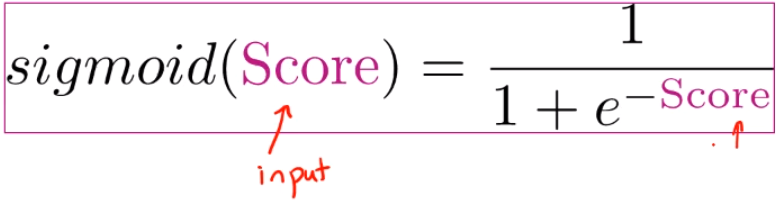
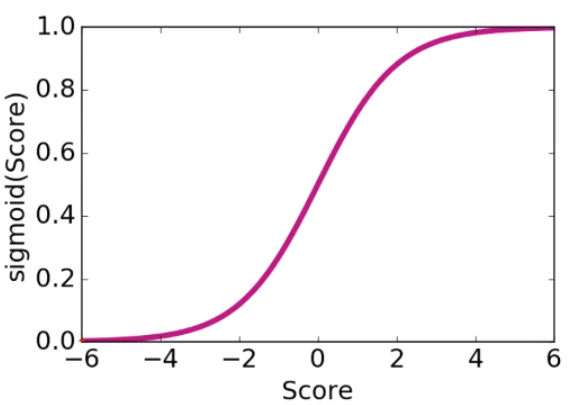
# Summary

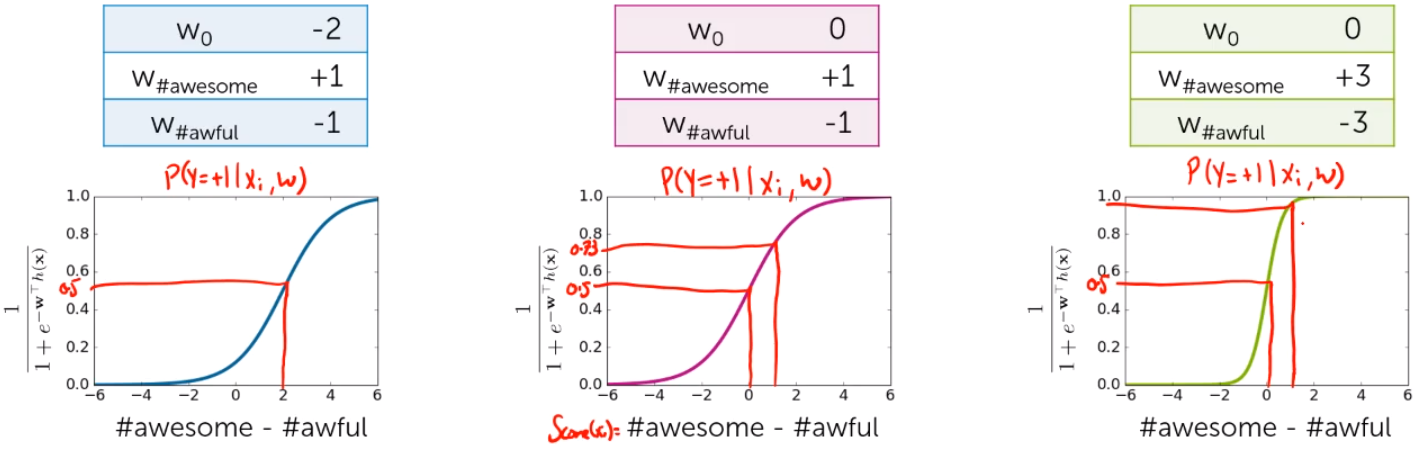
* Idea of classification is to predict a category or class based on some input
  + Classification is one of the most widely used techniques in machine learning, with a broad array of applications, including sentiment analysis, ad targeting, spam detection, risk assessment, medical diagnosis and image classification, web search
  + Can have multiclass classifier (many categories)
  + Picture classifier (using deep learning – dog or cat?)
  + Precision –
  + Recall –

## Linear classifier

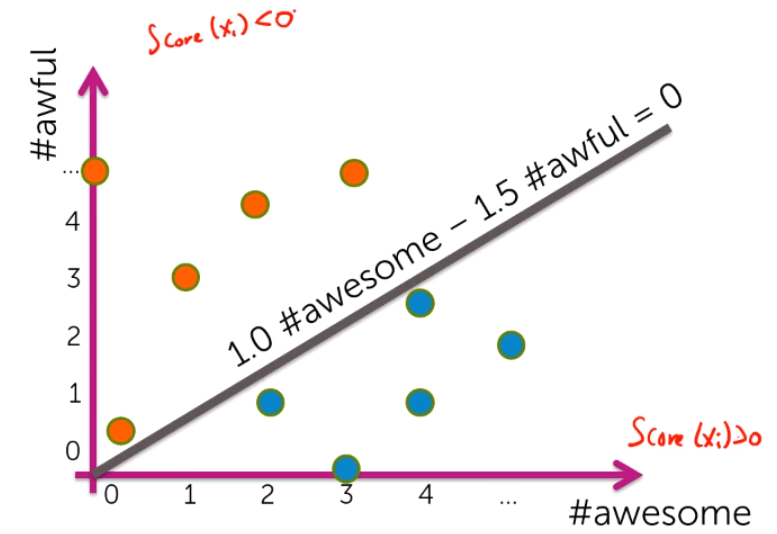
* + Logistic regression – prediction YES or NO (or 1 or 0)
  + For sentiment review (from text) - linear classifiear put weights (positive and negative) on words and calculates if the review is positive or negative. We are learning weights. If sum of number of words times their weights is positivie, then review is positive, otherwise –negative
  + We identify weights by maximizing probability that given input data (explonatory variables) would yield the given output. For that we maximaze loglikelihood function. We take log, because it makes calculations/derivations easier
  + To convert input to probability from 0 to 1 we use:



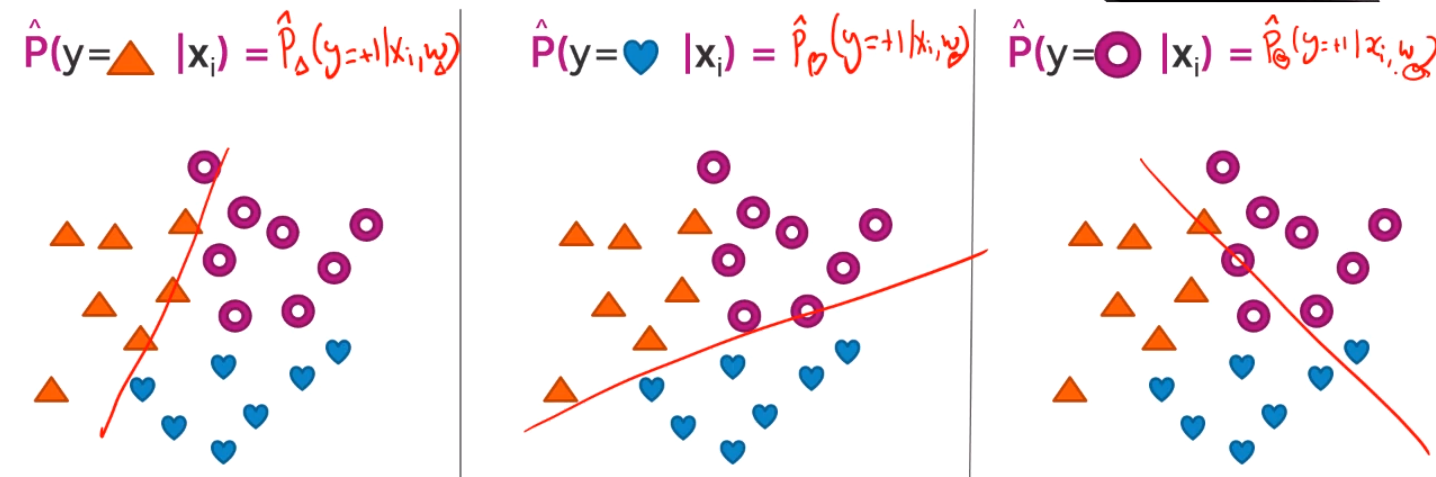




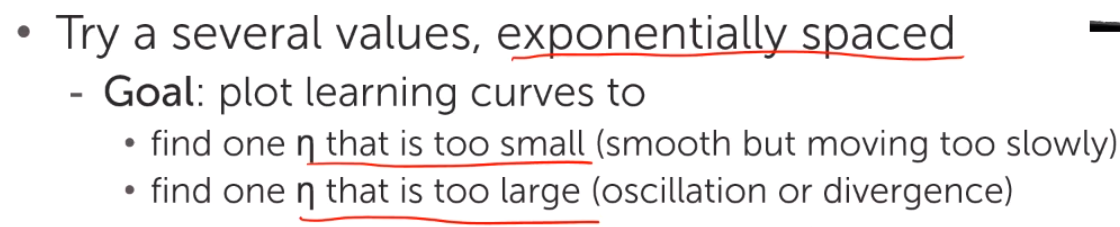
* + Linear decision boundary – line (plane) that divides all data points into the ones giving negative prediction, and others - positive

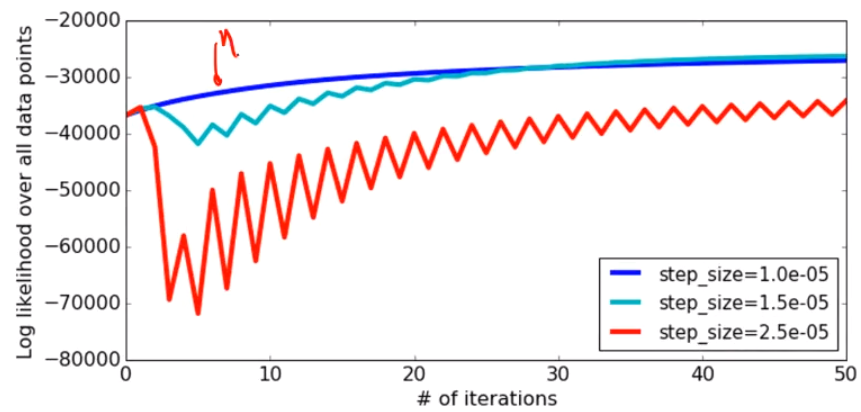


* + For multiclass estimation we can have 1-vs-all approach. For each individual class we have a binary model which predicts whether the given observation belongs 1) to this class, or 2) to any other class. Then we predict the class of this observation by choosing the class that got the highest probability.

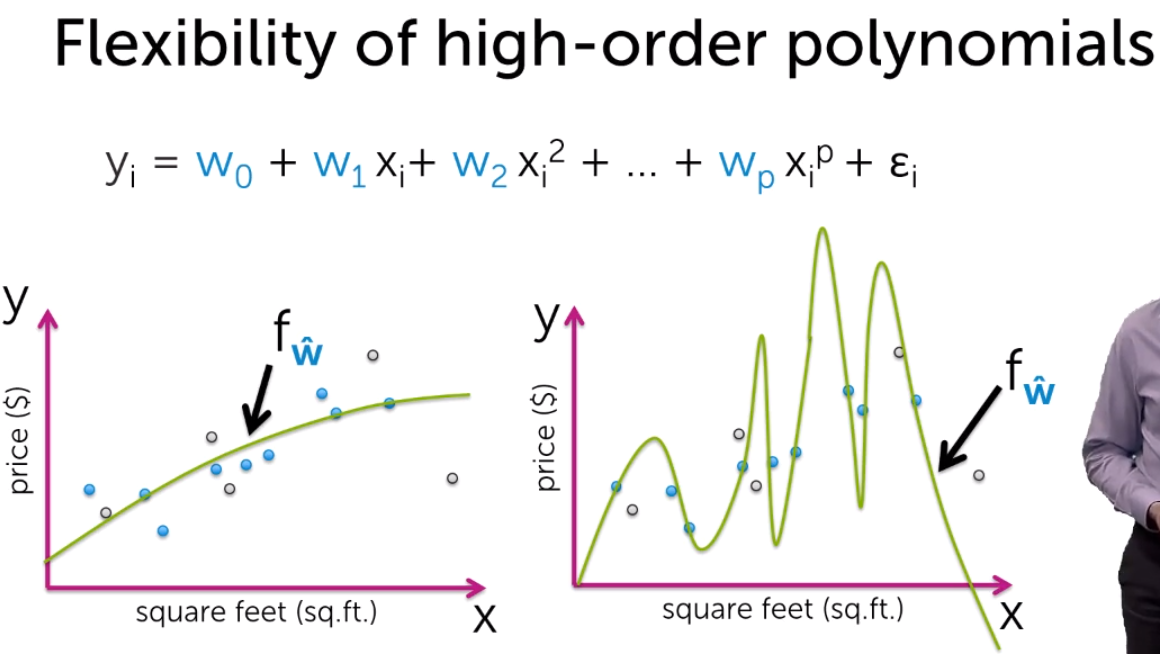


* + We use maximum likelihood estimation (MLE) to get weights that would result in the predictions that are as close as possible to the observed independent variables
  + In practice we use gradient ascent algorithm to identify optimal weights
  + Choosing step size in gradient ascent:

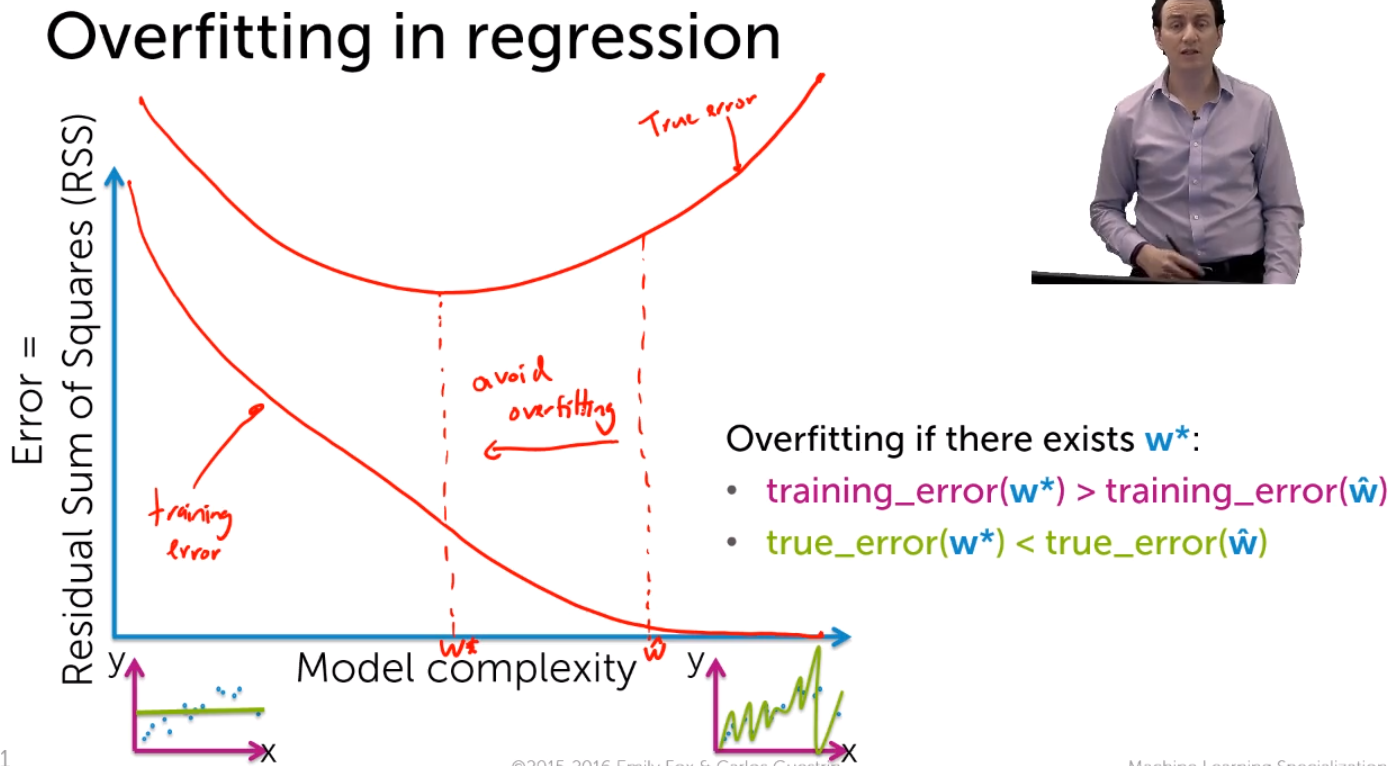




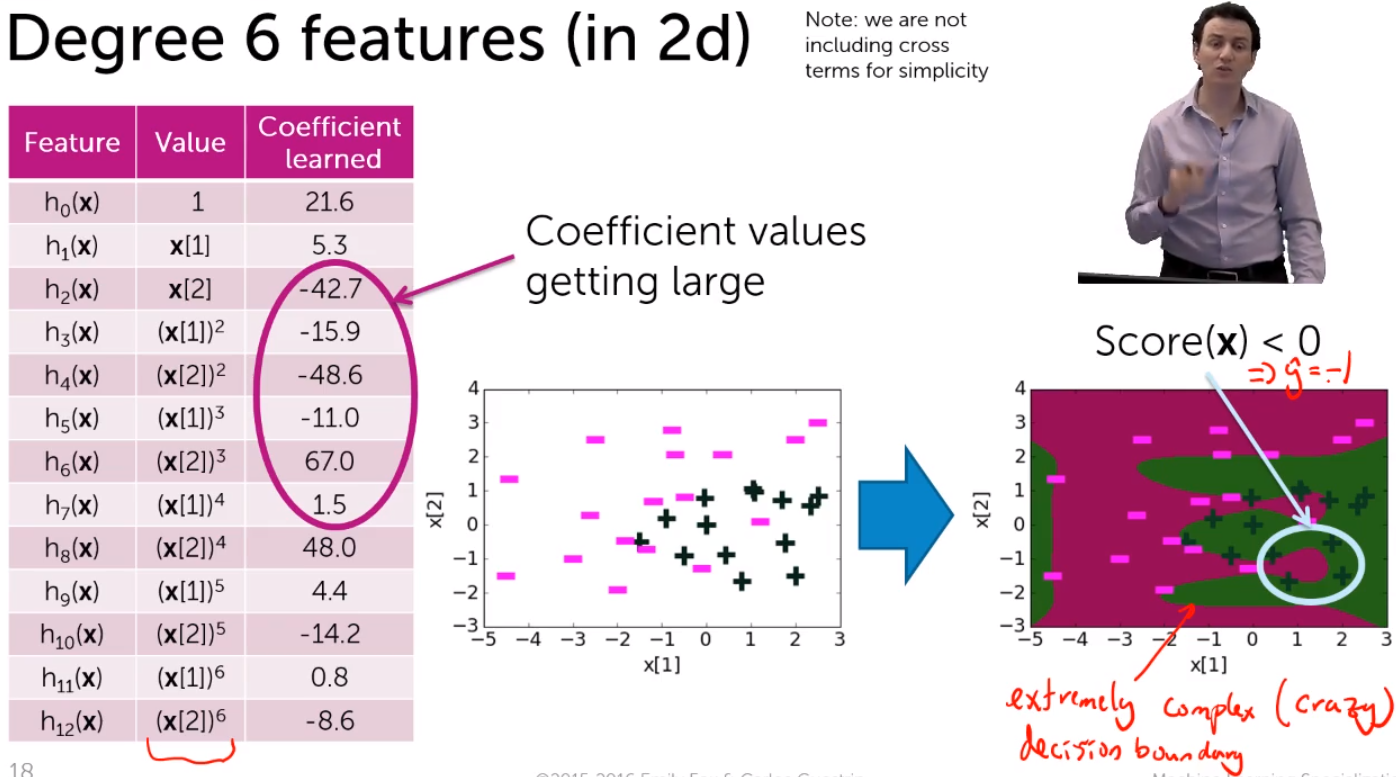
* Regularization and overfitting
  + Overfitting is a major problem in ML. Manually choosing the right variables is not an option, since there could be thousands of them. One way to address it is to use regularization.
  + Smooth vs overfitted estimation:



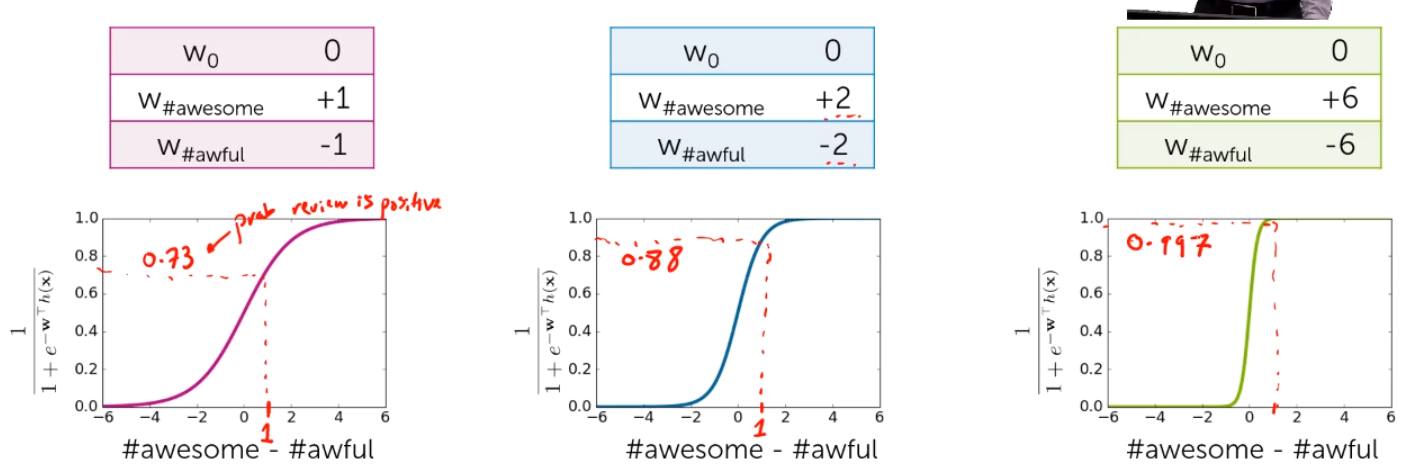
* + RSS vs model complexity:



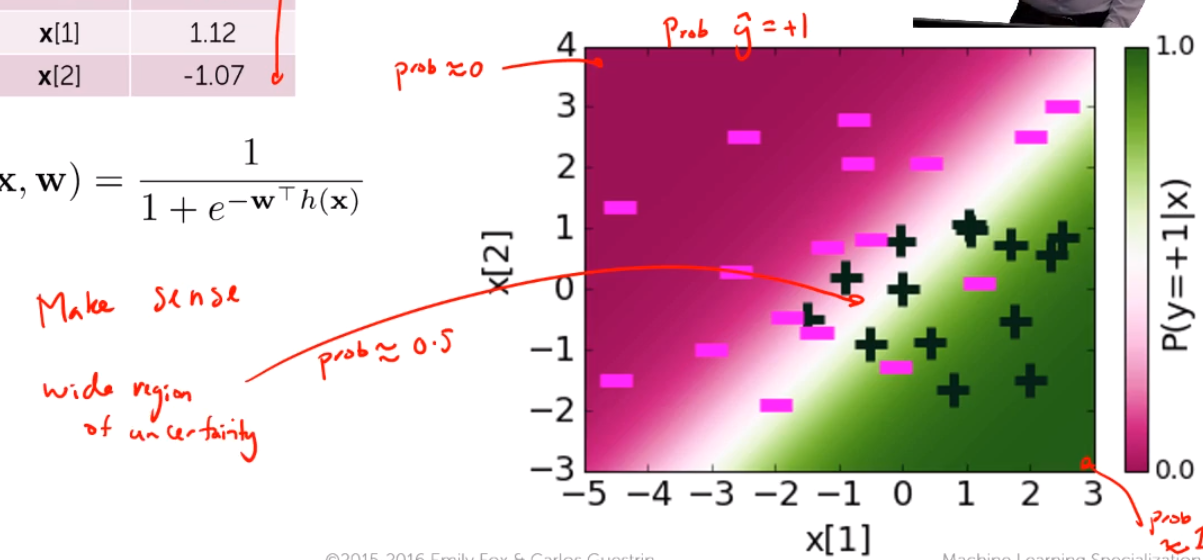
* + Overfitted higher order classifier with large coefficient values:

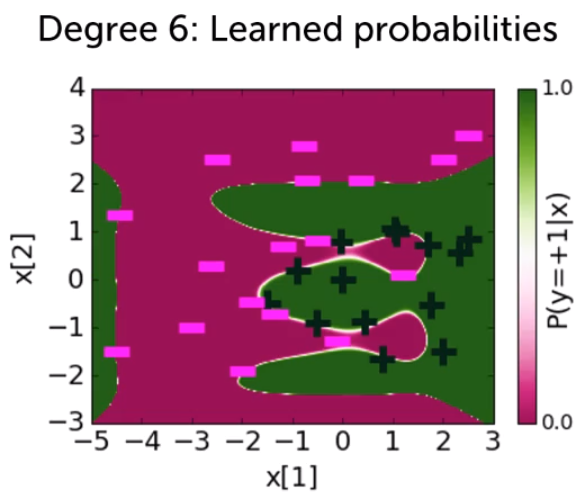


* + In overefitted classification problem we get very high coefficient. This makes our predictions very extreme and overconfident – e.g. we are either ~100% or 0% certain, nothing in between

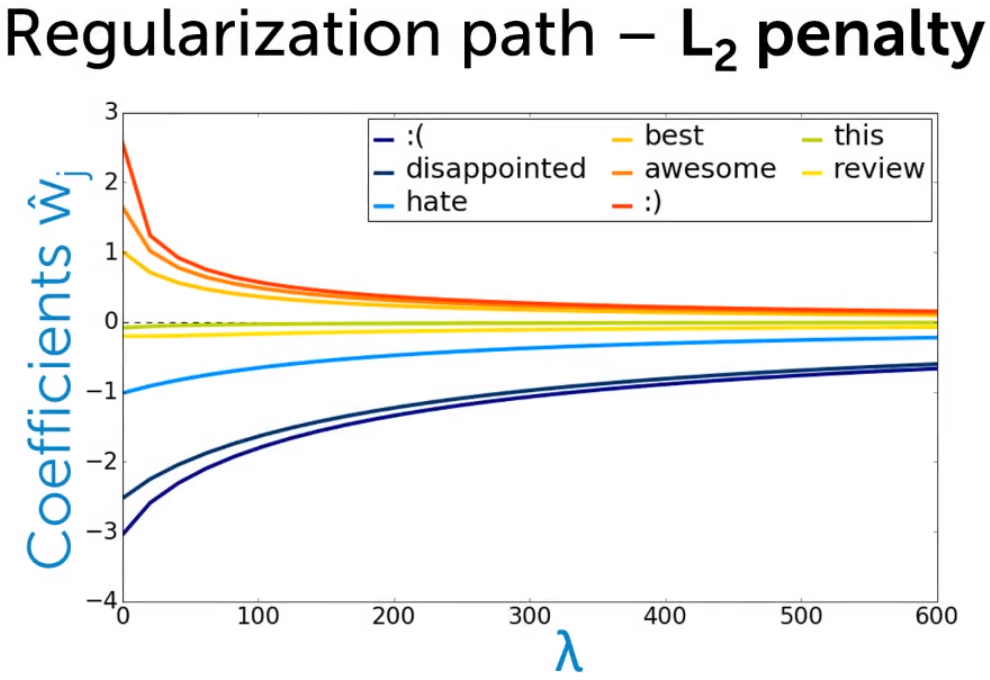


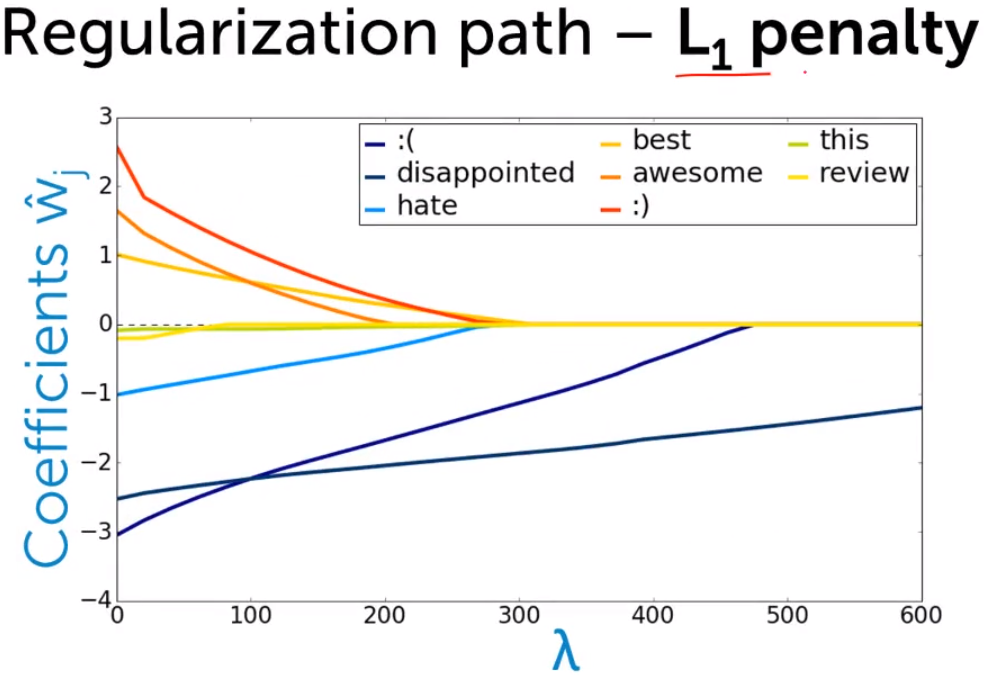
* + This overconfidence is seen by very narrow decision (uncertainty) boundary. Simpler model has a wider decision boundry:



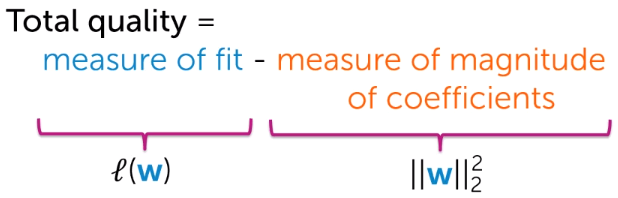
VS 

* + Regularization – there are 2 choices. L2 and L1. These are done by adding additional component to the estimation algorithm. This component penalizes too large weights on features.
    - L2 evenly pushes all coefficient closer (but not equal) to zero.
    - L2 is a sum squares = W1^2 + W2^2 … + Wn^2
    - L1 forces coefficients to become zero one by one. Thus, L1 reduces the number of variables (features) needed – it makes feature matrix more sparse. This sparsity can significantly reduce computational complexity
    - L1 = sum of absolute values = |W1| + |W2| + … + |Wn|
    - Intuition: in L2 you have a square, so in maximization/minimization it is more ‘efficient’ to reduce all coefficients a bit, rather than reduce only some to zeros. In L1 you have absolute value, so in maximization/minimization it is better to keep important coefficients (for log-likeglihood), and to make the unimportant ones to zeros.

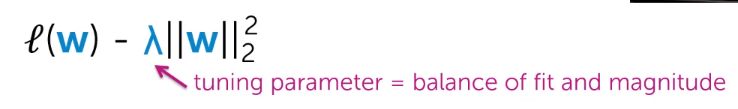




* + Thus we balance between measure of fit and measure of coefficient magnitude.



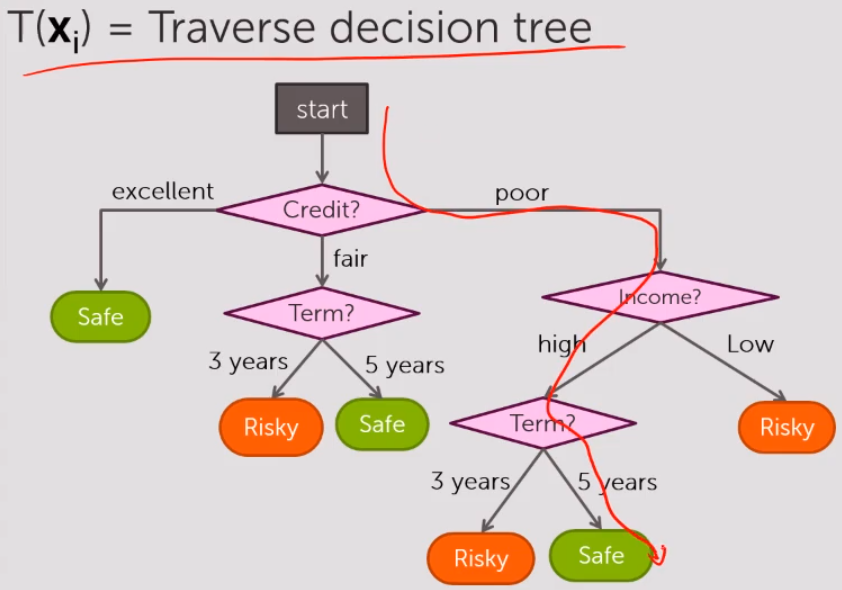
* + Maximize (tune lambda on validation set or using cross-validation)



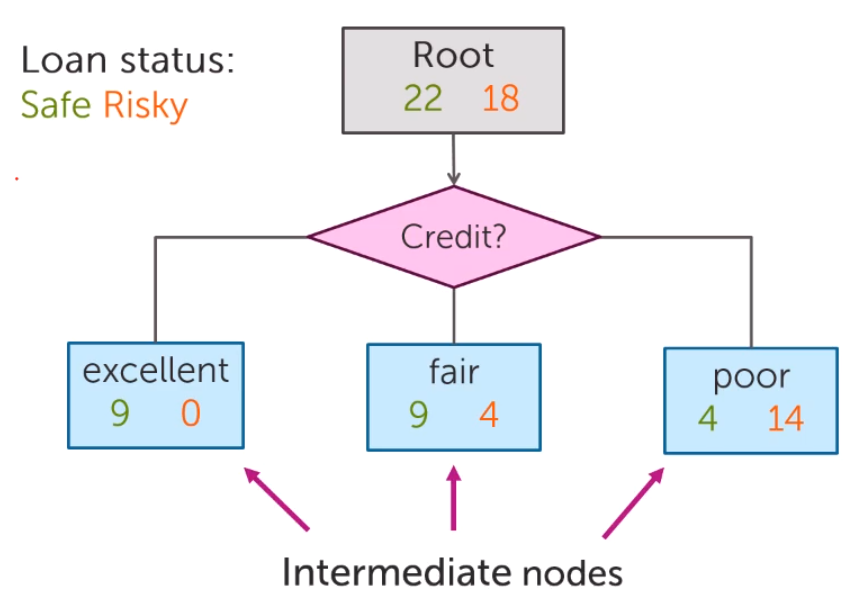
* + - As we increase lambda, coefficients become closer to zero and we make our uncertainty boundary wider
  + We train the model on the training set and test it on the validation set. To measure the model’s performance we use accuracy metric = correctly predicted observation / total observations, or error metrics = 1 – accuracy metric = incorrectly predicted observations / total observations
* Train-test-validation split
  + On training set you ‘train’ the model. On validation set you calibrate parameters like lambda. On the test set, you asses the overall performance of the model

## Decision trees

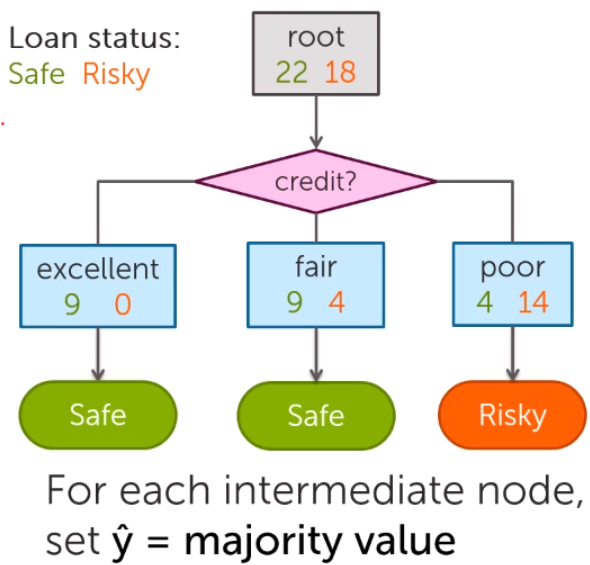
* Along with linear classifiers, decision trees are one of the most widely used classification techniques
* Very useful with boosting
* Predict outcome by going through steps



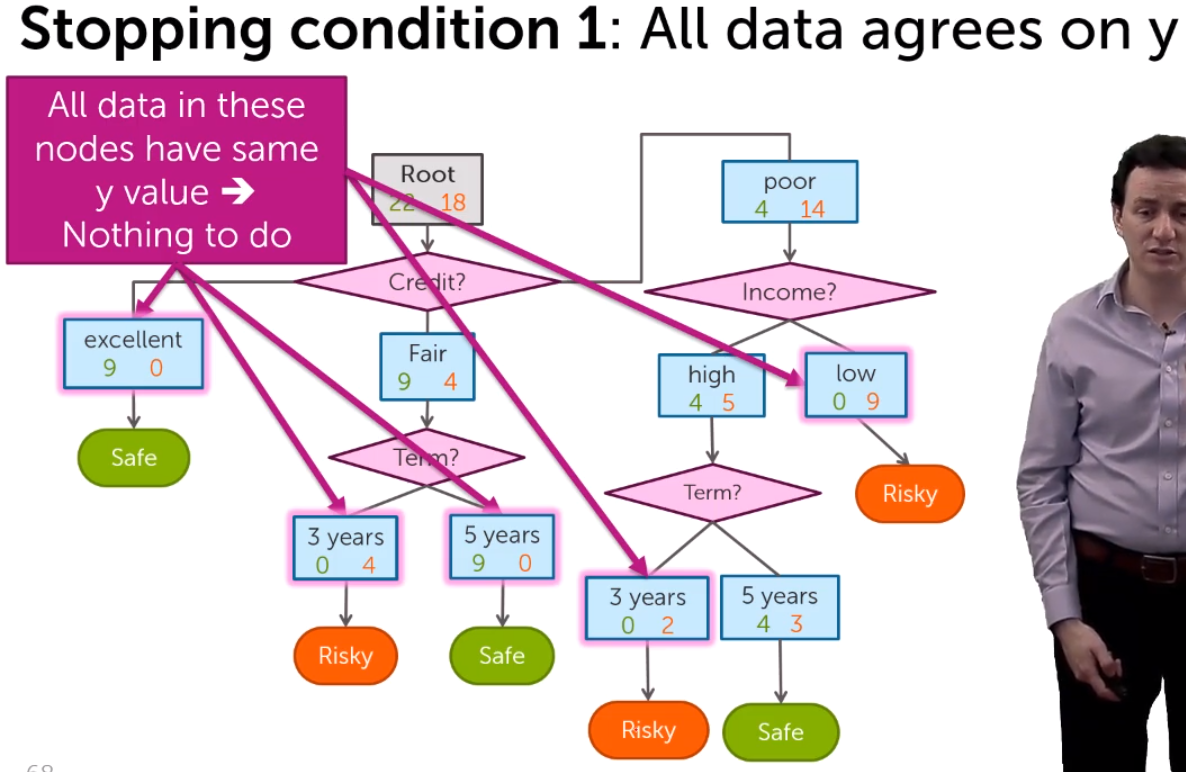
* The quality metric for decision trees is simply fraction of mistakes. Error = # incorrect predictions / # examples. In the algorithm we are minimizing this error
* Going through all the possible routes is computationally impossibly, that is why we use simple greedy algorithm (recursive)

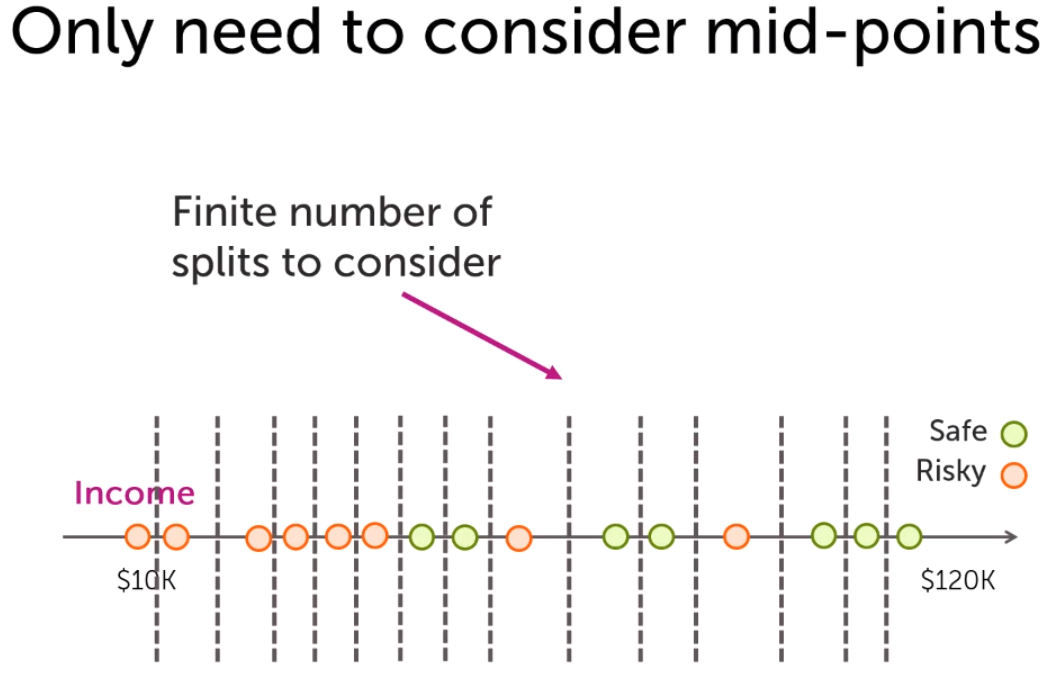


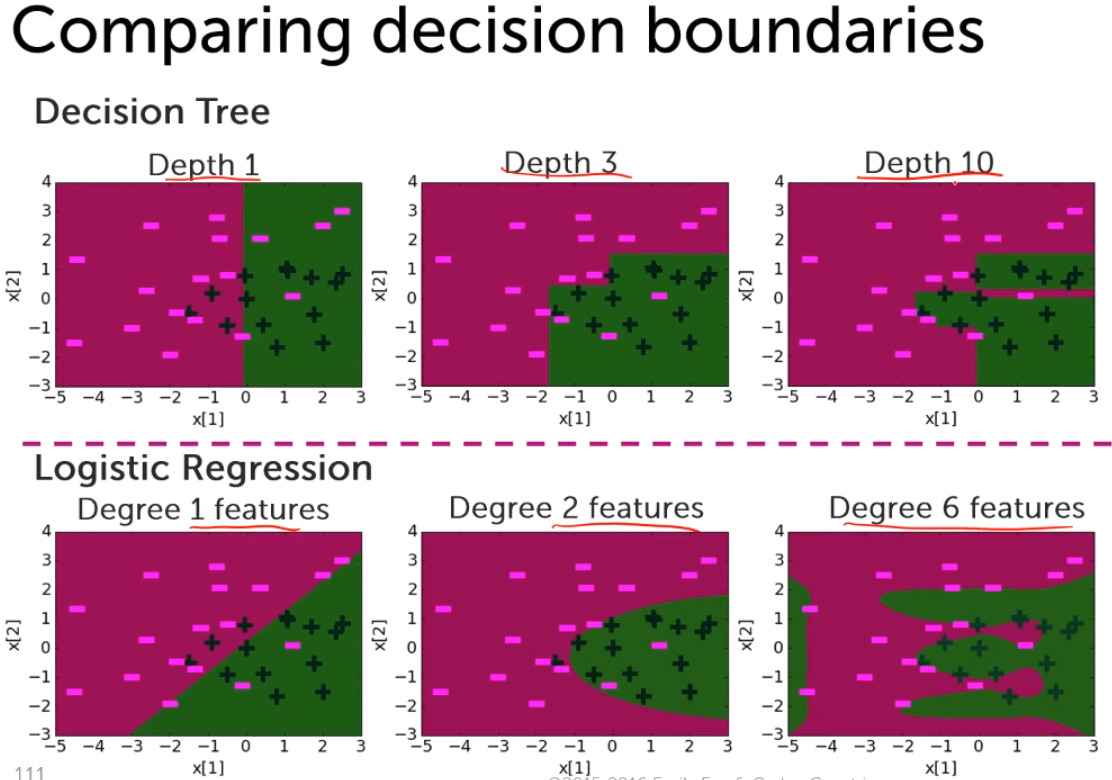
* Approach 1 – decision stump. At the given branch simply predict the outcome by looking at the majority value. E.g. if there 14 risky loans and 4 good loans at the branch poor credit, then we predict the loan to be risky.

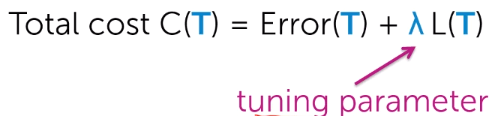


* Simple recursive algorithm at each split chooses the feature to split on that is minimizing the error at this point. At the next step it again chooses the next feature in a similar fashion. That is why it is called recursive.
  + There are several conditions when to stop.
    - Condition 1 – all data at a given node has the same value – just use this value for prediction then.
    - Condition 2 – we have run out of all the features – thus we take the majority value for prediction here.
    - Condition 3 – stop splitting if the split does not decrease our error. This one is dangerous, since after split our error can remain the same, however, in the following splits it could actually go down.



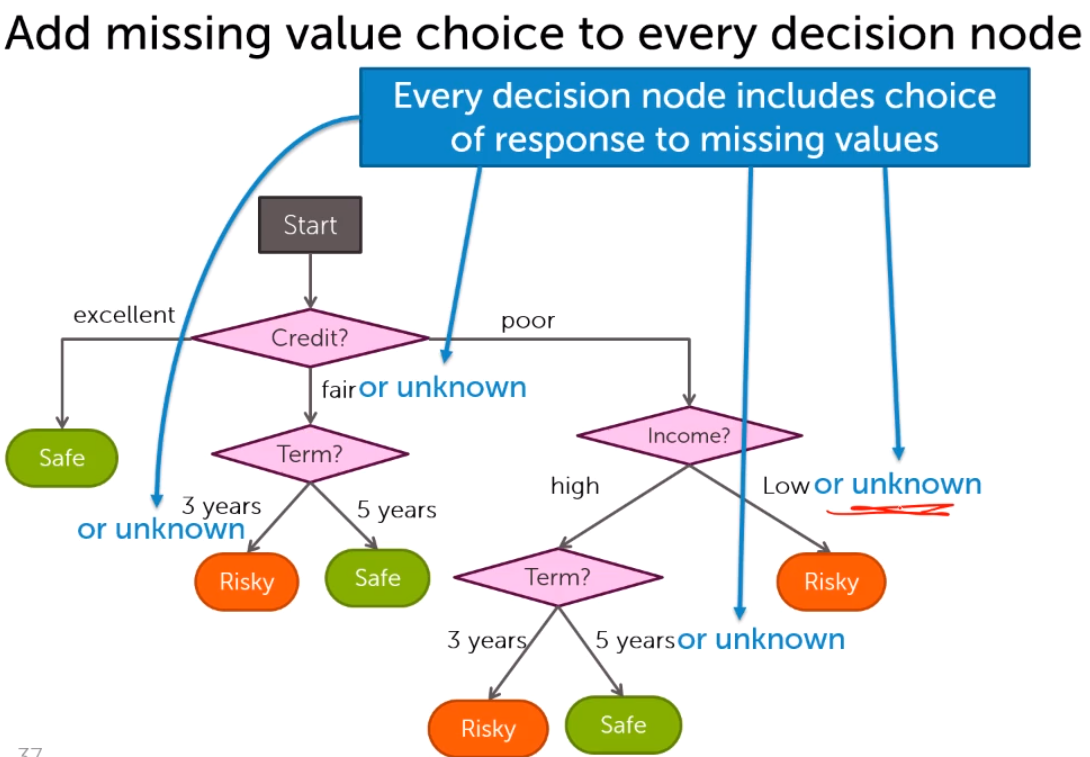
* Splitting continuous features
  + We split continuous features in categories. Income can be split in below 1000 EUR and above.
  + When deciding how to split – we can consider only the mid-point between each two neighbouring data points
  + 
  + Unlike categorical variables, continuous features can be split numerous times
* Decision boundries:



* Prunning –
  + There are two approaches in picking simpler trees. 1st approach is using early stopping counditions (all samples have the same class, classification error is not decreasing, no more features to split on, max depth reached, min number in the node or leaf is reached). 2nd approach – prunning. It means you simplify the tree after the algorithm is finished (get a complex tree and make a simple tree from that complex).
  + Condition 3 – stop splitting if the split does not decrease our error. This one is dangerous, since after split our error can remain the same, however, in the following splits it could actually go down. By using prunning you mitigate this risk.
  + Similary to regression, you can introduce a penalty term for decision trees. One could be – the number of leaves. Thus, you will balance between precision and complexity, that helps to deal with overfitting automatically. 
  + Prunning algorithm: 1) build a complex decision tree; 2) go from the bottom of the tree and see if total cost is reduced if the split on a given node is not implemented; 3) if total cost is smaller – prune (remove the split from the node and put a leaf there).

Boosting

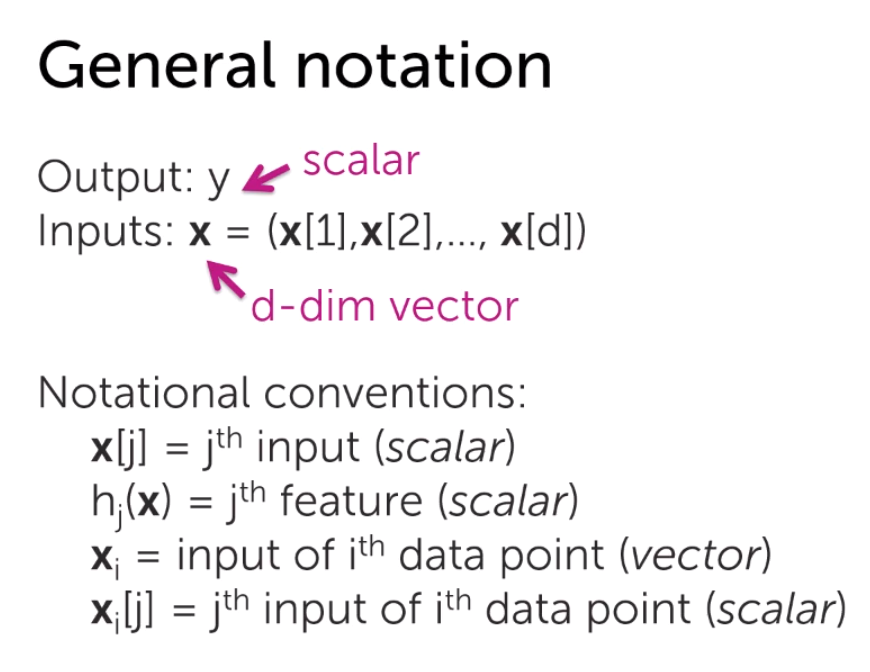
## Handling missing data

* Missing data affect both training and predictions
* Can modify the decision tree model to deal with missing data.
  + Strategy 1 – Purification – remove/skip missing data. Can lose valuable information though. OK if few observations with missing values though
    - Skip features with many missing values. OK if there are a lot of missing values for that feature
    - Is ok for training data, however, might not work when making predictions
  + Strategy 2 - Imputation – make a best guess to fill missing values
    - Just put the most common value for categorical value, or for numerical data put median or average
    - More advanced features – expectation-maximization algorithm
    - Can apply both on training data and on prediction
    - Imputation introduces bias (can lead to systemic bias)
  + Strategy 3 – modify learning algorithm to handle missing data (perhaps the best solution)
    - 
    - For each node we decide saporately where missing data should go
    - Modification to the algorithm 1 – when selecting the best feature to split on, we need to decide in which branch (left or right) the missing values should go. We put the missing values to the branch that would give the lowest classfication error for this split

## Boosting

* Boosting allows to complement various algorithms to do better prediction. Predictions are better when various models are used at the same time. Most Kaggle competition are won by teams who use boosting. Boosting had a tremendious impact on ML

# Some notes



* Overfitting in decision trees
  + Decision trees are particularly prone to overfitting. We can simply get as many leafs as many observation we have, to perfectly ‘predict’ them. By definition, we split nodes/leaves based on the lowest training error – thus we go directly to overfitting
  + Occam’s razor – from the all exmplanations, choose the simplest/easiest/ the one that require fewest assumptions



* + How to pick simplier trees?
    - Early stopping – stop before tree becomes too comlex (e.g. limit depth)
      * Determined by cross-validation
      * Too inflexible, since you might want to cut some branches earlier, but others - laters
    - Pruning – simplify tree after learning algorithm terminates
      * Add parameter ‘e’, and stop if error doesn’t decrease by more than ‘e’
    - Stop when there are only few observations left, otherwise will get overfit
      * E.g. set Nmin = 10 for small dataset or 100 for bigger datasets