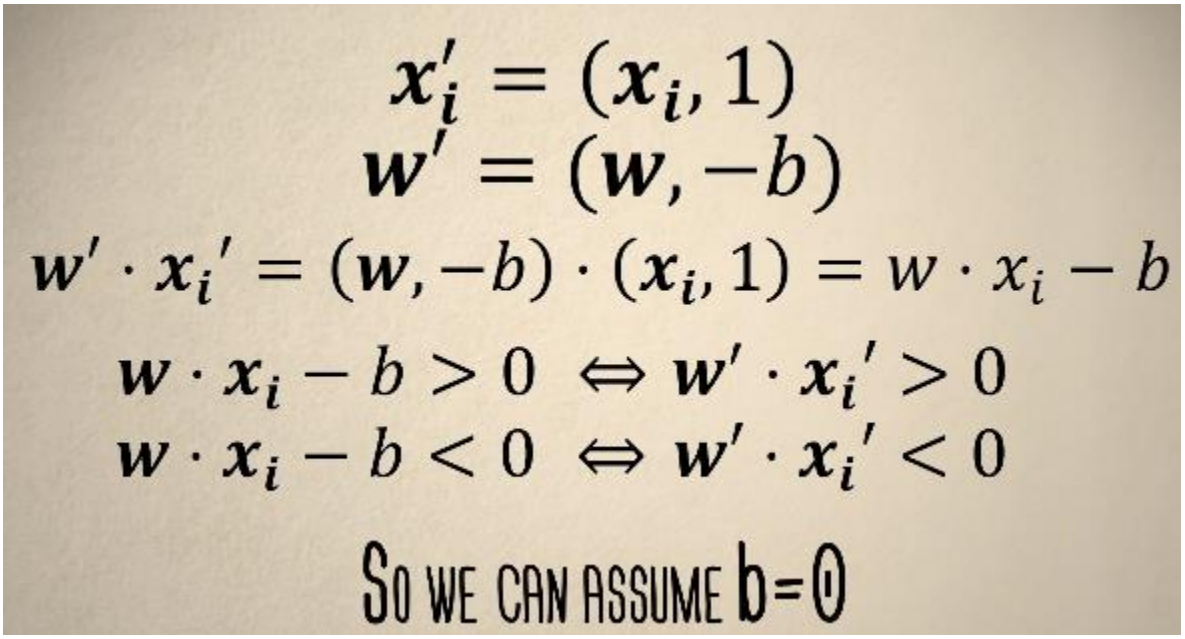
ends up being a little bit annoying. It'd be cleaner if we were in the case where b equals 0.



It turns out that there's a nice trick that lets us reduce to this case.

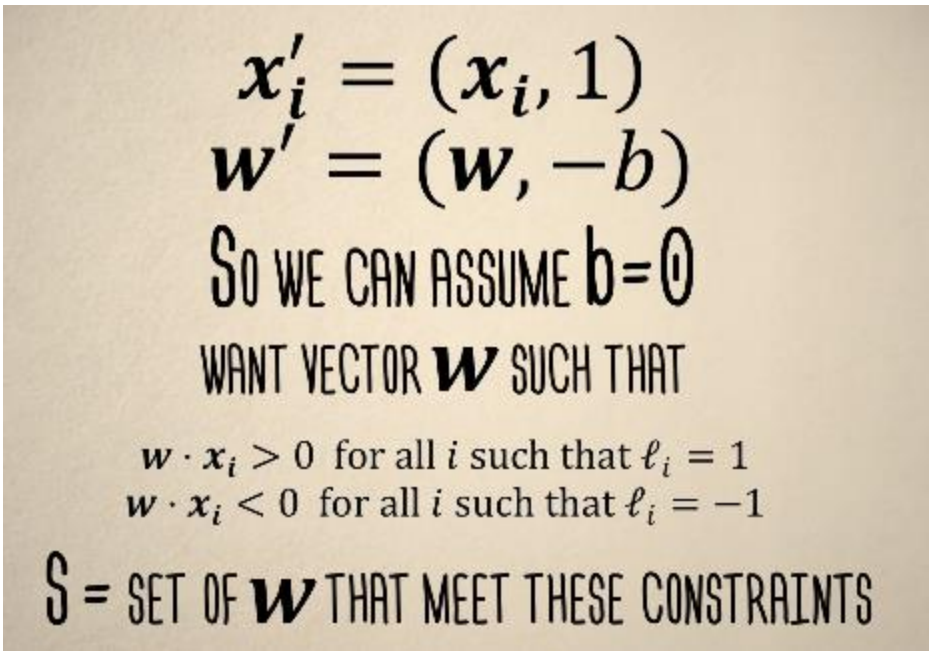


And the idea is just that we're going to add the dummy coordinate and set it equal to 1 for all of our training points. So, we now think of our training points as vectors, x sub 1 prime through x sub n prime, where x sub i is the d plus 1 dimensional vector X sub i comma one.



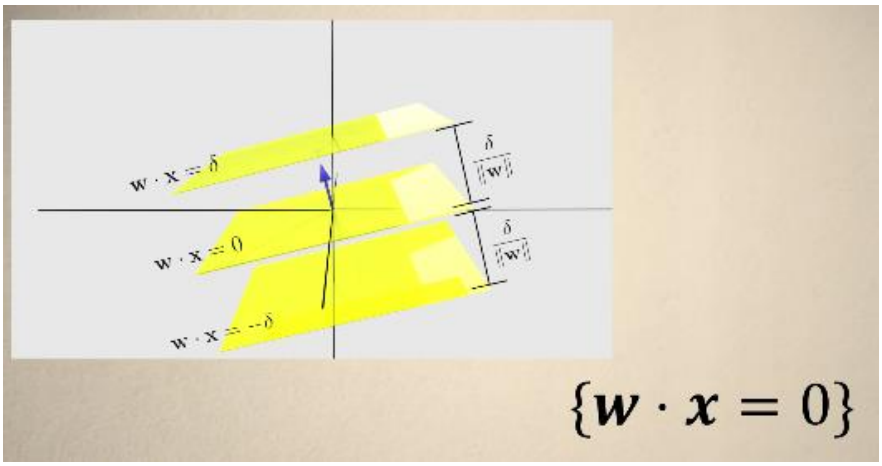
We then pull b into our weight vector by defining w prime to be the vector w comma negative b.
This makes it so that w prime dot x sub i prime equals w dot x minus b. So, finding a vector w such that w dot x sub i minus b is greater than or less than 0 is the same as finding a vector w minus prime, such that w minus prime .x sub i minus prime is greater than, or less than 0.

And now we don't have to worry about b. We can thus assume from now on that b is 0 without loss of generality.

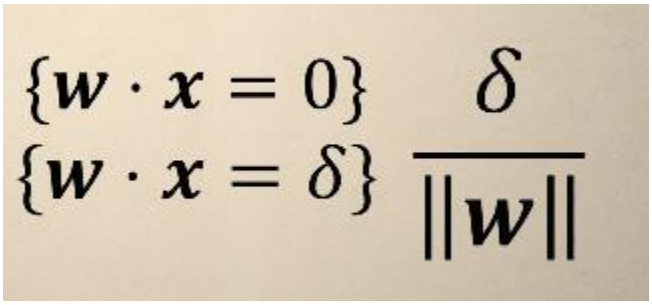


With this pre-processing step, the algorithmic problem of finding a classifier that works for all of the training data comes down to finding a vector w that meets all the constraints coming from the x of i. That is we want to find a vector w, such that w .x of i is greater than zero, for all i such that l sub i equals one. And w .x sub i is less than 0 for all i such that l sub i equals negative 1. Let s be the set of w that meet these constraints.

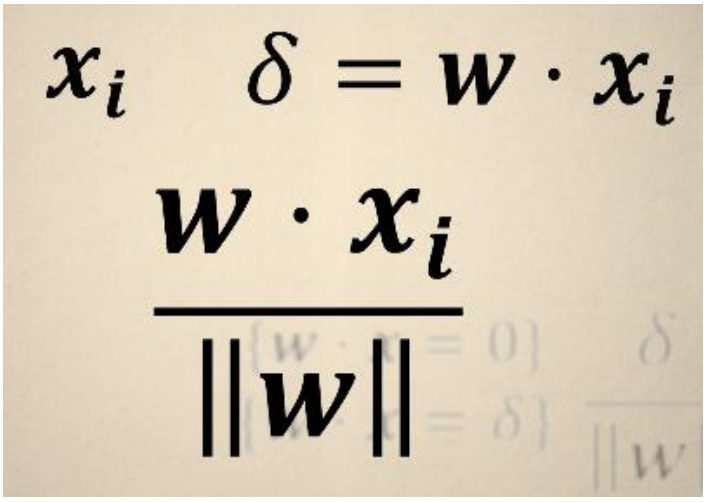
To find the support vector machine, we wanna find the w in s with maximum margin.



If we start with the hyper plane w .x equals 0, we can get shifted versions of it by just changing the right hand side. How quickly the hyperplane moves when we change the right hand side depends on the norm of w.



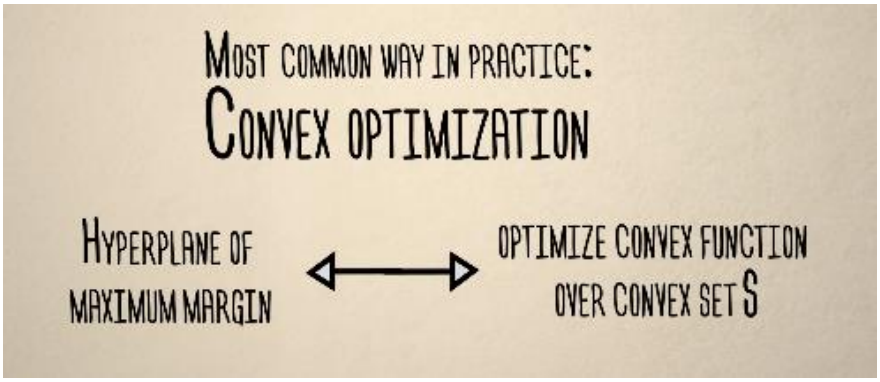
And one can check if the hyper plane w dot x equals delta, corresponds to shifting the original one by a distance of delta over the norm of w.



If we shift the hyperplane so that it hits a point xi, which corresponds to the margin of the hyperplane with respect to the point xi, this will be the hyperplane with delta equal to w .xi.
Which means that we shifted it by W. Xi over the norm of W. By scaling these constraints, we can write the problem of finding the hyperplane of maximum margin as just minimizing the norm of W.

Now, there are a few ways one can find the W we're looking for. In other sections of this course, both before and after this one, an important theme has been that we have these powerful algorithmic techniques available for solving convex optimization problems.

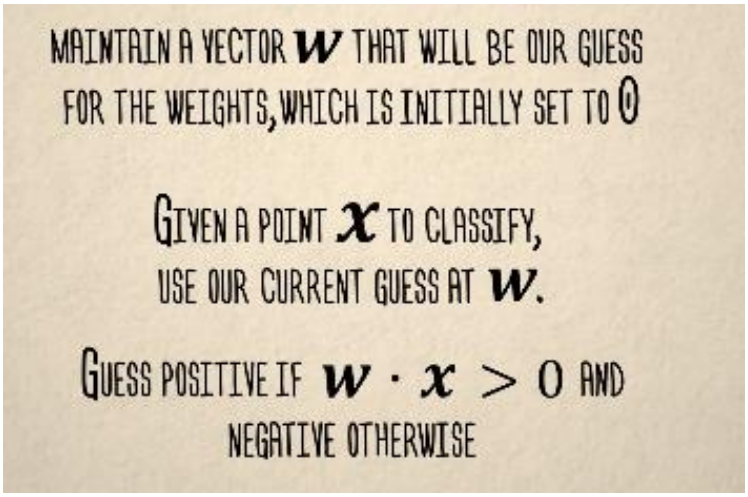
In practice, the most common approach for finding linear classifiers in general, and FPNs in particular, is to use convex optimization.



The basic idea is that one can rewrite the problem of finding the hyperplane of maximum margin as optimizing a convex function over the set S.

And we can then apply the convex optimization machinery. This machinery has been carefully tuned for SVMs, and there are excellent libraries available for doing this.

However, rather than just running through some algebra to write the problem in a convex form, and then referencing the existing general algorithms for these problems, I'd like to instead talk about a beautifully simple way of solving the problem, that I think actually might be a bit more insightful. It's called the perceptron algorithm.



And it's not quite as efficient as the convex optimization routines, and I'm just gonna talk about it for finding any linear classifier, rather than to find the one with the largest margin. But it has some beautiful ideas in it. And it will let me give you a full, self-contained algorithm.

Moreover, it solves a more general version of the problem called online learning.

We can now finally describe the output.
The idea is as follows.
Instead of being given all of the training data, we'll imagine that somebody hands us the points from it one at a time. Possibly giving us the same point more than once.

Each time we're given a point, we have to assign it to a category. After we do, we find out what the actual label was, and we can use this information for the later points.

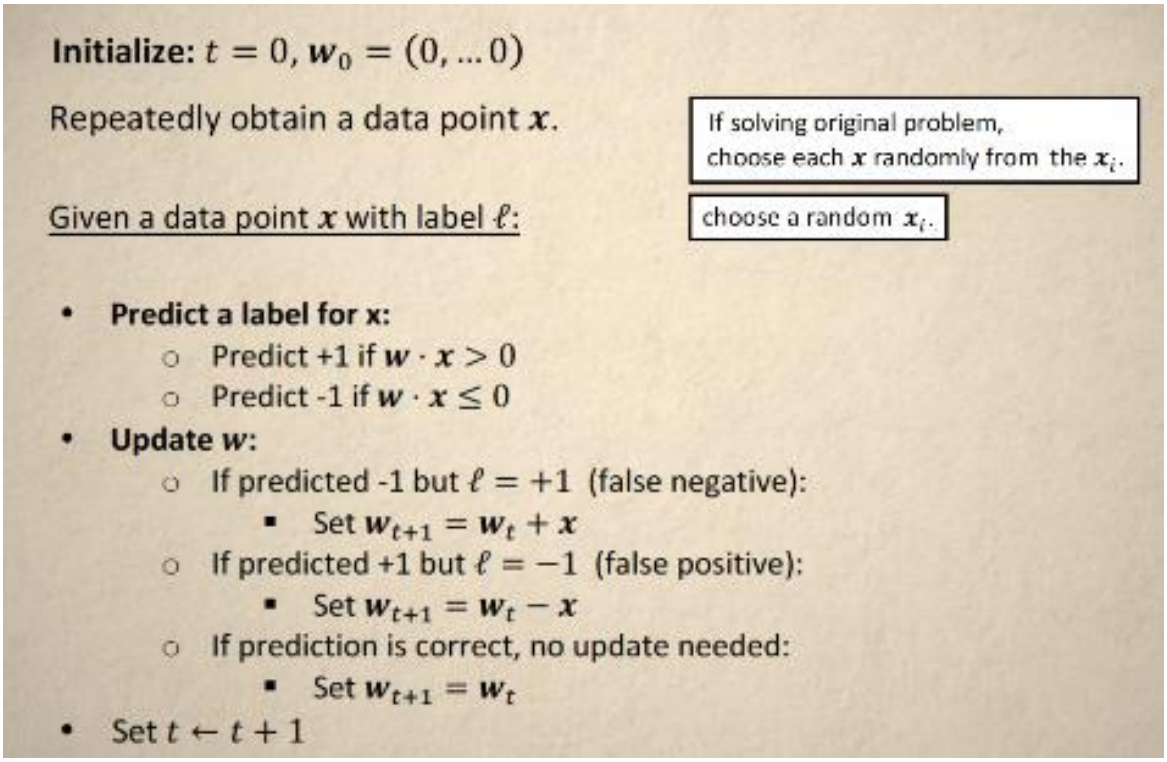
Now, the difference here is that we have to make our decisions online. That is, we don't get to see the whole training stat before we have to classify some points. Our algorithm for doing this is going to be very simple.

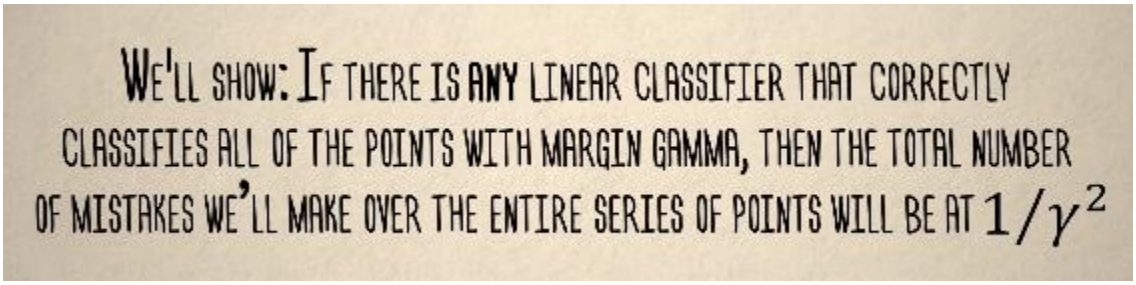
We'll maintain a vector w that will be our guess for the weights, which is initially set to zero. Given a point, X, to classify, we'll then use your current guess for W.

So, we'll guess positive if W .X is greater than 0, and negative otherwise. Now, if we guessed right, we'll leave our guess for W unchanged.

However, if you make a mistake, we'll update our guess to try to fix it. Specifically, if we guess negative, and the answer was positive, we'll replace w with w plus x.

This increases w .x, so it pushes w in the direction of answering positive for x. Similarly, if we guess positive when we should have said negative, we'll replace w with w minus x.

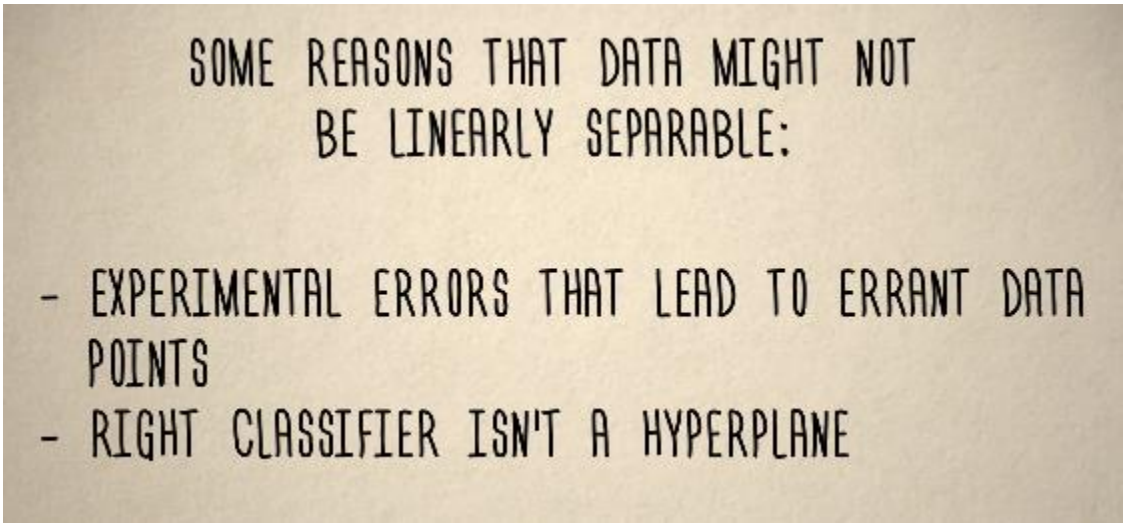




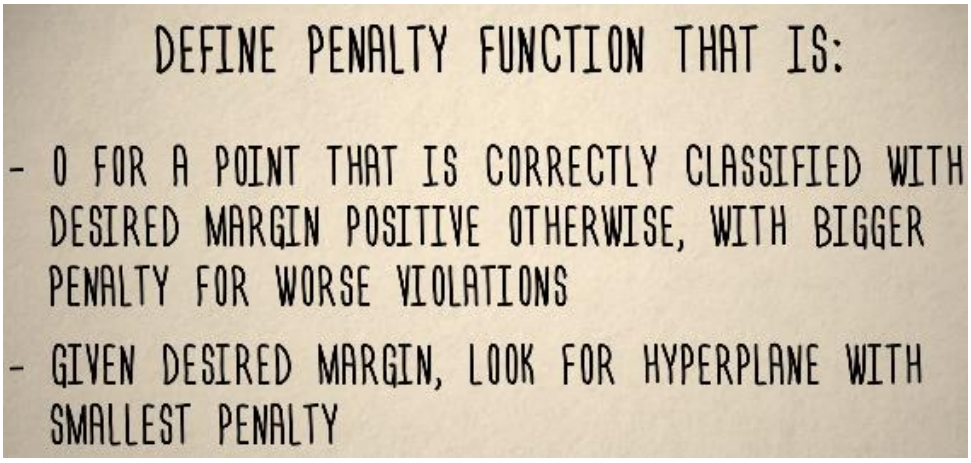
3.11 Perceptron Proof

In this section, we'll prove our previous claim that the perceptron algorithm makes at most one over gamma squared total mistakes, where gamma is the margin. I should mention that this section may be slightly more mathematically involved than the others. I think it provides a lot of insight into why the perceptron algorithm works, so I decided to include it. But nothing we do later will rely on the details of this proof. So if you skip this or you don't fully follow the details, it won't cause any problems with the later material.

3.12 Perceptron & Data that is Not Linearly Separable



In this section, we'll give a brief overview of some of the simple modifications that allow us to apply support vector machines in these more complicated settings.



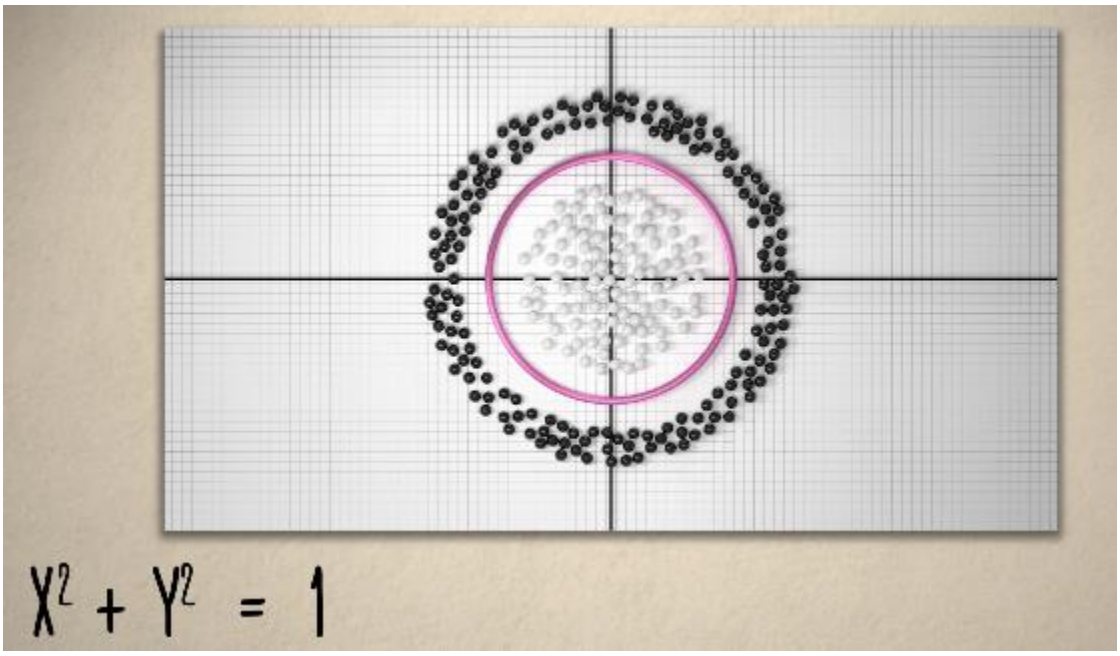
In the first case, where we want a linear classifier, but there's no hyperplane that perfectly classifies the training data, the idea is just to slightly modify our goal.Instead of requiring our hyperplane to cross by all of our points in some given margin, we'll make a penalty function that zero for a point that is classified by the hyperplane with this margin and that penalizes points that violate the constraint. And we'll make the size of the penalty depend on how bad the violation is.

Then given some margin, we can look for the hyperplane that has the smallest penalty. Note that there's a tradeoff here. If we demand a larger margin, we should expect more violations and thus a larger penalty so we have to somehow decide how to balance the two. There's also some discretion in how we choose our penalty function.

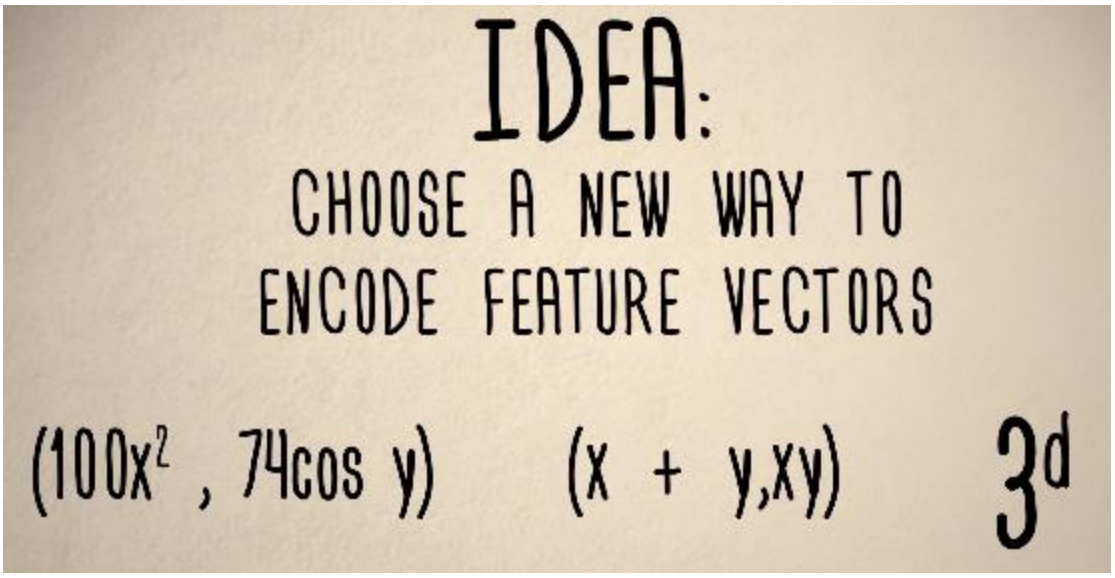
Researchers have studied how to make these choices, and there's some simple, commonly used penalty functions that work very well in practice. And in fact, for these penalty functions, it turns out we can actually find the optimal hyperplane using algorithms that are almost the same as the ones we used in maximum margin setup.

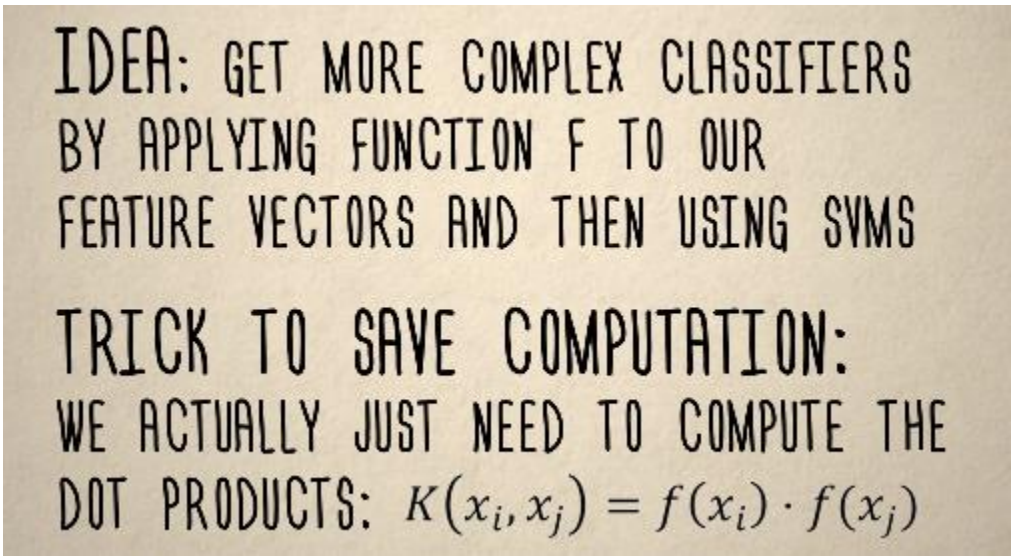
Let's now talk about the second problem I mentioned which actually seems to be a bit more problematic.This is the problem where there is a right classifier, but it's just fundamentally not a hyperplane. For instance, it could be that the best way to classify the data is by looking at some higher degree polynomial instead of a linear function.

To see how this could happen, let's look more carefully at the example I showed before. In this case, any linear separator that we try will miss-classify a significant faction of the trait data. Here's a picture of what this would look like. However, we would correctly classify all the training data and get what visually seems to be the right classifier by looking at the degree two polynomial x squared + y squared, labeling a point as positive when it's bigger than one and negative when it's smaller than one.

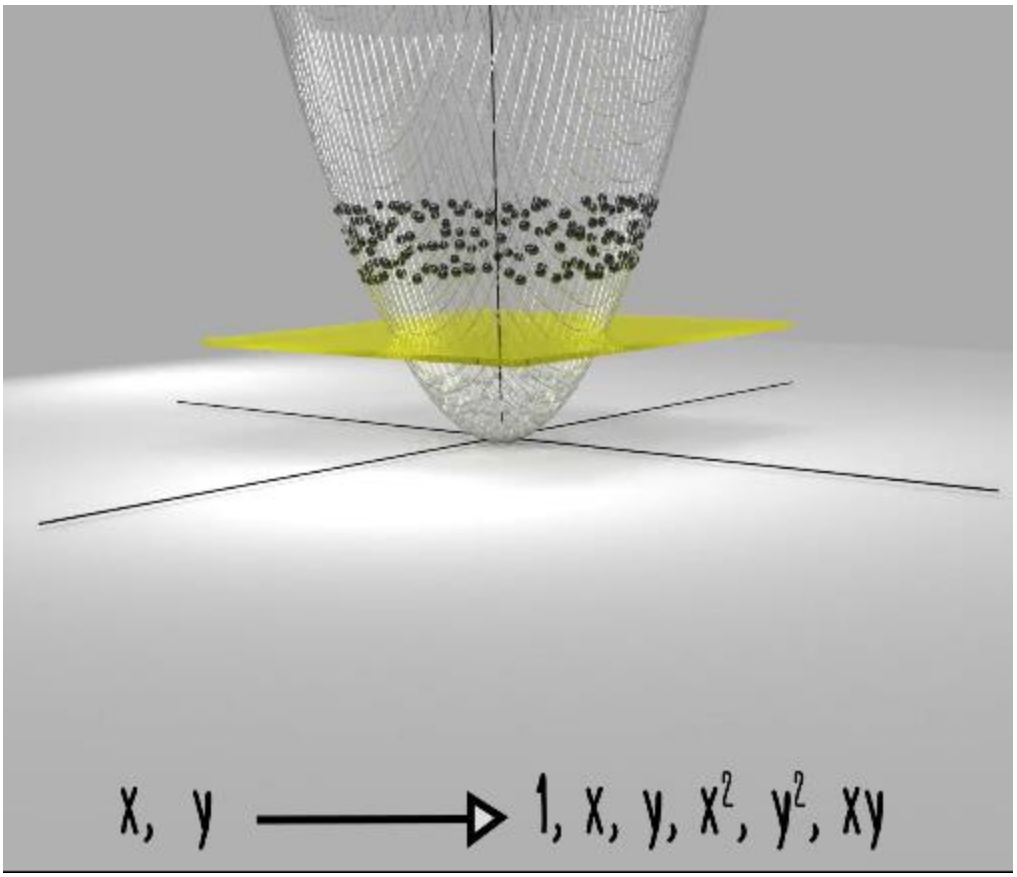


Unfortunately, our whole approach was tuned into finding a hyper plan so seems like we need to start from the scratch and try with all of the theory and all the algorithm that we have. Before completely anew if we wanna get to classifier that has some other shape. Like the ones we got to be higher. Our priority we don't even know if we can do this on a reasonable way. Our algorithms, like the perceptron algorithm we described, or the convex programming methods that I mentioned, were all really about linear separators. Priori, it actually seems quite possible that finding the best classifiers with say degree two polynomials could just be fundamentally harder, and end up being too hard a problem, one that we can't solve with any efficient algorithm.





it lets us apply our SVM machinery to get non linear classifiers.



$$1, x, y, x^2, y^2, xy$$

$z_1 = 1$
$z_2 = x$
$z_3 = y$
$z_4 = x^2$
$z_5 = y^2$
$z_6 = xy$

$$f(x,y) = (z_1, z_2, z_3, z_4, z_5, z_6) = (1, x, y, x^2, y^2, xy)$$

$$f(x,y) = (z_1, z_2, z_3, z_4, z_5, z_6) = (1, x, y, x^2, y^2, xy)$$

$$c_1 z_1 + c_2 z_2 + c_3 z_3 + c_4 z_4 + c_5 z_5 + c_6 z_6$$

$$c_1 + c_2 x + c_3 y + c_4 x^2 + c_5 y^2 + c_6 xy$$

LINEAR CLASSIFIERS APPLIED TO Z ⟷ QUADRATIC CLASSIFIERS APPLIED TO X,Y

$$c_1 + c_2 x + c_3 y + c_4 x^2 + c_5 y^2 + c_6 xy$$

$$c_1 = c_4 = c_6 = 1$$
$$c_2 = c_3 = c_5 = 0$$

$$1 + z_6 + z_6 = 1 + x^2 + y^2$$

FOR QUADRATIC CLASSIFIERS:

- APPLY A FUNCTION INTO HIGHER-DIMENSIONAL SPACE
- LOOK FOR LINEAR CLASSIFIERS

x^3, xy^2, x^2y, y^3

FOR DEGREE 3 CLASSIFIERS:

X,Y -> X^3, X Y^2, X^2 Y,Y^3

x^3, xy^2, x^2y, y^3

FOR DEGREE k CLASSIFIERS:

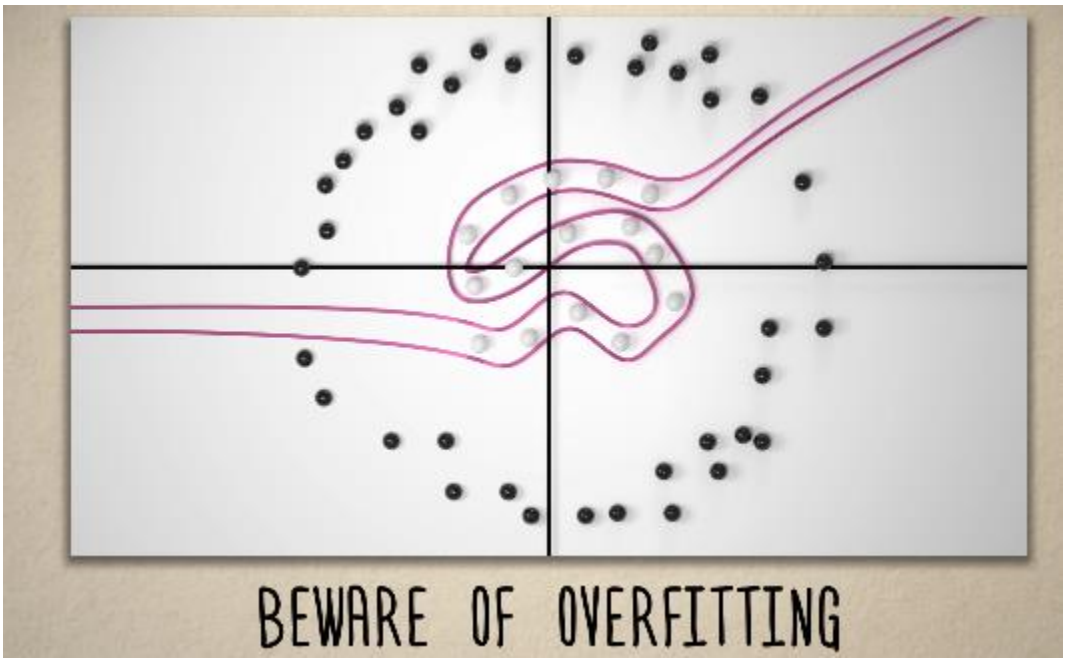
X,Y -> ALL MONOMIALS OF DEGREE $\leq$ k

WORRY: HUGE NUMBER OF COORDINATES
TOO BIG TO EFFICIENTLY COMPUTE?

SOLUTION: YOU CAN DESCRIBE ALGORITHMS
IN A WAY THAT DOESN'T REQUIRE COORDINATES
OF THE DATA, JUST THE DOT PRODUCTS

$$f(x_i) \cdot f(x_j)$$

$$f(x_i) \cdot f(x_j) = (1 + x_j \cdot x_j)^d$$

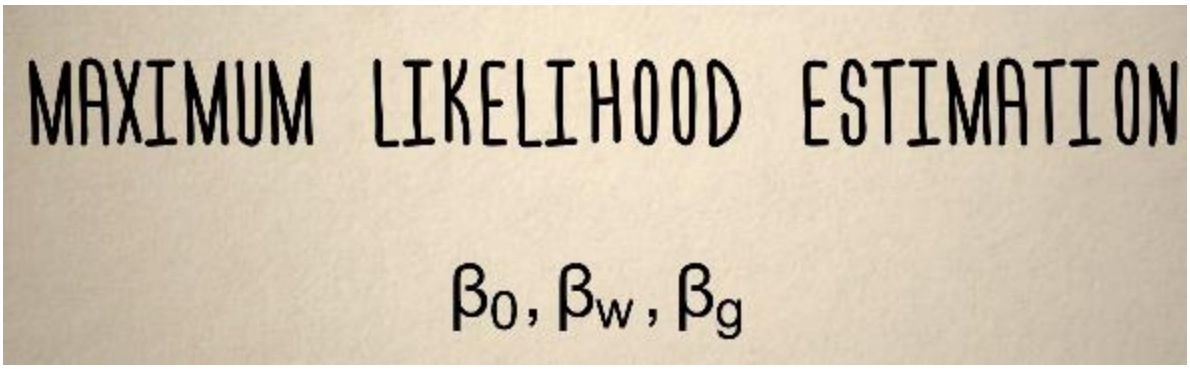


3.13 Estimating the Parameters of Logistic Regression

It is now time to discuss methods for estimating the parameters of logistic regression. Namely the parameters such as beta 0, beta, beta w, and beta g.

As we noted earlier, these parameters are estimated using historical data. In particular, the data will be used in combination with a so called maximum likelihood estimation.

Let us first describe the general principle in abstract terms and then illustrate it with a simple example. Suppose, we have a collection of the data in the following form.



$$(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$$

Here X1, X2 and all the way to Xn are observations for independent variables.
And Y1, Y2 and all the way to Yn indicate whether the event of interest takes place.
In particular, Ys only take values 1 and 0.
Here, the convention is that Yi takes value 1, if the event took place for the ith element of the sample, and takes value 0, otherwise.

Now, remember, that according to the logistic regression model with one independent variable, we have the following relationship between the observations and data.
Say, for example, that Y1 equals 1, then we have the following relationship. On the other hand, say, for example, that Y3 equals 0. Then according to our model, we have the following.

$$P(Y_1 = 1|X_1) = \frac{e^{\beta_0 + \beta X_1}}{1 + e^{\beta_0 + \beta X_1}}$$

$$P(Y_1 = 1|X_1) = \frac{e^{\beta_0 + \beta X_1}}{1 + e^{\beta_0 + \beta X_1}}$$

$$P(Y_3 = 0|X_3) = \frac{1}{1 + e^{\beta_0 + \beta X_3}}$$

$$P(Y_1|X_1) = Y_1 \frac{e^{\beta_0 + \beta X_1}}{1 + e^{\beta_0 + \beta X_1}} + (1 - Y_1) \frac{1}{1 + e^{\beta_0 + \beta X_1}}$$

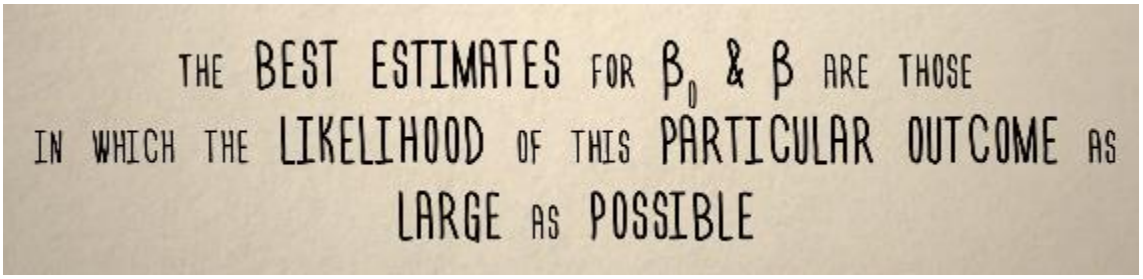
$$P(Y_3|X_3) = Y_3 \frac{e^{\beta_0 + \beta X_3}}{1 + e^{\beta_0 + \beta X_3}} + (1 - Y_3) \frac{1}{1 + e^{\beta_0 + \beta X_3}}$$

The probability of observing a particular outcome for all individuals from 1 to n simultaneously is the product of probabilities for individual outcomes. And can be represented by the following expression.

$$P(Y_1, \ldots, Y_n|X_1, \ldots, X_n) = \prod_{i=1}^{n} \left(Y_i \frac{e^{\beta_0 + \beta X_i}}{1 + e^{\beta_0 + \beta X_i}} + (1 - Y_i) \frac{1}{1 + e^{\beta_0 + \beta X_i}} \right)$$

Now recall that the values of Xs and Ys are actually known to us.These are what we have collected in our data sample. The only unknowns are beta 0 and beta coefficients, which we want to estimate.

The maximum likelihood estimation is a principle which suggests that the best estimation of beta0 and beta are those which makes the expression on the right hand side of the equation as large as possible.



In words, one could say, the best estimate for beta 0 and beta are those which make the likelihood of this particular outcome as large as possible.

Body weight	Diabetes (Yes/No)
186	0
132	0
156	1
220	1
190	0
152	0
285	1
172	1
205	1
146	0

Let us turn to a concrete example to illustrate this idea. Imagine, that we have the following data on some individuals. Their body weight and whether or not they have developed diabetes. For simplicity, we will consider the case of a small data size, consisting of only ten data points.

But naturally, in practice, you would want to use methods with a larger number of data samples for better accuracy. The left column simply gives the list of body weights for the ten people. The right column indicates whether the corresponding individual has diabetes. The value 1 stands for the case when the answer is yes, does have diabetes.

And the value 0 stands for the case when the answer is no, does not have diabetes. For example, the individual with weight 186 pounds corresponding to the first row in the table does not have diabetes. And the individual with a weight 156 pounds corresponding to the third row of the table does have diabetes.

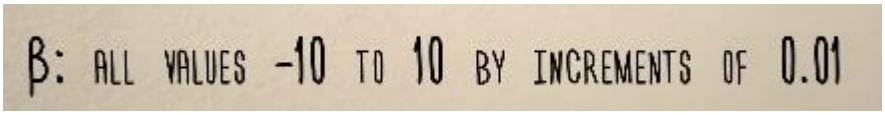
Then according to our formula, we need to find beta 0 and beta, which maximize the following expression.

$$\left(0\frac{e^{\beta_0+186\beta}}{1+e^{\beta_0+186\beta}}+1\frac{1}{1+e^{\beta_0+186\beta}}\right)$$
$$\times\left(0\frac{e^{\beta_0+132\beta}}{1+e^{\beta_0+132\beta}}+1\frac{1}{1+e^{\beta_0+132\beta}}\right)$$
$$\times\left(1\frac{e^{\beta_0+156\beta}}{1+e^{\beta_0+156\beta}}+0\frac{1}{1+e^{\beta_0+156\beta}}\right)$$
$$\vdots$$
$$\times\left(0\frac{e^{\beta_0+146\beta}}{1+e^{\beta_0+146\beta}}+1\frac{1}{1+e^{\beta_0+146\beta}}\right)$$

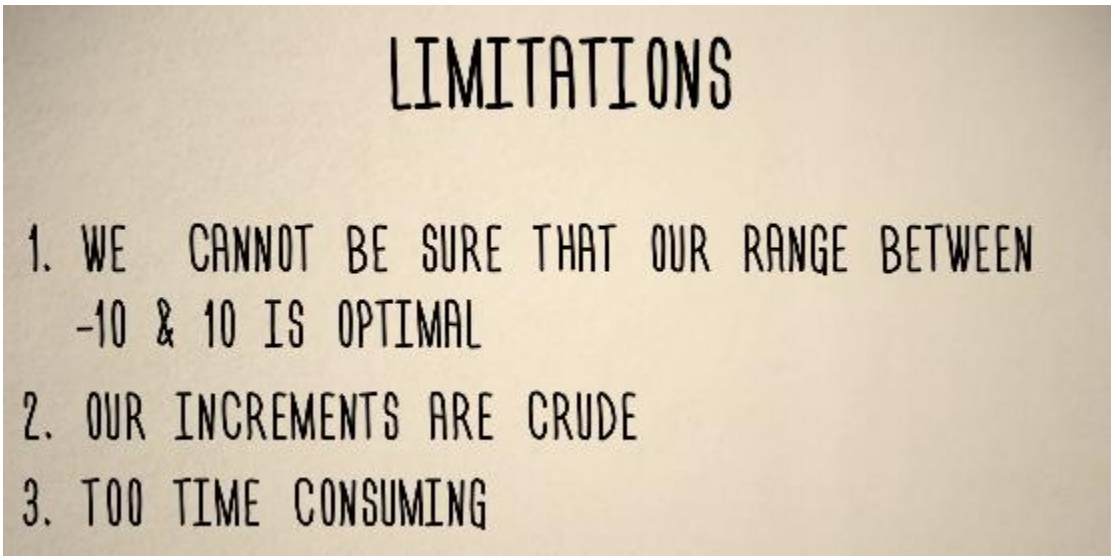
Now ignoring the term where we multiply by 0s, we can simplify the formula as follows.

$$\log 1-\log(1+e^{\beta_0+186\beta})$$
$$+\log 1-(1+e^{\beta_0+132\beta})$$
$$+\log(e^{\beta_0+156\beta})-\log(1+e^{\beta_0+156\beta})$$
$$+\cdots$$
$$+\log 1-\log(1+e^{\beta_0+146\beta})$$

So, we need to find beta 0 and beta which make this expression as large as possible. How can we do this?



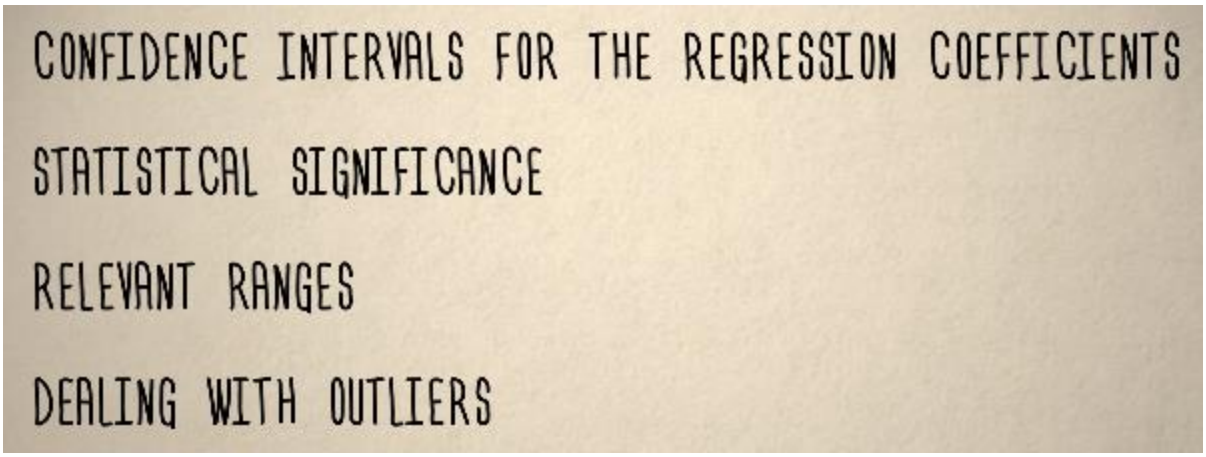
One straightforward approach is to try all possible values of beta 0 and beta within some range, by say, iterating over small increments.For example, try all beta 0 and beta between -10 and 10 with increments say 0.01 and select the pair which gives the largest value. This method however, has several important limitations.



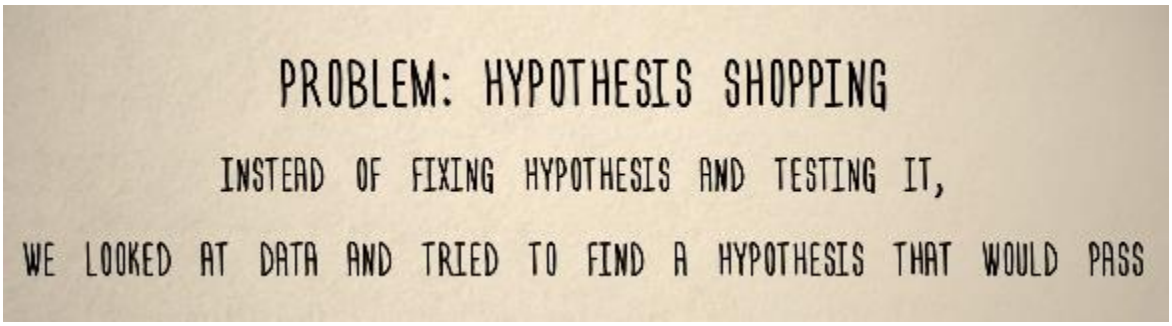
The concave function is particularly nice when it comes to finding its largest value, namely solving the maximization problem. There is a variety of methods, including the so called Newton's method or the gradient descent method, which find the maximum values for such functions.

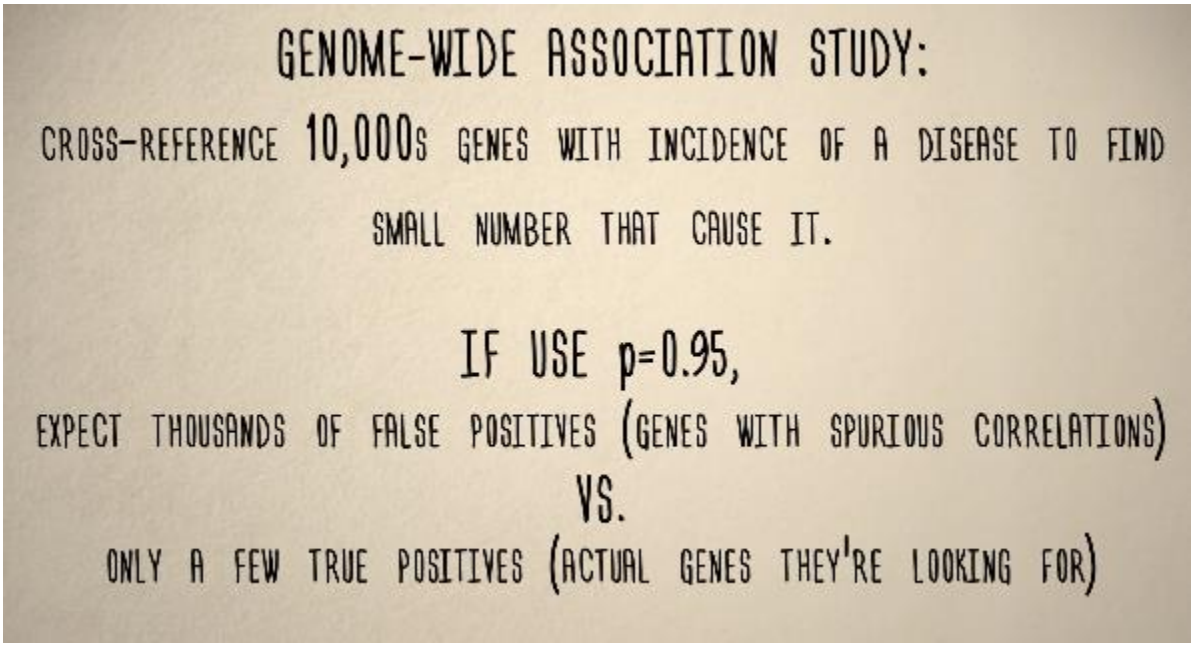
These methods work by finding larger and larger values of function by small incremental improvements, until the value of x corresponding to the largest value of the function f of x is found.

The crucial concavity property guarantees that these steps will converge to the point which is the maximum point for the function f.



3.14 Misapplications of Statistical Techniques





3.2.1 Introduction to Deep Learning

In this unit, we're gonna study a revolutionary, exciting, new approach to classification, called, deep learning. In contrast to the approaches we've talked about so far, which are, for the most part, well understood, no one really knows when, or why, deep learning works so well.

In fact, truth be told, there's a lot of hype around it, and it's sometimes difficult to separate the facts from the fiction. What I hope that you'll get out of this unit, is a better sense of what new applications deep learning enables, what makes it so exciting, but also what are the caveats and dangers that come along with it.

Deep learning is a major driving force behind many of the recent advances in machine learning, and it's arguably the reason why today truly is the golden age for machine learning.

But there are also many subtle issues lurking underneath the facade. I think it makes sense to start with the excitement, so why is everyone so excited about deep learning?

The human mind is amazing, this phenomena is often called one-shot learning, because it takes just a few examples to learn entirely new category.

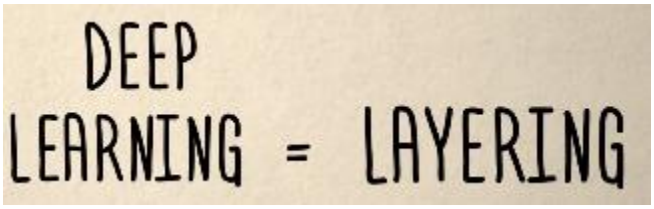
But until recently, the state of the art algorithms can only achieve error rates between 25 and 30% on ImageNet. This is still quite good. If you were to guess five categories randomly for each image, your error rate would be 99.5%. But it's still quite far away from what humans can achieve, which is somewhere in the range between 5 and 6%. Some of the categories are so specific, that humans get them wrong too.

This all changed in 2012 when a team of researchers used deep learning to achieve error rates about 15%. In fact, even these bounds were quickly improved upon.

Every year since 2012, the competition has been won by deep learning. And the entire field of computer vision has shifted its center of mass towards these new techniques.

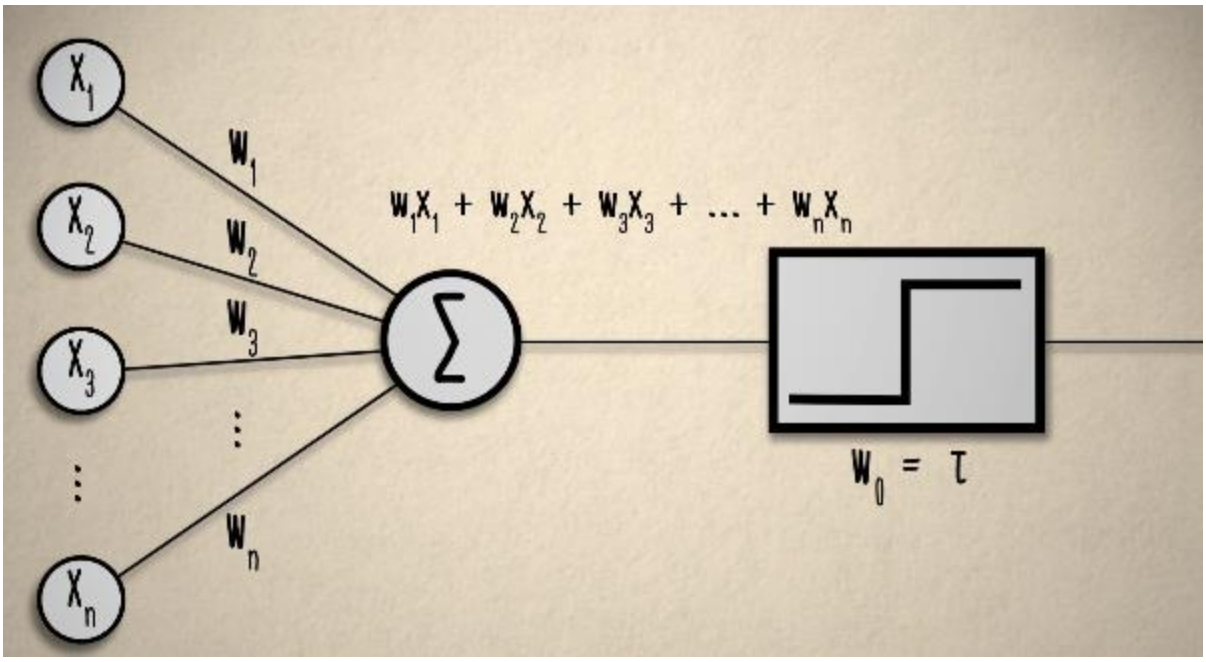
The current best algorithms achieve error rates less than 5%, and amazingly, perform about as well, if not a little bit better, than humans do.

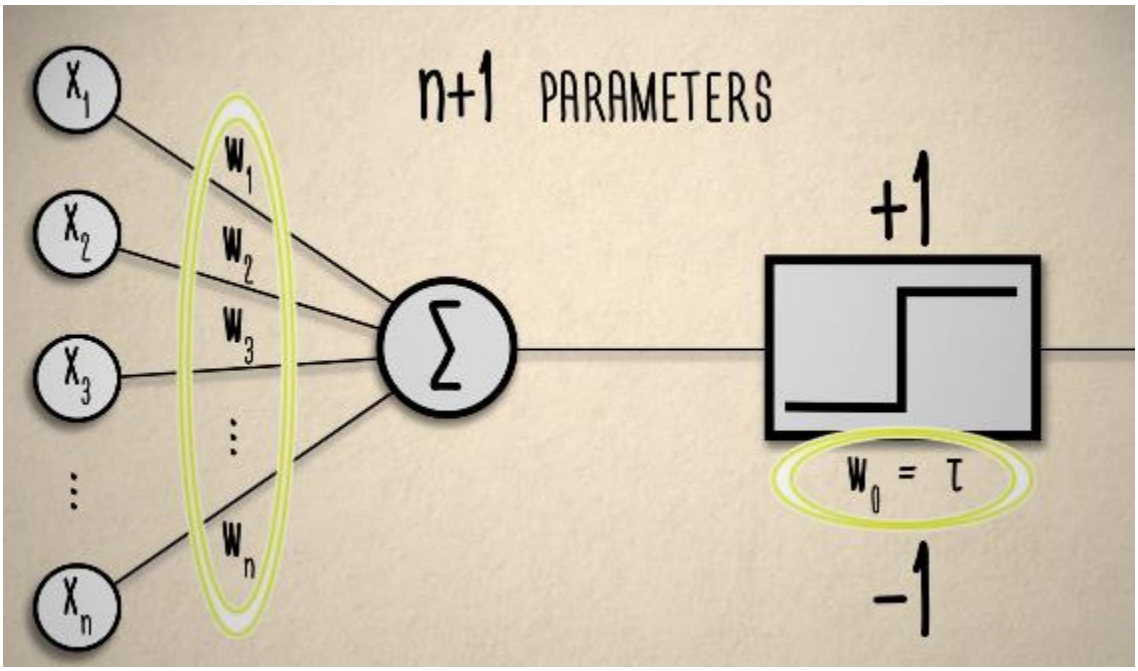
3.2.2 Classification using a single linear threshold (perceptron)



Whenever someone mentions the term deep learning, you should immediately think layered learning. That's what deep learning is about, building complex, hierarchical representations from simple building blocks.

Let's start with the building blocks. In fact, when we covered support vector machines, we already covered what's a common building blocks of deep networks. Remember the classifer itself? That's sometimes called a perceptron.





A perceptron is a function that has several inputs and one output. Let's say that it has n inputs just to fix some parameters. Then the output of a perceptron is computed in two steps.

In the first step, we compute a linear function of the inputs. The coefficients of the linear function are called weights.

In the second step, we take the output of the first step and compute a threshold. This threshold takes any value above some cut off tao and maps it to the value +1. And maps everything below tao to -1.

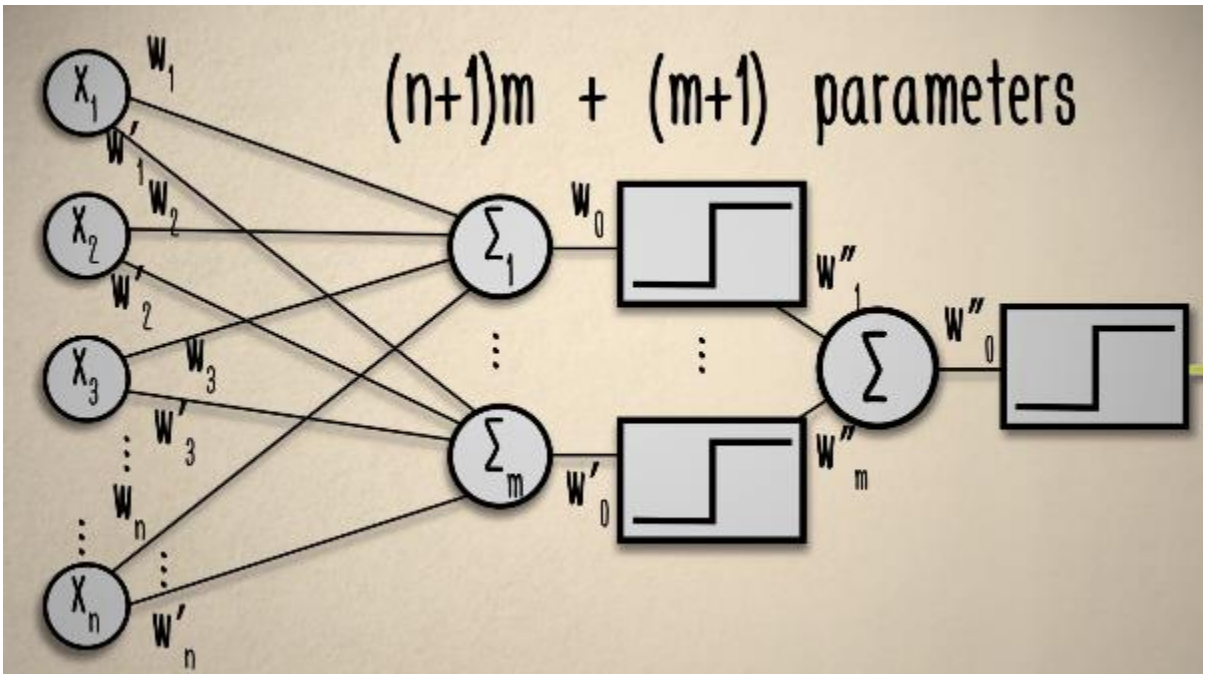
The second step is the only non-linearity.

Now just to make sure that the previous definition was clear, how many parameters define an n input one output perceptron? While we need n weights and one threshold tao, so in total that's n+1 parameters.

Now what's the connection to support vector machines? The class fire that we learned was itself a perceptron. As a class fire it only recognizes linear patterns. And to combat this, we introduced the idea of mapping the input space to a new space using a kernel.

In deep learning, we will take a very different approach where we layer perceptron on top of each other to get our non-linearities.

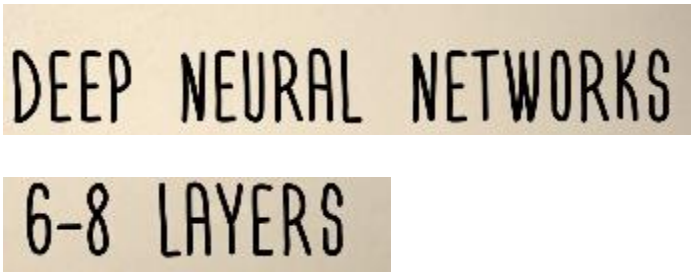
So how can we create more complex functions out of perceptrons as building blocks? Well we can layer them.



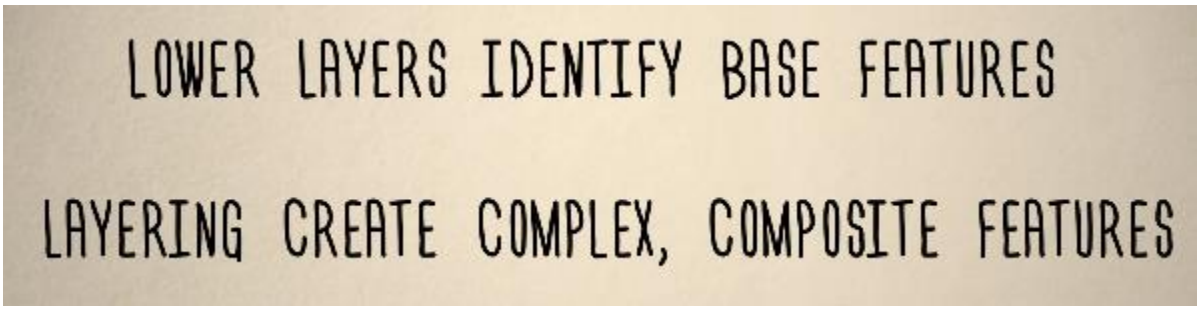
In the simplest set up, imagine that there are just two layers of perceptrons. Our input is an end-dimensional vector representing a picture. Each perceptron in the first layer has n inputs, each with its own weight. So each perceptron in the first layer computes its own thresholded linear function.

Now comes the interesting part. If we have n perceptrons in the first layer we can have a single perceptron in the second layer, which takes as its inputs all of the outputs of the other n perceptrons. So how many parameters define this model? Well, we have (n+1)m parameters for the first perceptrons in the first layer and (m+1) parameters for the perceptron in the second layer.

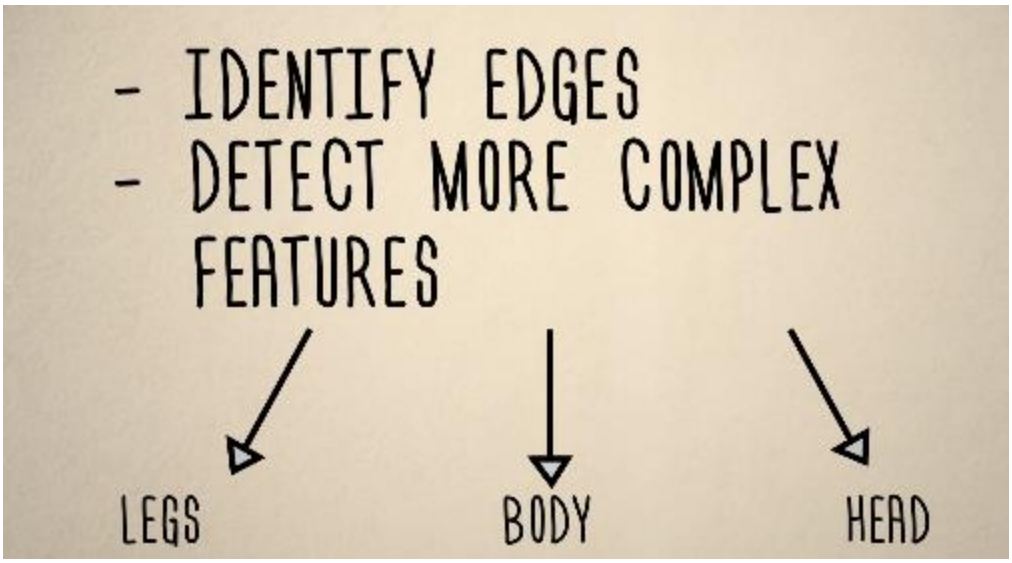
What's important is that the overall function we've computed that takes in an n dimensional input factor and outputs a plus one or minus one is much more interesting and complex than the type of function we got from a single perceptron. We can take this idea much further and have more than two layers.



The functions that we get are called deep neural networks, and in practice one usually sets them to have between six and eight layers and millions of perceptrons in between.

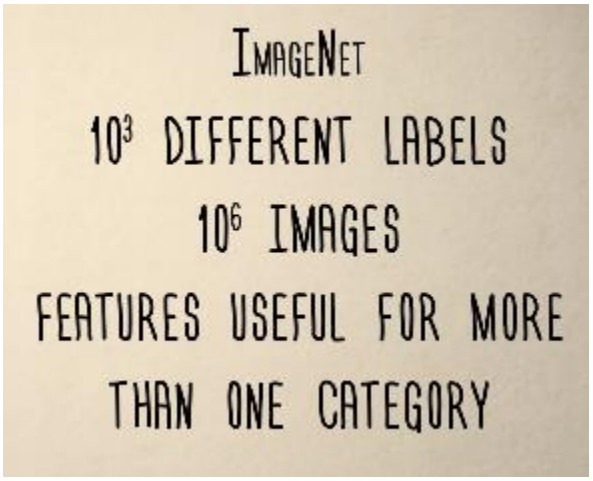


There are several fragments of intuition behind these types of functions.
The hope is that the lower level layers of the network identify some base features, like edges and patterns, and that each layer on top of them builds on the previous layer to create much more complex, composite features.



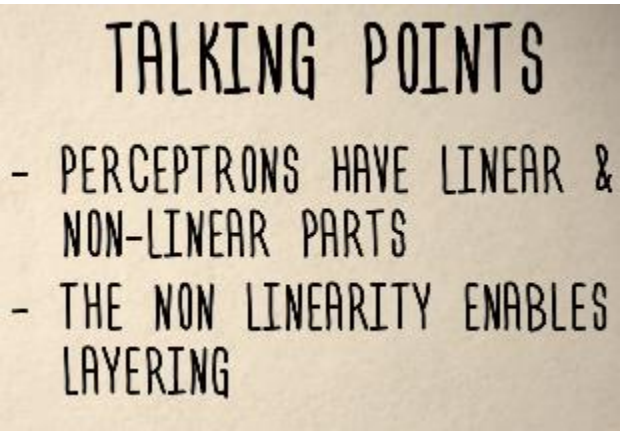
The intuition is how might you build up a feature that represents whether or not a dog is present in an image?

First you would identify its edges. And then you'd identify which collections of arrangements of edges represent legs and which represent the body and the head and then which arrangements of those parts represent the dog?



Recall that in ImageNet we want to correctly classify according to 1,000 different labels at once.
Even though there a million total images, that's not actually that many examples per label. So what's important is that the features that are useful for identifying one breed of dog can be useful in identifying other breeds of dog as well.

In this sense, a deep network can have a thousand outputs, one for each label, built on top of a common deep network underneath it, one which is hopefully identifying useful, high level representations that are needed to understand images.



First, the perceptron has a linear and non-linear part, the threshold function.
If it wasn't for the threshold, creating deeper networks would not buy us anything.
We would still be composing linear functions.
And no matter how deep we make the computation, the function we get would still be linear.

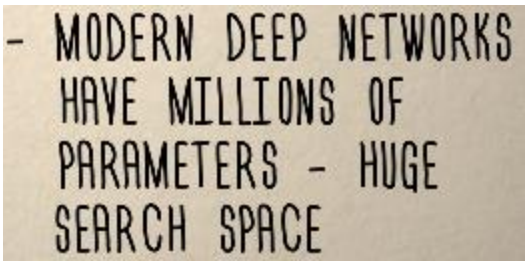
It's the non-linearity that we added that makes deep network so functionally expressive.

In fact, there are many other non-linear functions that we could have chosen instead of a threshold.

We could have chosen a logistic, sigmoid, or hyperbolic tangent, or any other smooth approximation to a step function. This and many other aspects of the architecture of deep neural networks are all valid design choices that have their own merits.

There are many research papers that grapple with issues like which non-linear functions work the best, and how should we structure the internal layers for example, using convolution.

We won't talk more about these issues, but it's good to know that they're very important.



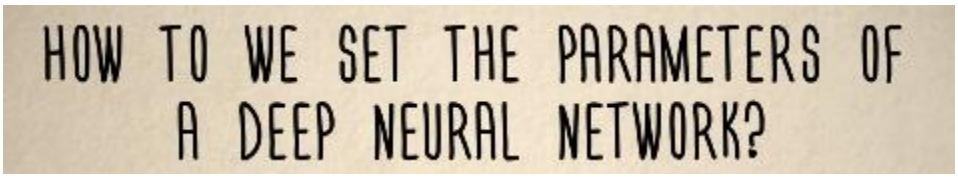
Second, we've said nothing about actually finding the parameters of a deep network. Modern deep networks have millions of parameters, which is a very large space to search over.

When we talked about the support vector machine, we had the perceptron algorithm, that told us if there is a linear class file, we can find it algorithmically.

But even if there is a setting of the parameters of the deep network, that really can classify images accurately, into, say different breeds of dog, how can we find it?

There is no simple answer to this question. There are approaches that seem to work in practice, but why they do it is still very much a mystery and perhaps a phenomenon that has to so with some of the strangeties of searching in such a high dimensional space.

3.2.3 Hierarchical Representations



Our first goal is to cast the problem of finding good parameters as an optimization problem.

There is an important class of optimization problems called convex optimization problems for which we have many good algorithms, and we understand quite well when and why they work.

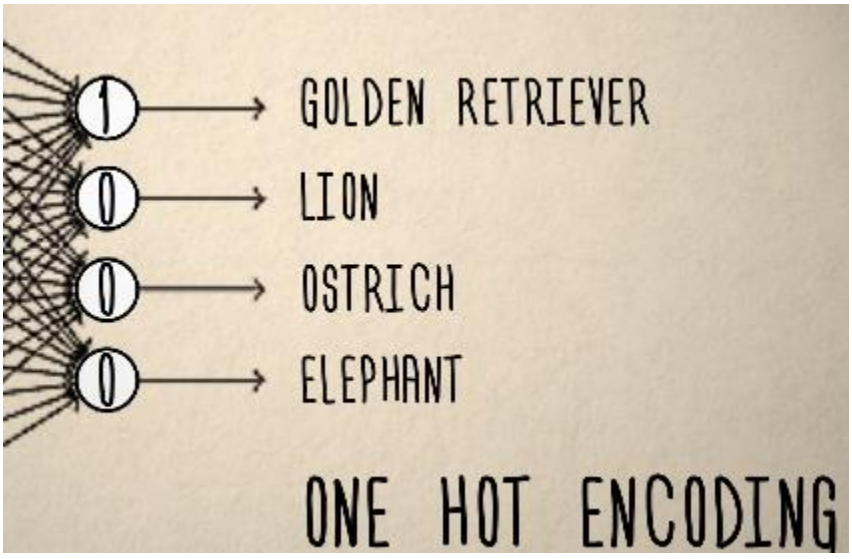
But life is not so easy in the case of deep neural networks.
The optimization problems that come out will be non-convex and unwieldy.

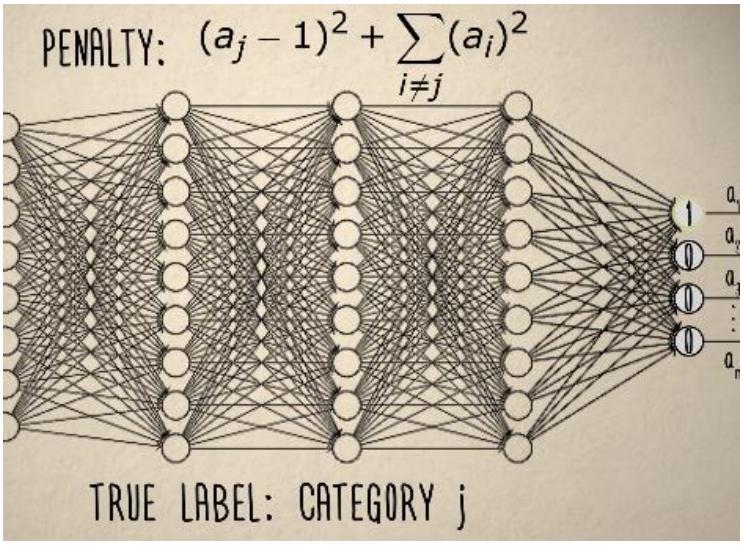
We'll talk more about this issue later, but first, let's work towards defining the optimization problem we'll be interested in.

Returning to the image net example, our deep neural networks have n inputs and 1,000 outputs, one for each category that we'd like to assign images to.

What we're hoping for is that when we feed in a picture as an n dimensional vector, that the output in the last layer contains a 1 in the correct category and 0's in all the other categories.

This is sometimes called the one hot encoding.





Now, what if you give me all the parameters of a deep network?
How could I evaluate how good these parameters are?

I could take each one of the 1 million examples in ImageNet, feed each picture into the network, and check whether you got the correct label.

To be concrete, let me introduce what's called the quadratic cost function.

Let's say the input is some vector x and it's true label is category j, then the output that I would like to get is all 0's except for a 1 in the j of output bit. Now, I could penalize you by how far off your output is from this idealized output.

If on input x, you output a1, a2 up to am. As your m output bits, I could take as a penalty aj minus 1 squared, plus the sum of all the other ai's squared.

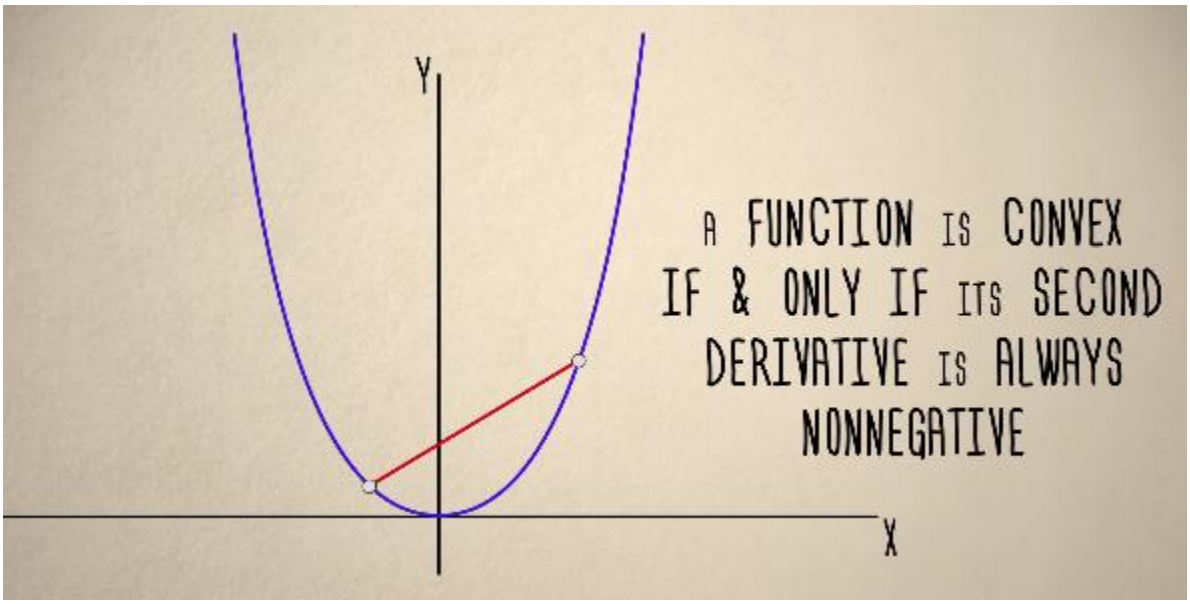
This is a function that evaluates the 0 if you get exactly the correct output, and evaluates to something non-zero if you get the wrong category. What we've done is we defined a cost function.

Now if we want to find a good setting of the parameters, why not look for the setting of the parameters that minimizes this average cost.

This is an optimization problem.
We have an explicit function that depends on the parameters, as well as the training data, that we'd like to minimize.

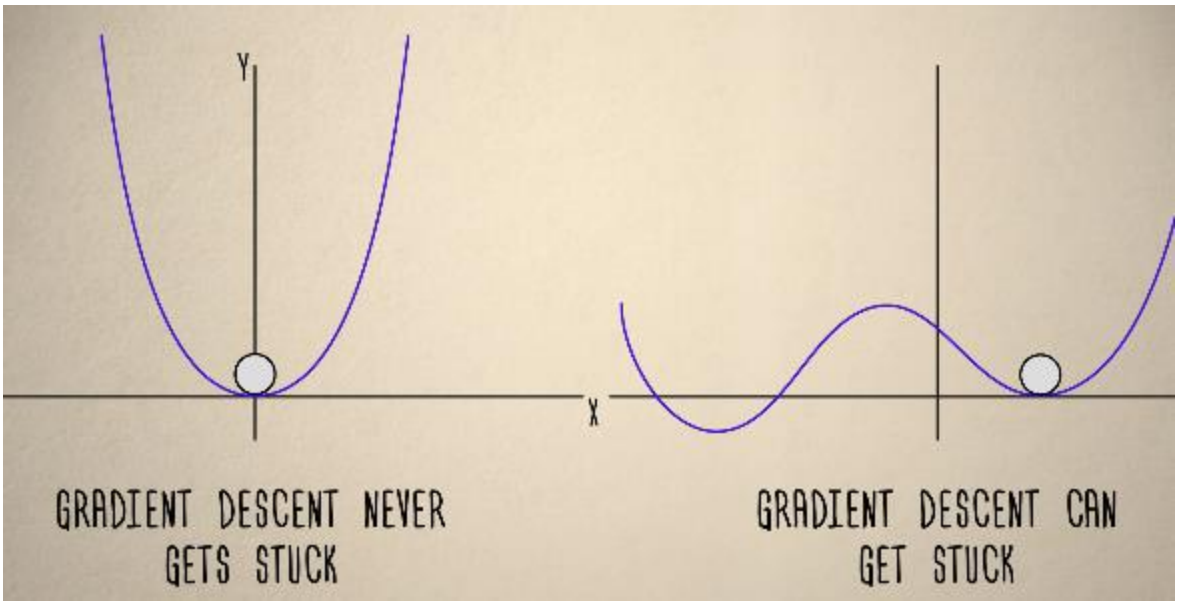
It turns out, that if you find a setting of the parameters that works well on the training examples, i.e. Has low average cost, it achieves low error on new sets of examples that you give it too.

This can be made mathematically precise, but it's too much of a digression to get into here.
In any case, we have what we're after.
We have an optimization problem that, if we could solve, would find good parameters.



The important point is that convex functions have no local minima that are not also global minima.

There is nowhere that you can get stuck if you're greedily following the path of steepest descent that isn't actually the globally optimal solution. But in the non-convex case, you absolutely can get stuck.

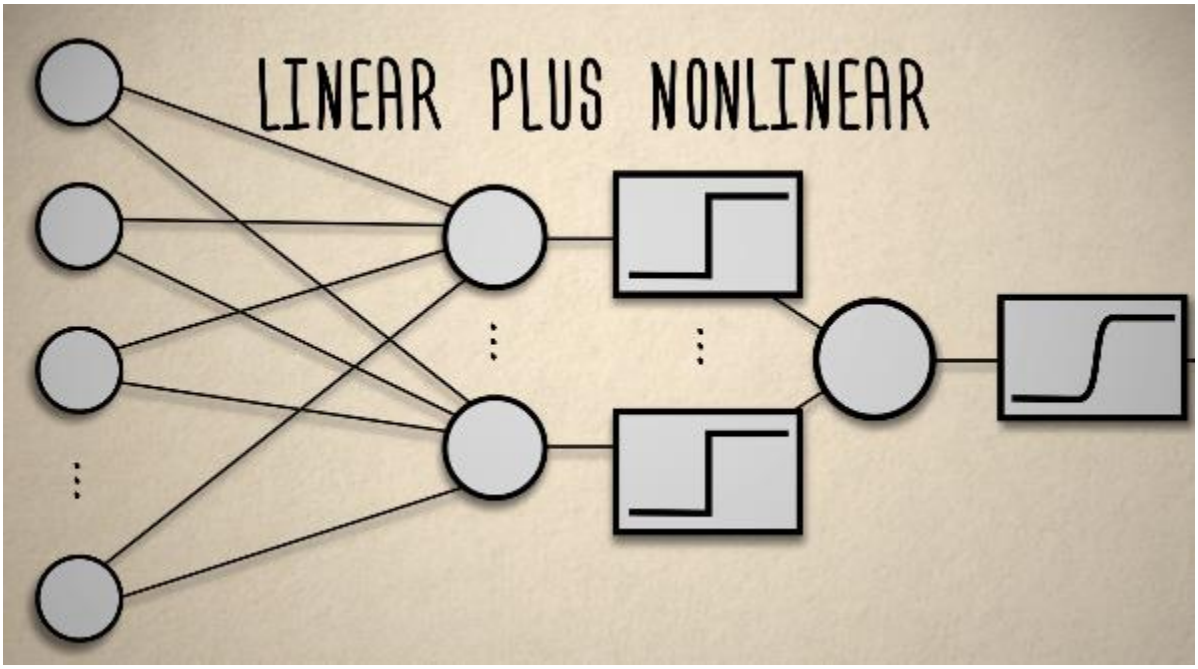


So what we're going to do is use gradient descent to find the best parameters for our deep neural network, even though the function we're trying to minimize is non-convex.

We're taking an algorithm that's guaranteed to work in the convex case, that we know does not always work in the non-convex case, and using it anyways.
One of the greatest mysteries is that it still seems to work.

What it finds is not necessarily the globally optimal solution, but even the locally optimal solutions it finds are, seemingly, still good enough.

3.2.4 Fitting parameters using backpropagation

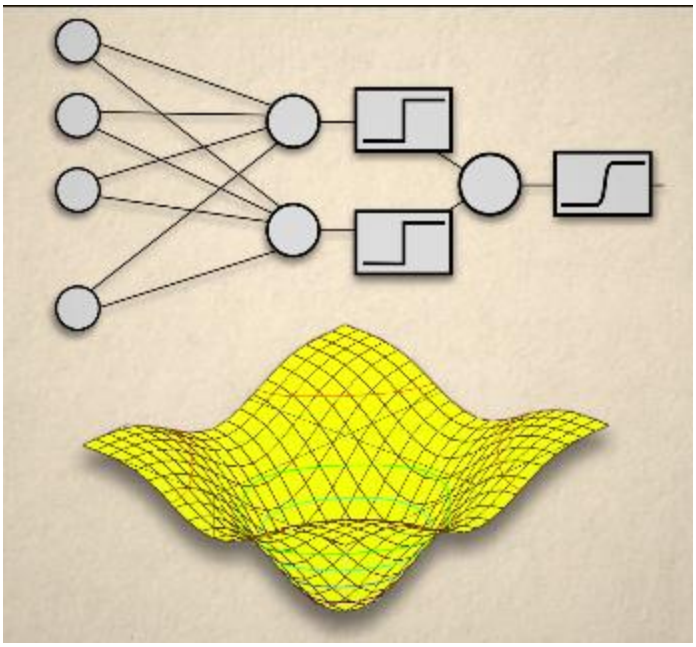


Now let's talk about how to apply gradient descent to a deep neural network. We have a cost function that we'd like to minimize. Let me mention a technicality that we haven't addressed so far.

When we talked about perceptrons, we described them as a linear plus nonlinear part. We took the threshold function to be our nonlinearity.

But there are many other reasonable choices like a sigmoid function. For what we talk about next, you should think sigmoid. Our perceptrons compute a linear function of their inputs, using their weights, adding a constant term called the bias and feed this value into a sigmoid function to get their output.
The reason it will be important to work with sigmoids is that these functions are smooth and de-frenchable. Or as the derivative of a threshold is everywhere either 0 or undefined.

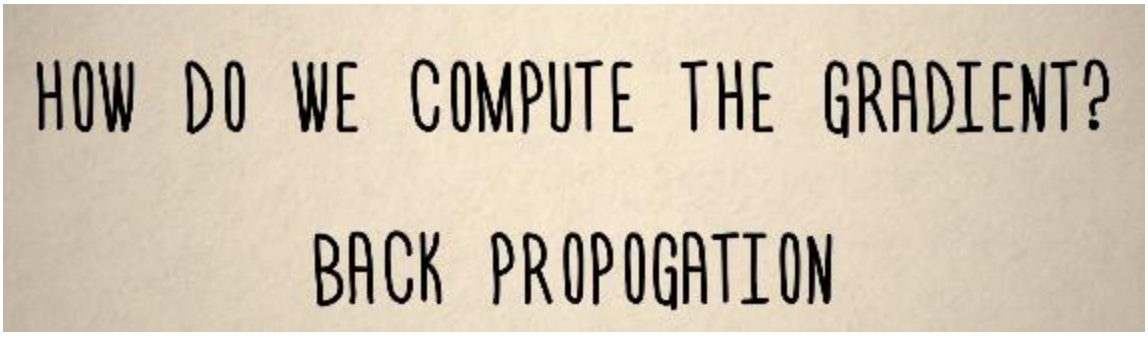
In any case what we've talked about so far is using grading decent to minimize or approximately minimize a non-convex function.



Our functions come from computing the quadratic cost function on the output of the last leg.
And it depends on all of the weights and biases inside the network.

The gradient is a measure of how changes to the weights and biases change the value of the cost function.

And we know we want to move in the direction opposite to the gradient because we want to minimize the cost function.



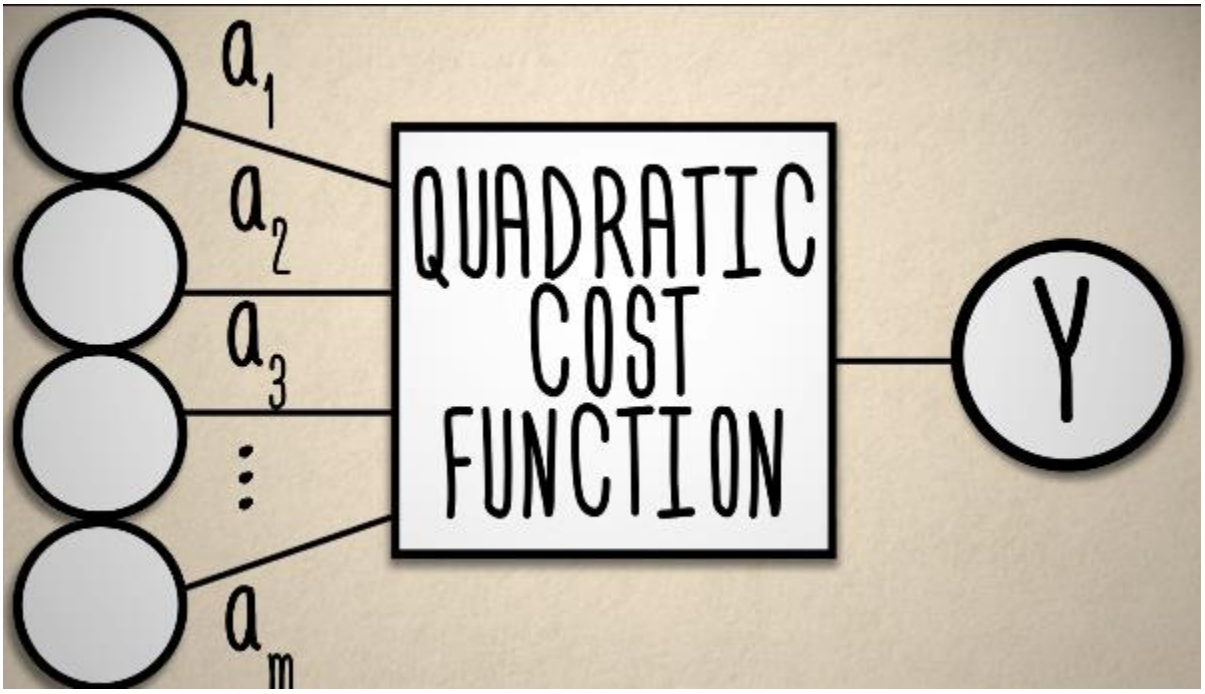
The main question now is, how do we compute the gradient?

In the context of neural networks, we can compute the gradient using what's called back propagation.

The equations that define back propagation are quite a mouthful, and involve a lot of matrix vector notation.

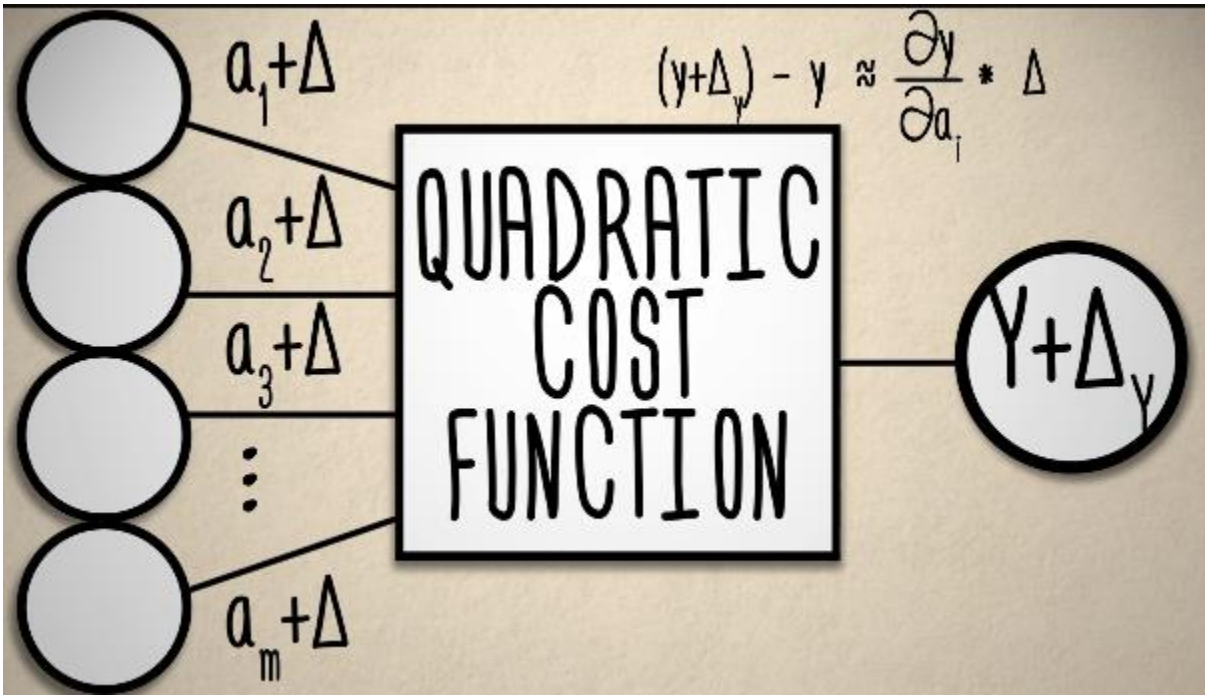
The intuition is much simpler, though. What back propagation is really doing is using the chain rule to compute the gradient layer by layer.

Let's get into the intuition. Imagine in the M outputs of our our neural network are a1, a2 up to am.

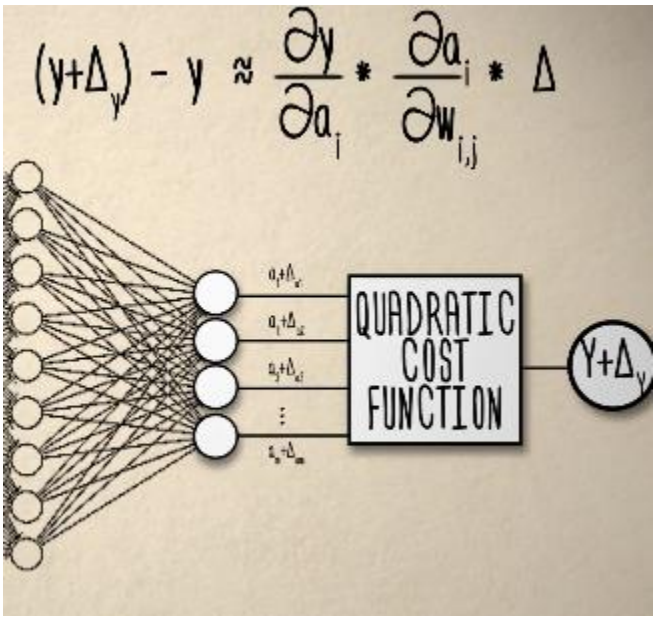


And these are parameters that we feed into the quadratic cost function.

Then it's straightforward to compute the partial derivative of the cost function with respect to any of these parameters.

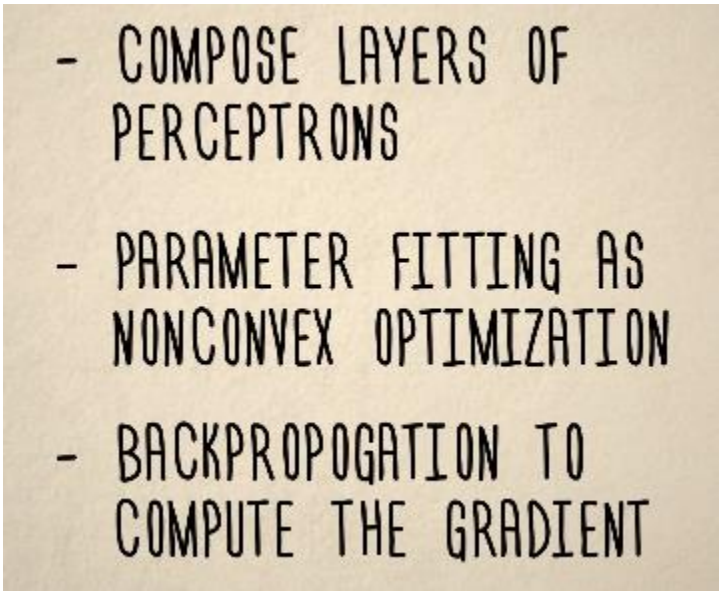


Now the idea is that these m outputs are themselves based on the weights in the previous life.
If we fix the IF output, it's a function of the weights coming into the ith perceptron in the last layer, and also of its bias. We can compute again how changes in these weights and biases affect the ith output.



This is exactly where we need the nonlinear function, in our case, the sigmoid, to be differentiable.

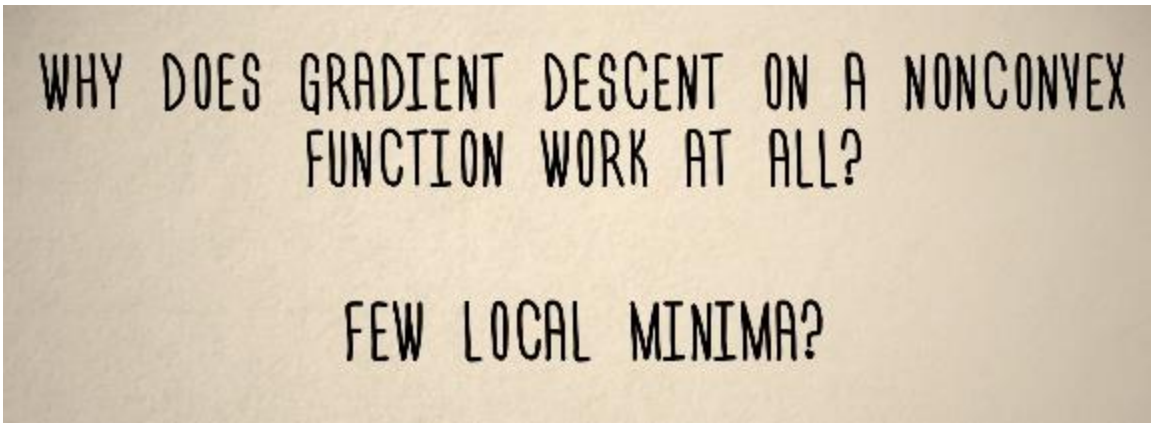
Back propagation continues in this manner, computing the partial derivatives of how the cost function changes.
As we vary the weights in the layer based the partial derivatives that we've computed for the weights and biases in the layer above it. That's it, that's back propagation.



Now we have all the tools we need to apply deplar.
We talked about how to build up complex nonlinear functions from perceptrons as building blocks.

We talked about how to cast the problem of fitting its parameters as an optimization problem, albeit a non-convex one.

And we talked about how to compute the gradient in a layer wise manner using back-propagation.



Why does gradient descent on a nonconvex function work at all?

In lower dimensions, it seems obvious that it really does get stuck.
The truth is that no one knows why it seems to work in high dimensions.
There's several possible explanations. Maybe these functions are closer to convex than we think.

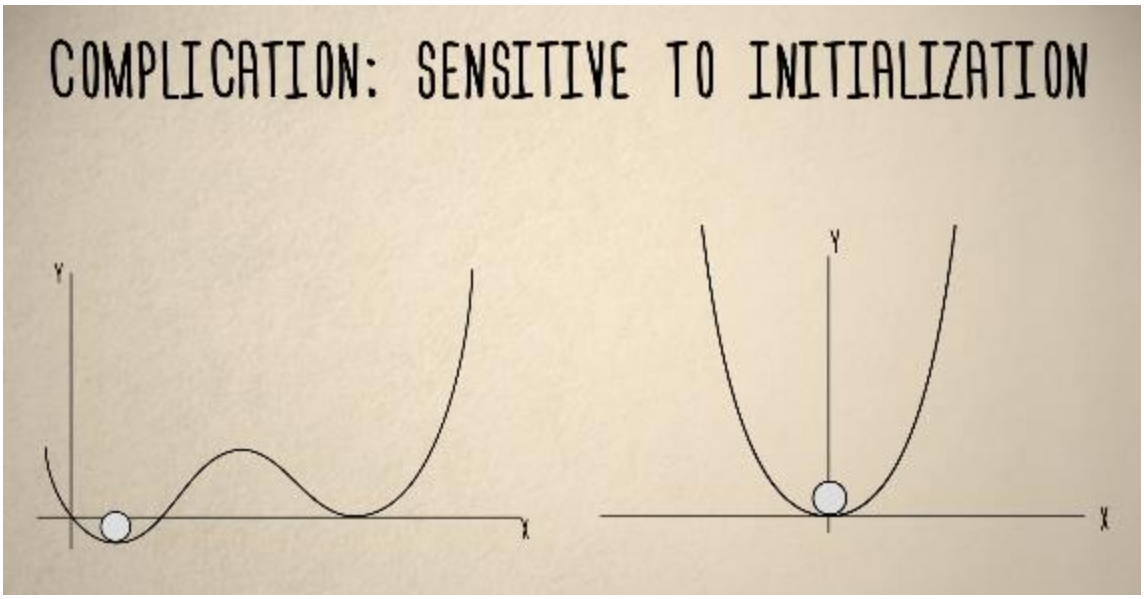
And are at least convex on a large region of space. Another factor is that what it means for a point to be a local minimum is much more stringent than higher dimensions.

A point is a local minimum if every direction you try to move him, the function starts to increase.

Intuitively it seems much harder to be a local minimum in higher dimensions because there are so many directions for you to escape from.

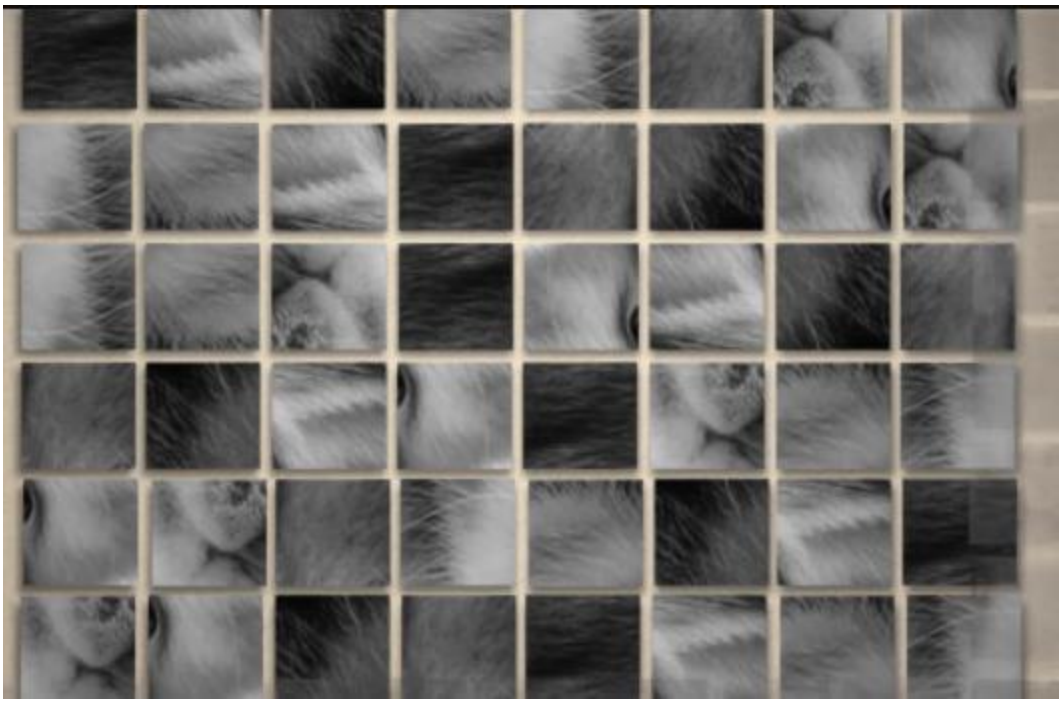
Yet, another take away from this discussion is that when you apply back propagation to fit the parameters, you might get very different answers depending on where you start from.

This just doesn't happen with convex functions.
Because wherever you start from, you'll always end up in the same place - the globally optimal solution.



3.2.5 Convolutional Neural Networks

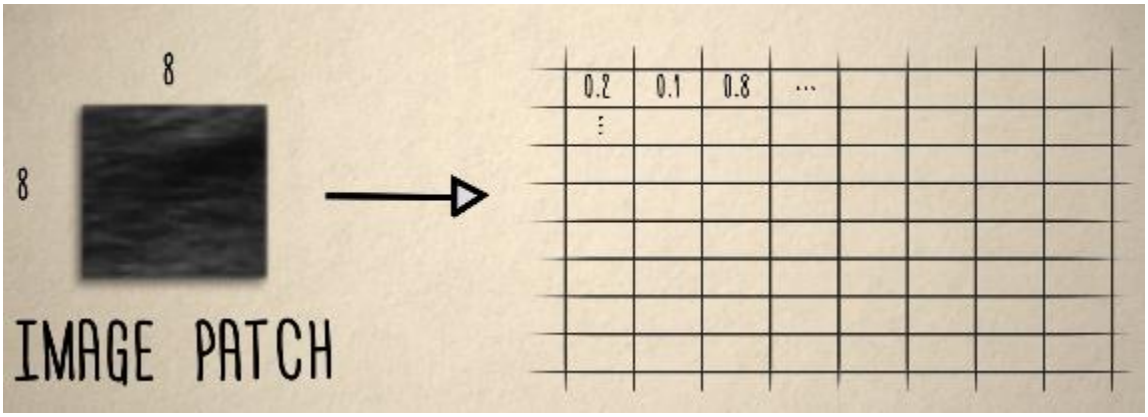
The first and arguably most important is the notion of the convolutional neural network.
Let me explain how the first layer works.
Let's say that our input is a grayscaled image that's 256 by 256.
We can break it up into 4 by 4 image patches.
In total we have a 64 by 64 grid of these patches that comprise the image.

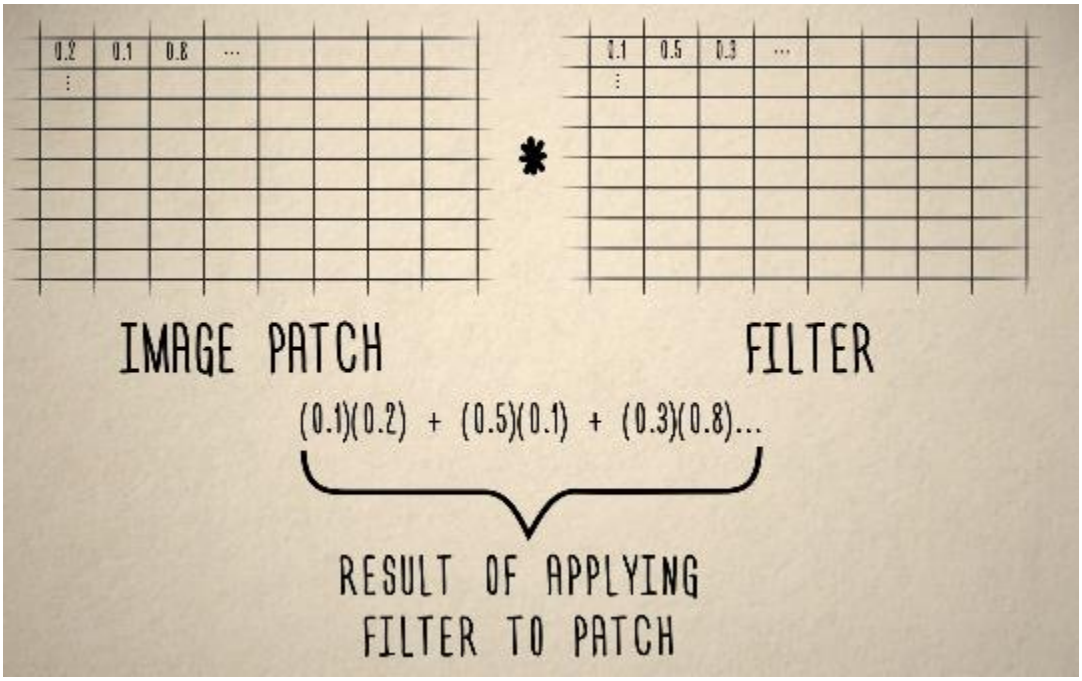


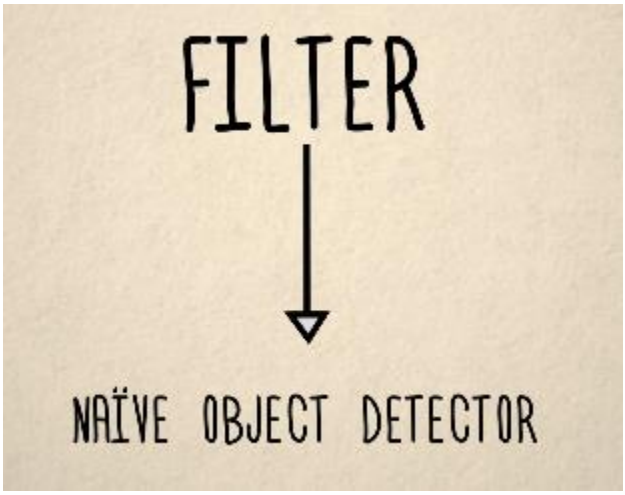
Now let's consider a linear function on just the 8 by 8 image patch.
Sometimes this is called a filter, in our case a filter is an 8 by 8 grid of weights.

And when we apply a filter to an image batch what we do is we take an inner product between them as vectors.

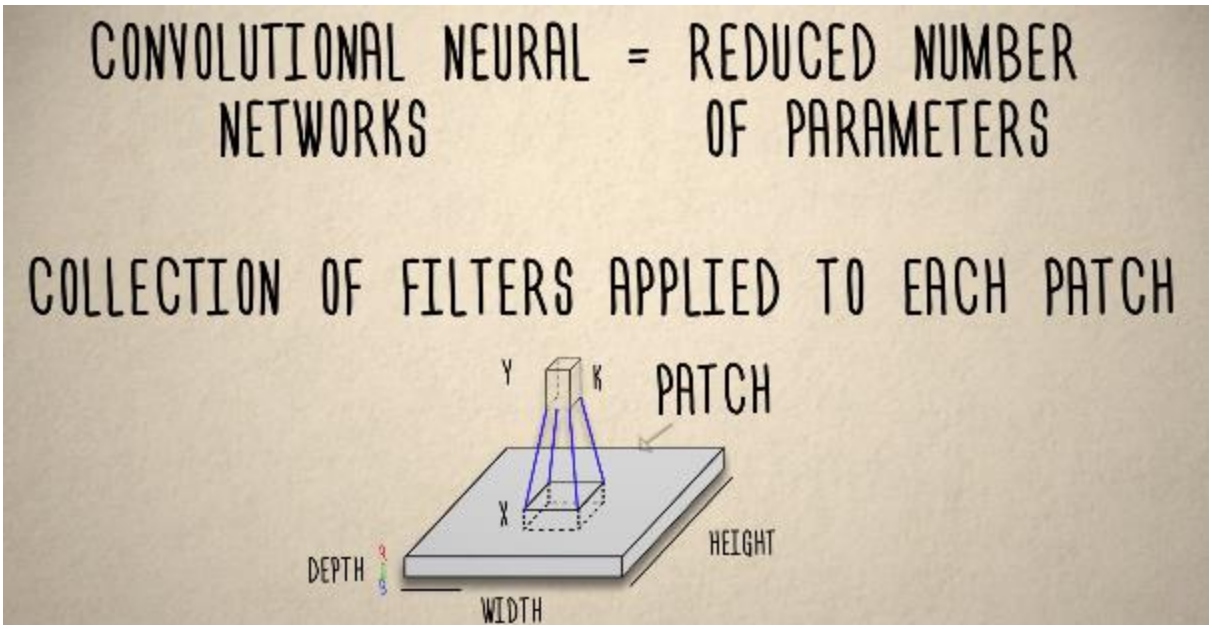
So if we apply a filter to each one of the patches we get a new 64 by 64 grid of numbers.
What good is this? Remember the intuition is that the first few layers detect simple objects like edges.
We can think of a filter as a naive object detector.
And instead of having it have different parameters when we apply it to each of the different image patches.







Why not have the same parameters throughout?
This drastically reduces the number of parameters and consequently there are many fewer things to learn.



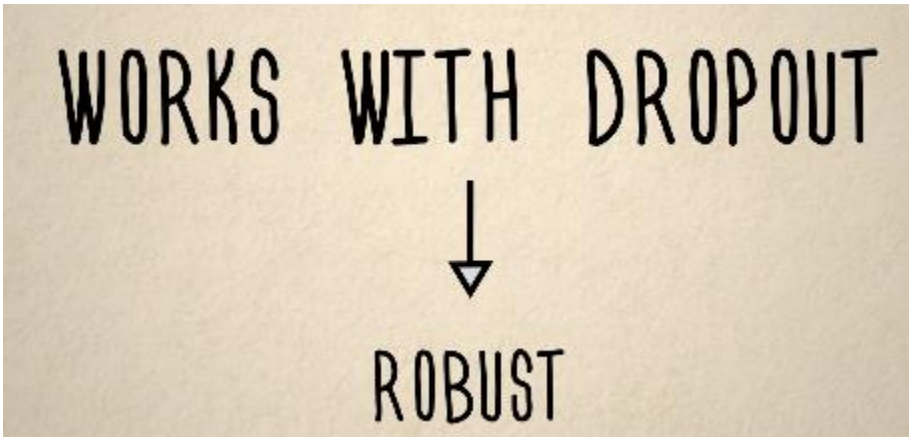
If we take this idea to its natural conclusion we get convolutional neural networks.

In the first layer, we have a collection of filters, each one is applied to each of the image patches.

And together, they give the output of the first layer after applying a non-linearity at the end.

This is already a major innovation because it means we can work with much larger neural networks.

In practice, just the first few layers are convolutional, and the others are general and fully connected.



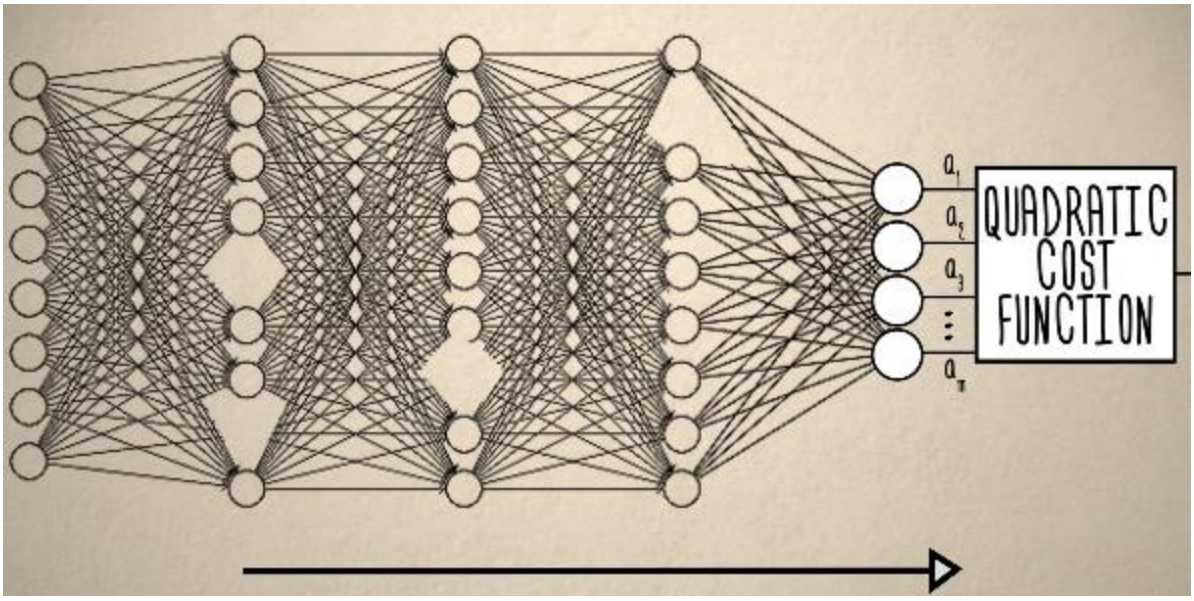
Another important idea is the notion of dropout.

Here when we compute how well a neural network classifies some image, say through the quadratic cost function.

We instead randomly delete some fraction of the network and then compute the new function from the input to the outputs.

The idea is, if a neural network continues to work even if we drop perceptrons from the intermediate layers.

Then it must be spreading information out in a way that no node is a single point of failure.



Training a neural network with dropout makes the function we learn more robust. This is not a precise statement, but it's the prevailing intuition.

3.2.7 Limitations of Deep Neural Network

The truth is that we don't really understand why things like gradient descent work so well and even then, the actual features that it finds are difficult to interpret.

We can't readily take a deep neural network that we've learned and see why it's working so well.

HOW MUCH DO WE HAVE TO CHANGE THE IMAGE TO CHANGE THE LABEL THAT THE NETWORK ASSIGNS?