Generalization and Cost Function

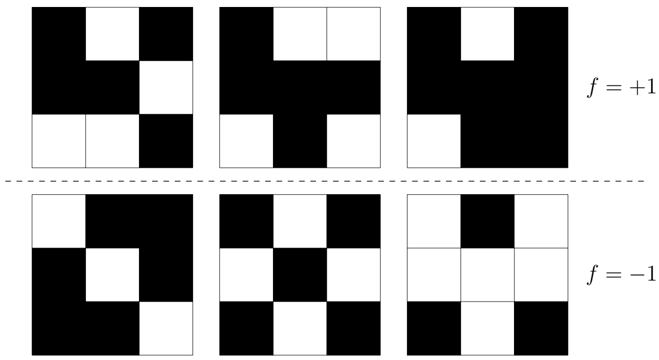
Intuition from Last Exercise

- Perceptron is just having a straight line and assign everything above to one. class and everything below to another class.
- The math is simple $h(x) = \text{sign}(\mathbf{w}^T\mathbf{x})$. Don't forget to pad the $\mathbf{x}$.
- The line is defined by the relation $\mathbf{w}^T\mathbf{x} = 0$. What do you expect the boundary that separate positive and negative, it must be zero.
- The learning algorithm is there to pick the "best" line.
- The more training data you have the more chance that your E_{out} will match E_{in}. We hope that our hypothesis will "generalize" $E_{\text{in}} = E_{\text{out}}$.
- The less complicated your model is the more chance that your E_{out} will match E_{in}. Intuitively, the less complicate your model is the chance is that you will actually learn instead of memorizing.
- However this doesn't tell you that E_{out} will be low. It just says that E_{in} and E_{out} will be close. That is they can both be terrible but they are closely terrible..

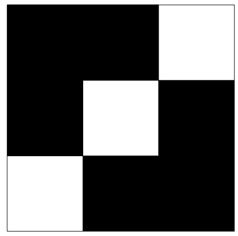
Feasibility of Learning

Learning is trying to find out the target function. The only information we have at the hand is the training data.

Consider the following puzzle.



It is impossible to say what is the value for



- One could say it is +1 if the lower left corner is white and -1 otherwise.
- One could say it is +1 if there exist a symmetry axis and -1 otherwise.

Which one is right? Both perform perfectly on the training dataset. The real world problem is not on the training dataset. It is how your hypothesis will perform *out of sample*

Is it possible for us to learn? Short answer is yes. The long answer is Probably Approximately yes. (This is a real mathematical term).

E_{in}, E_{out}, Generalization

To know exactly what we mean by learning, we need to understand the concept of *Generalization*. Generalization is all about how the *in sample* error compare to the *out of sample* error. We define the term error quite simply as the fraction of sample we got wrong. The in sample error is calculated by taking the ratio of the number of sample we got wrong class for it and the total number of sample

$$E_{\text{in}} = \frac{\text{\# of training sample we misclassify}}{\text{Total \# of training sample}} \quad (1)$$

The out of sample is defined similarly.

$$E_{\text{out}} = P(h(x) \neq f(x)) \quad (2)$$

This E_{out} is an unknown quantity signifying the probability that the hypothesis(h) will not match the real target function(f) in real world.

It is very important that you understand the different the two. Exercise 1 should give you a very good idea of the difference between the two.

When we got our hypothesis from the learning algorithm, we hope for two things

- E_{in} is low. This means that our hypothesis performs well at least in the training sample.

- The hypothesis will generalize $E_{\text{in}} \approx E_{\text{out}}$. That it will continue to perform well *out of sample*.

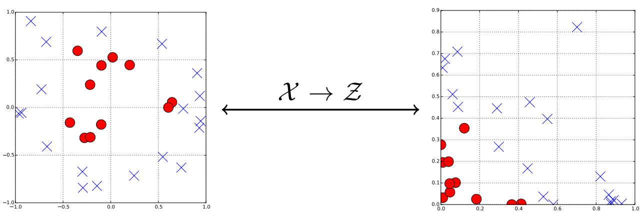
Can you tell me what to do so that the chance that E_{in} and E_{out} match is higher?

Remember when we do pattern recognition we only use E_{in} as a guideline. E_{out} is the real thing we are trying to make it low.

Non Linear Transformation

Enough for the theory, now let us go back to practical stuff.

Not all target function is linearly separable. But we can sort of making it more linearly separable by doing a non-linear transformation.



For example, in the figure above, the $\mathcal{X}$ space on the left is not linearly separable. But, looking at it, we can see that ok the classes has something to do with the distance from teh origin. So, transforming (x_1, x_2) to $\mathcal{Z}$ space might be a good idea. We can do this by just using $Q : \mathcal{X} \to \mathcal{Z}$:

$$Q(\mathbf{x}) = \begin{bmatrix} x_1^2 \\ x_2^2 \end{bmatrix} \quad (3)$$

After we transform our data from $\mathcal{X}$ space to $\mathcal{Z}$ space it is clear that we can just use linear perceptron to separate the two.

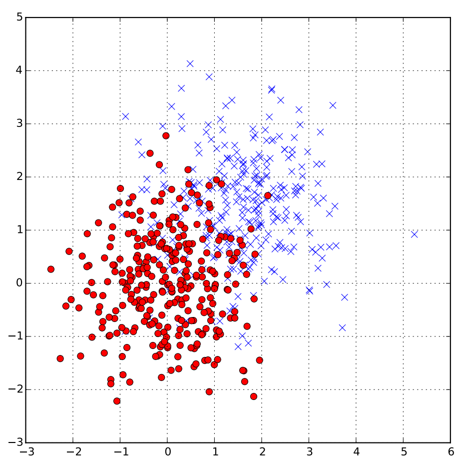
Your homework is to design a program that does this (and plot it out). This is an excellent programming exercise.

Noisy Target

In all of the application, your target function is not even deterministic. The same input vector may give you different output. For example, consider the credit card application, someone with the same salary, same sex, same debt and same number of dependent my have entirely different credit behavior.

The target function that we talk about is really target probability (density). The target probability is also a conditional one: $P(y|\mathbf{x})$. This reads the probability that the object of input vector $\mathbf{x}$ will be of class y. Or concretely the probability that a customer with $\mathbf{x}$ (salary, debt) would be good customer(y=+1).

An example of noisy target function is shown below



Pocket Algorithm

So far we have learn Perceptron Learning Algorithm which deals with linearly separable data.

If we apply Perceptron Learning Algorithm to non-linearly separable data. The algorithm will not stop. (Why?)

We can modify this very simply by just keeping(pocketing) the one with the best error rate as we go. Then we stop after number of iteration and return the best one we found so far.

See exercise.

Cost Function

In pocket algorithm, we choose the hypothesis with the lowest E_{in} defined as the fraction of training data we misclassify.

Of course, the final hypothesis must have something special. But, you may ask is this E_{in} really the thing we want to minimize?

The true answer is depends on the application.

Let us consider the a fingerprint algorithm.

It takes your fingerprints and it tells whether the customer is in the database or not.

There are two customers for this a supermarket and CIA.

Supermarket

The supermarket use fingerprint identification to give discount to loyal customer. There are 4 things that can happen.

1. A customer **x** is in the database and the fingerprint algorithm say that **x** is in the database.

2. **x** is **not** in the database but the fingerprint says that [x is in the database. This situation is called *false positive*. But such situation is not so bad for the supermarket. They just have to give a discount to another customer. No big deal.

3. **x** is in the database but the fingerprint says that **x** is not in the database. This is called *false negative*. For supermarket, this is quite a big deal since you will embarass a loyal customer. We really do not want this to happen. So we would penalize this by a lot.

4. **x** is **not** in the database and the fingerprint algorithm rejects **x**. This is good. It does what it is supposed to do.

Given the above requirement that we really don't want false negative. Intuitively, if you make your algorithm very strict, you will get less false positive but you will get more false positive vice versa. It is clear from the requirement that the false negative and false positive should not be penalized as equal. The false negative should be penalize a lot more. So once could construct a cost function that looks like the following:

	Accept	Reject
In the DB	0	1000
Not in the DB	10	0

This table can be translated to the the formula

$$E(h) = \frac{1}{N}(1000 \times \# \text{ of false negative} + 10 \times \# \text{ of false positive})$$

If we minimize this cost function, will will get $\approx$ the thing we want.

Note that given a fixed training data, the cost function is a function of hypothesis. Each hypothesis will give you different amount of false negative an false postive.

CIA

Now let us consider the same fingerprint application for CIA agents.

When the fingerprint does the right thing. Good, we like it. But, the real deal is when it misclassify. There are two ways to misclassify:

1. False negative. **x** is a CIA agents. He tries to log into his computer using fingerprint. The algorithm denies him. No big deal. He will just try again and if after 10th try it doesn't work he can just go ask IT dept to fix it.

2. False positive. $\mathbf{x}$ is not a secret CIA agent but a spy. If the system let the spy in, tons of information will leak and that is super bad for the CIA.

Given the above requirement, we really should want to penalize false negative heavily. One may come up with a cost function that looks something like:

	Accept	Reject
In the DB	0	1
Not in the DB	1000	0

which translate to

$$\begin{aligned} E(h) = &\frac{1}{N}(1 \times \# \text{ of false negative} \\ &+ 1000 \times \# \text{ of false positive}) \end{aligned}$$

Take Home Lesson

Cost function is something to minimize to get what you want so make one up to do what you need to do.

Next week we are going to learning to tell computer to minimize the cost function.