Other kinds of regression:

eg: exponential regression:

Eg: fit $y = Ae^{Bx}$ to

$\{(x_1, y_1), (x_2, y_2), \dots (x_N, y_N)\}$.

This could work:

minimize $F(A,B) = \sum_{j=1}^{N} \left(Ae^{Bx_j} - y_j\right)^2$

least squares fit:

$\frac{\partial F}{\partial A} = 0$, $\frac{\partial F}{\partial B} = 0$, solve for A, B.

equations won't be so nice.

However: $y = Ae^{Bx}$

$$\log(y) = \log(Ae^{Bx})$$

$$= \log(A) + \log(e^{Bx})$$

$$\log(y) = \underbrace{\log(A)}_{b} + \underbrace{B}_{m} x$$

instead, minimize

$$G(b,m) = \left\| \begin{pmatrix} 1 & x \end{pmatrix} \begin{pmatrix} b \\ m \end{pmatrix} - \begin{pmatrix} \log(y) \\ \vdots \end{pmatrix} \right\|^2$$

(A: $\begin{pmatrix} 1 & x \end{pmatrix}$, p: $\begin{pmatrix} b \\ m \end{pmatrix}$, y: $\begin{pmatrix} \log(y) \\ \vdots \end{pmatrix}$)

⇒ Usual linear regression formula (normal equation) — gives answer for b, m

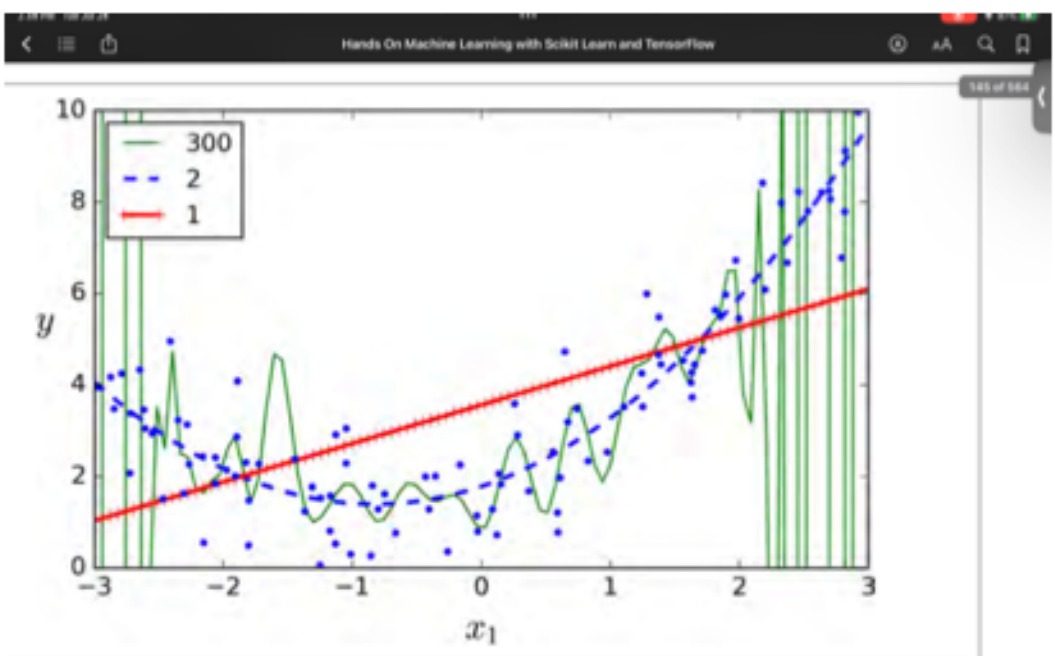
then $b = \log(A) \Rightarrow A = e^b$, $m = B$.

Issues with regression.

Underfitting vs. Overfitting:

example: Polynomial regression — find a degree k poly that fits data best.

→ better fit the higher the degree is.



4-14. High-degree Polynomial Regression

300 degree - overfit -

Can check using cross validation:

If MSE average fa cross validation is much bigger than the MSE using the training set — that's a sign that your model has been overfitted — ie. too many parameters.

A way to incorporate using a small number of parameters is to incorporate that into your cost function:

regularization ← adding parameters increases the value of the cost function.

eg. fitting $y = X \cdot \theta = \theta^T x$ to data (x_j, y_j)

X ← $\theta_1 x + \theta_2 y + \cdots = (\theta_1 \theta_2 \cdots)\begin{pmatrix} x \\ y \\ \vdots \end{pmatrix}$

Normal cost function:

$$C(\theta) = \sum_j \left| \theta^T x_j - y_j \right|^2$$

θ: coefficients to optimize

Regularized cost function

$$C(\theta) = \sum_j \left|\theta^T x_j - y_j\right|^2 + \frac{1}{2}\alpha \sum \theta_j^2$$

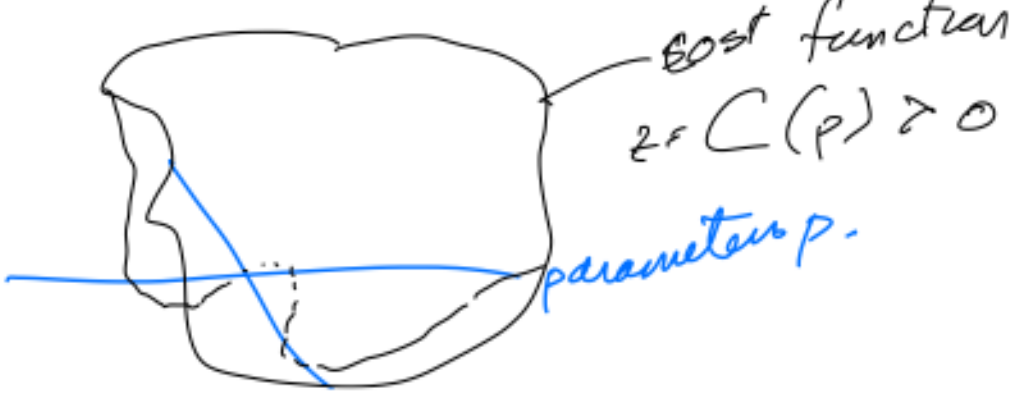
(α: constant.)

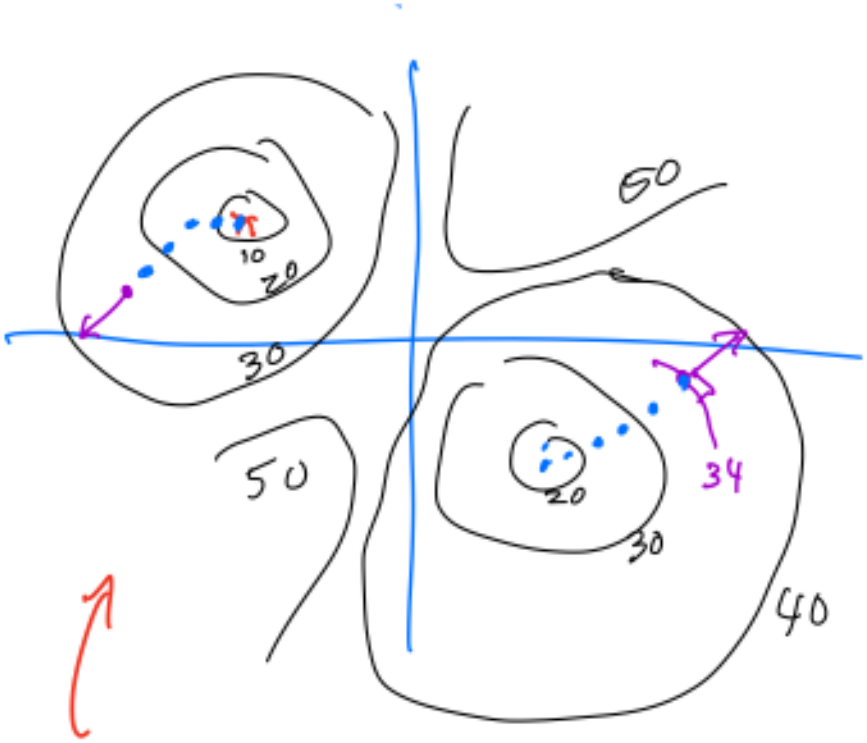
This will be minimized when $\theta_1, \ldots, \theta_N$ are small.

So MSE won't always be reduced by adding more parameters.

Other information can be gained from look at "learning curves" — see text.

How do we find minimums of cost functions when the functions are complicated? Numerical solutions?





At this point, the set of parameters would give us the minimum value of the cost function.

Gradient Descent Algorithm for finding the minimum of $C(p)$: pick a random starting point $p_0 = (p_{0,1}, p_{0,2}, \dots)$. Move in the direction of $-\nabla C$ to find the next point.

- Problems:
  - Might approach local min instead of global min.
  - Might bounce around too much if you travel too far at each step.

Typical way to approach gradient descent:

★ p_0 = initial pt. (randomly chosen).

$p_{k+1} = p_k + -\alpha \nabla C(p_k)$

α: probably a small number.

look for what p_{1000} is.

To avoid problems:

① have α get smaller with k.

eg $\alpha = \frac{1}{1+k}$.

② Do this several times (with random parameter changes) — pick the smallest $C(p)$ result. → prevent finding only a local min that is not a global min.

To speed up gradient descent:

- $\nabla C(p_k)$ is hard to calculate — uses entire dataset.

Instead: Use a subset of the points (chosen randomly to make a partial cost function — use to calculate approx. gradient).

eg, in a data set of 1000000 pts, choose 100 each time.

→ mini-batch stochastic gradient descent.
↑ random, discrete.

– can give pretty accurate results fast (even for linear regression).

Up to this point, we have wanted to predict y with

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_N x_n$$

or $f(\theta, x)$ ← some function

where y is a numerical quantity.

How do we use machine learning to classify data?

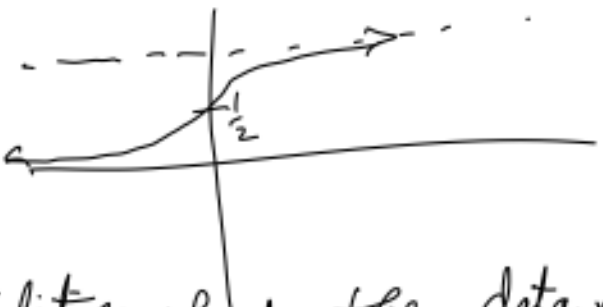
Given parameters θ, data x, determine the type of a data pt –

→ give answers between 0 & 1 for each type – giving the probability if it is

of that type or not,

Use logistic function — do "logistic regression"

$$\sigma(t) = \frac{1}{1+e^{-t}}$$



We get probabilities that the data pt. is k^{th} type is p_k

$$p_k = \sigma(\theta_k^T x)$$

could be $\sigma(A, \theta)$

θ_k: parameters used for the k^{th} type.

x: data.

Data $y_{k,j}$ = j^{th} data pt.

$$y_{k,j} = \begin{cases} 1 & \text{if pt is type } k \\ 0 & \text{otherwise.} \end{cases}$$

want $p_{k,j}$ to be close to 1 if $y_{k,j} = 1$

want $p_{k,j}$ to be close to 0 if $y_{k,j} = 0$.

$-\log(p_k)$ ← if $p_k \approx 0$ then this is large.

Cost function: $\sum -y_{kj} \log(p_k) - (1-y_{kj}) \log(1-p_k)$

Babb Stochastic Gradient seach for min →

- find best parameters -

then our estimated

$$p_{kj} = \sigma(\theta_{kj} x_j)$$

↑ should be close to 1 when data pt is of type k_j close to 0 otherwise.

What is the prediction?

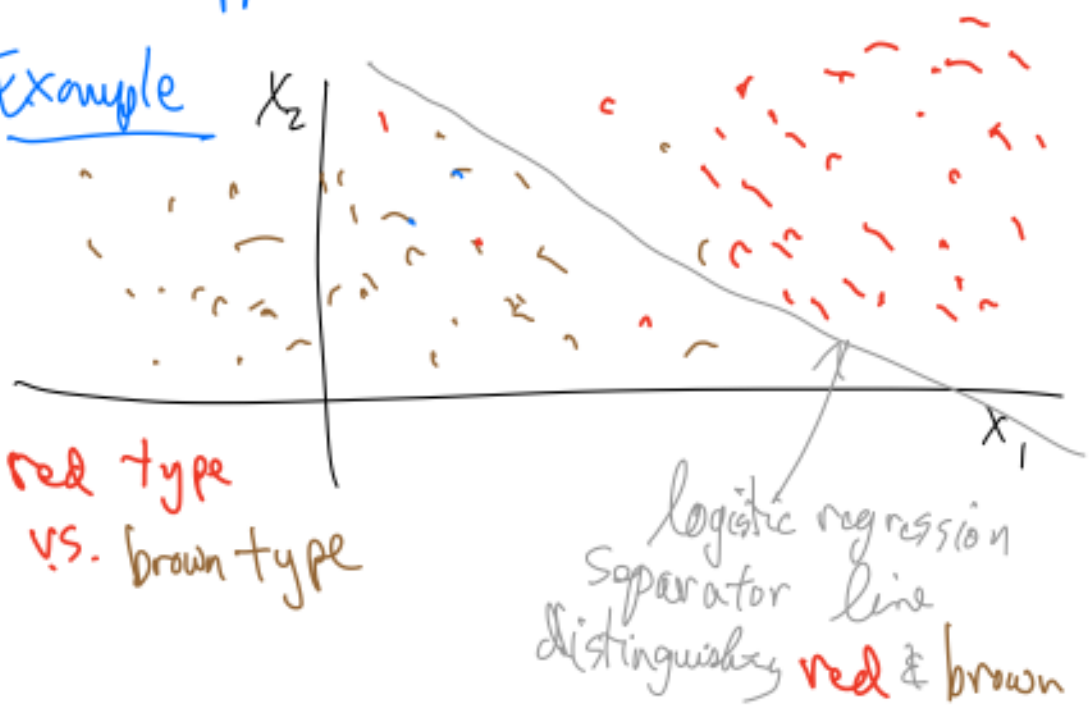
For each j, the predicted type is the k for which p_{kj} is greatest.

Those types of regression classifiers – all have the same idea as the above.

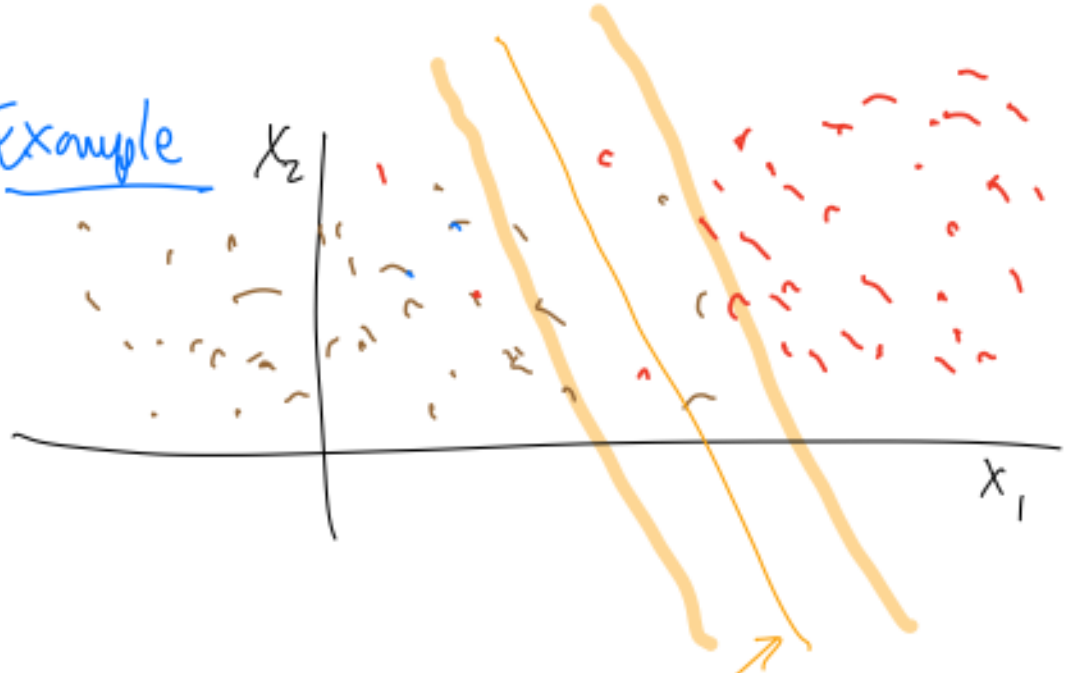
Another type of Classifying Algorithm:

Support Vector Machines.

Example



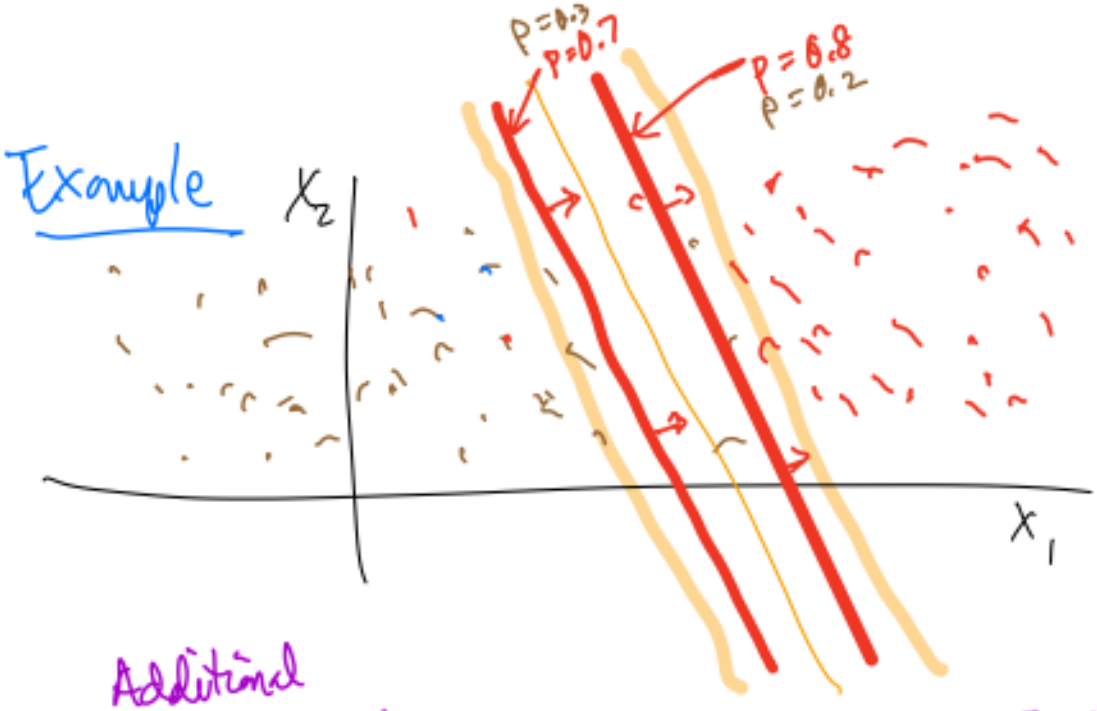
Example



Support Vector Machines classification line.

Goal: find the best middle line of a highway that separates the data (mostly) with the widest possible width.

parameters for support vector machines — how wide to make the highway, smallest number of outliers. You can associate different levels of costs for these things.



Additional

Information we can get from SVM

→ we can make parallel lines that give probabilities that a new data point would be of red type or of brown type.

Another technique: K-nearest neighbors: For each new data pt, find the k closest data pts in training set → choose the most common type as the type of your data pt.