HALMSTAD UNIVERSITY

Learning Systems (DT8008)

• Overfitting and Generalization
• Ensemble Methods

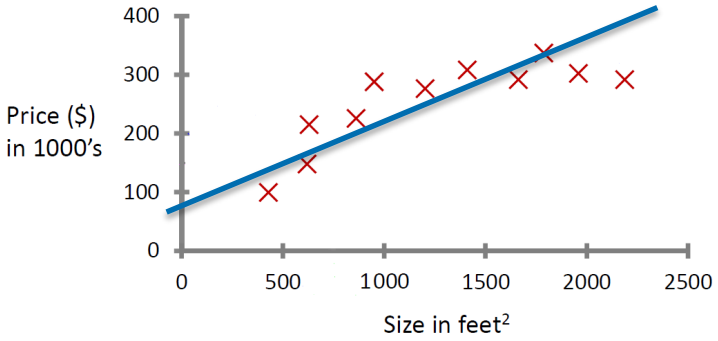
Dr. Mohamed-Rafik Bouguelia
mohamed-rafik.bouguelia@hh.se

Halmstad University

The Problem of Overfitting

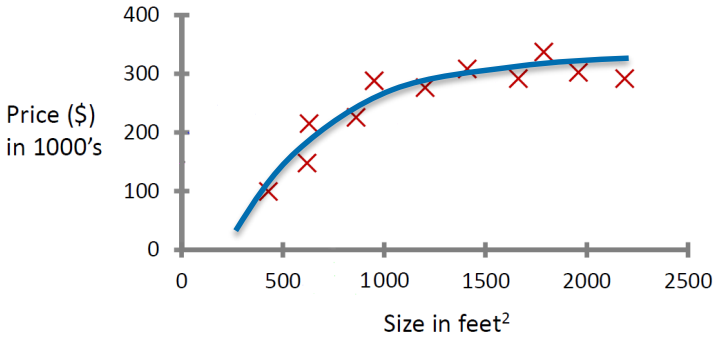
The problem of overfitting - regression

Example: Linear regression (housing prices)

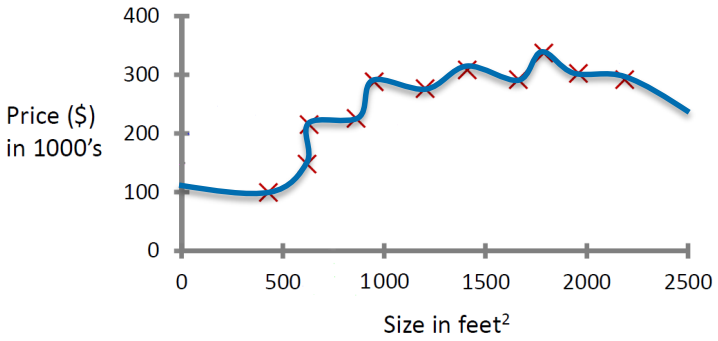


$$h_\theta(x) = \theta_0 + \theta_1 x$$

Which of these models do you think is a good model **for this data**?



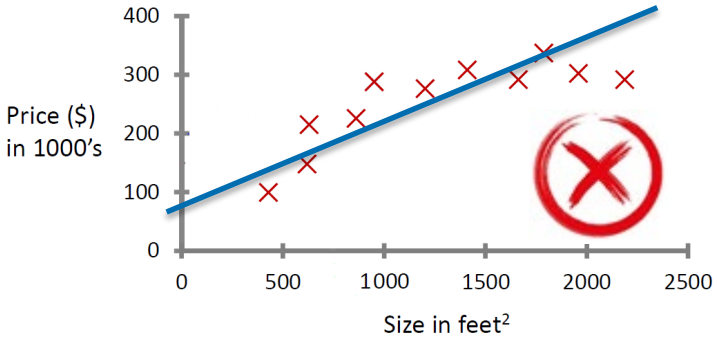
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6 + \theta_7 x^7 + \theta_8 x^8 + \theta_9 x^9 + \theta_{10} x^{10}$$

The problem of overfitting - regression

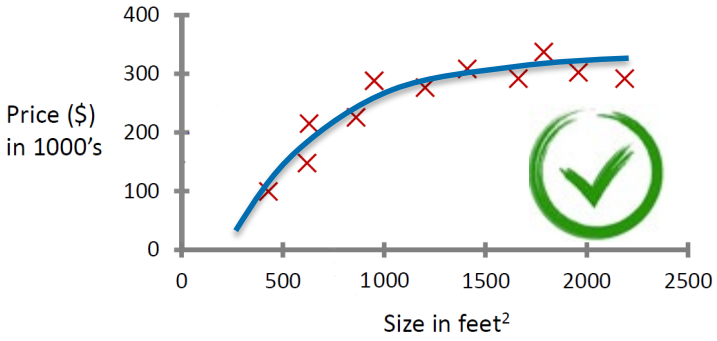
- **Underfitting**
- The model has a high **bias**
- The model makes a strong assumption that the housing prices will vary linearly with their size, but ends up not fitting the training data well (poor fit).



Simple model

$$h_\theta(x) = \theta_0 + \theta_1 x$$

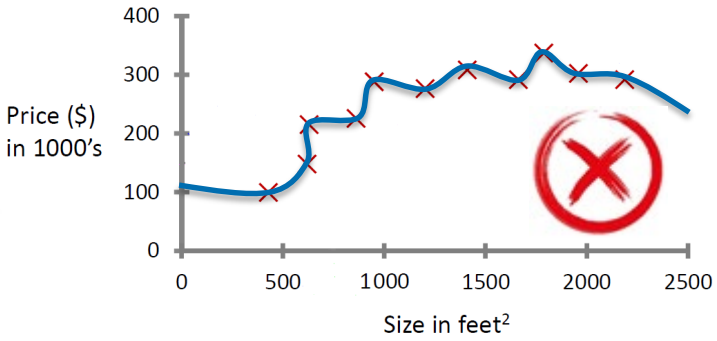
- Just right
- It fits the data pretty well



More complex model

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

- **Overfitting**
- The model has a high **variance**
  - The space of possible hypothesis functions of this order is too large (too variable), and we do not have enough data to construct a good hypothesis of this type.
- Seems to fit the training data perfectly, but will have very poor performance on new data. It has 0 error on the training data, but it does not generalize well.

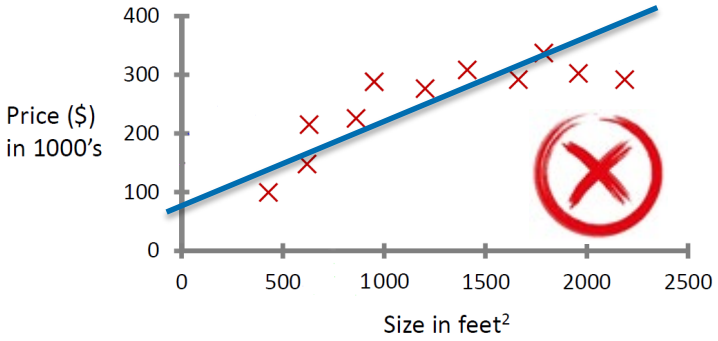


Much more complex model

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6 + \theta_7 x^7 + \theta_8 x^8 + \theta_9 x^9 + \theta_{10} x^{10}$$

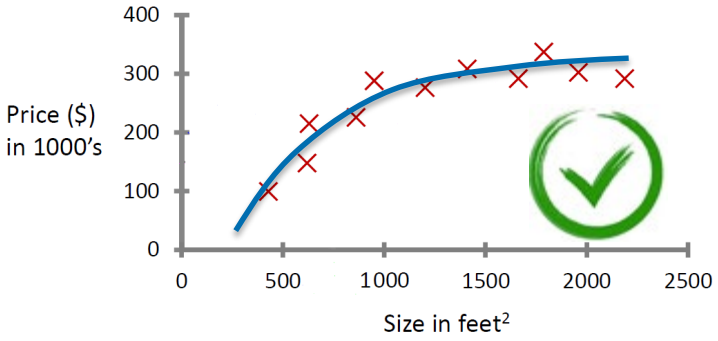
The problem of overfitting - regression

Overfitting: If we have too many features, the learned model may fit the training set very well ($E(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left[h_\theta\left(x^{(i)}\right) - y^{(i)}\right]^2 \approx 0$), but fails to generalize to new examples (predict prices on new examples).



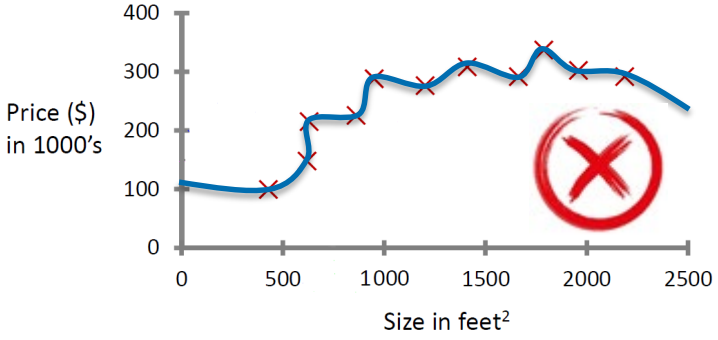
Simple model

$$h_\theta(x) = \theta_0 + \theta_1 x$$



More complex model

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$



Much more complex model

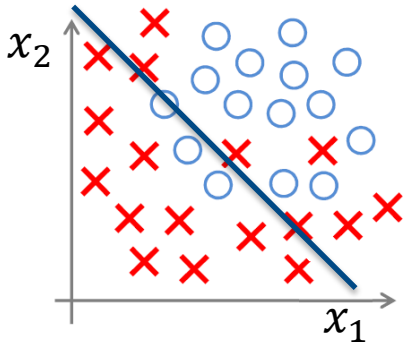
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6 + \theta_7 x^7 + \theta_8 x^8 + \theta_9 x^9 + \theta_{10} x^{10}$$

The problem of overfitting - classification

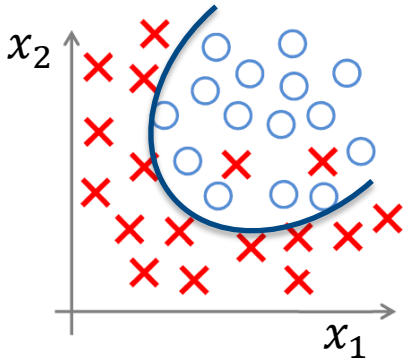
Example: Classification
(with Logistic Regression)

NOTE: g here is the sigmoid function.

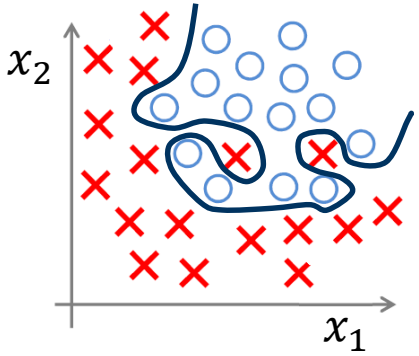
Which of these models do you think is a good model **for this data**?



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5\, x_1 x_2)$$



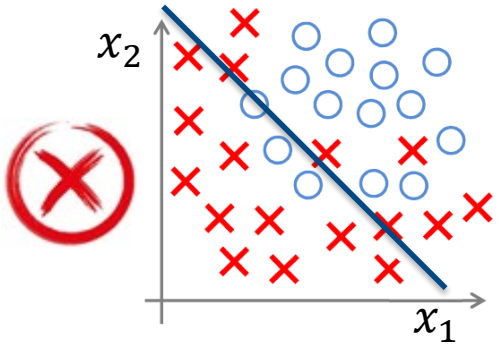
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4\, x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \cdots)$$

The problem of overfitting - classification

Example: Classification
(with Logistic Regression)

NOTE: g here is the sigmoid function.

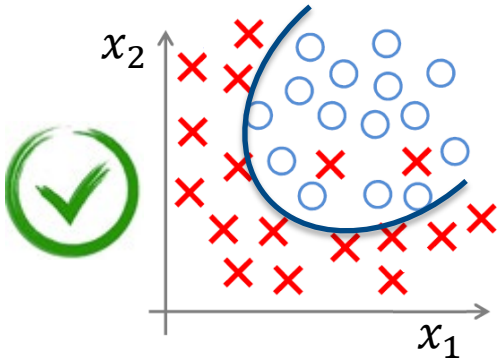
Underfitting
(high bias)



Simple model

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

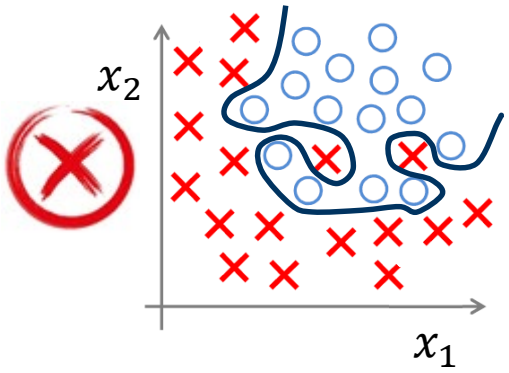
Ok



More complex model

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5\, x_1 x_2)$$

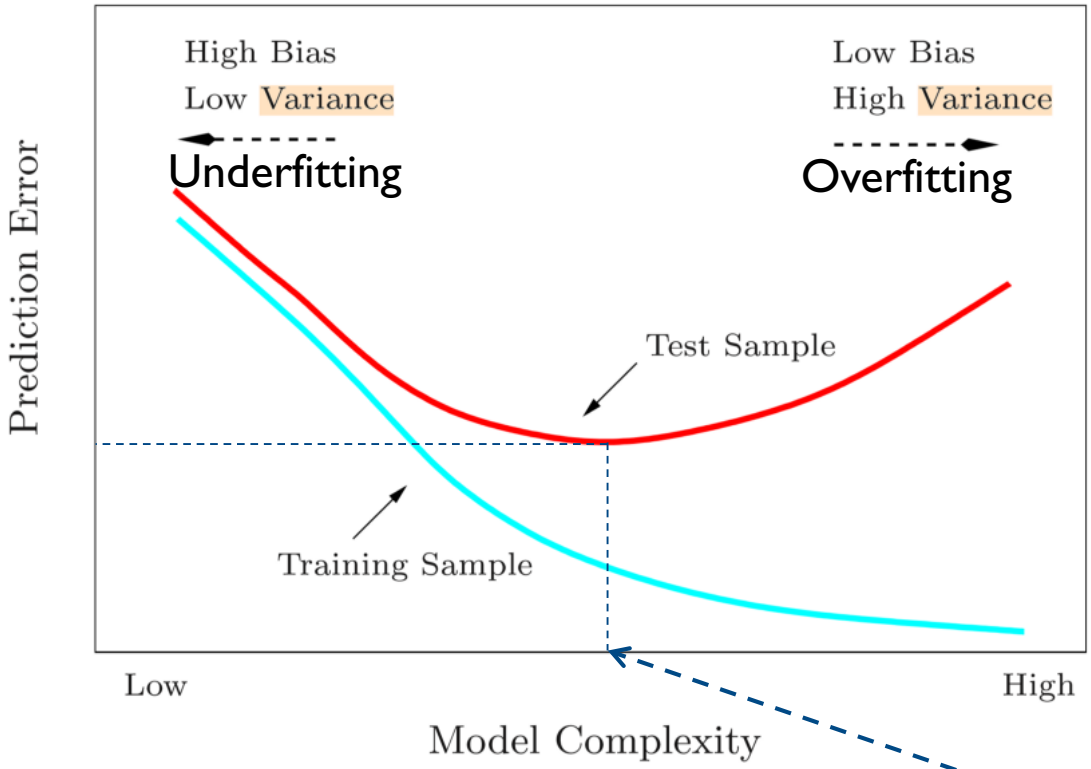
Overfitting
(high variance)



Much more complex model

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4\, x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \cdots)$$

The problem of overfitting



- What makes it more likely to overfit?
  - ❖ Not enough training examples (small training dataset)
  - ❖ Too many features
  - ❖ Using a non-convenient type of models / hypothesis functions (e.g. too much complex for our problem / data).

A good model complexity

Generalization Error

Generalization Error

- The training error is the error of the model h_θ on the training examples.
  - It is **not** an estimation for the error that the model h_θ will have when deployed and applied on new data.

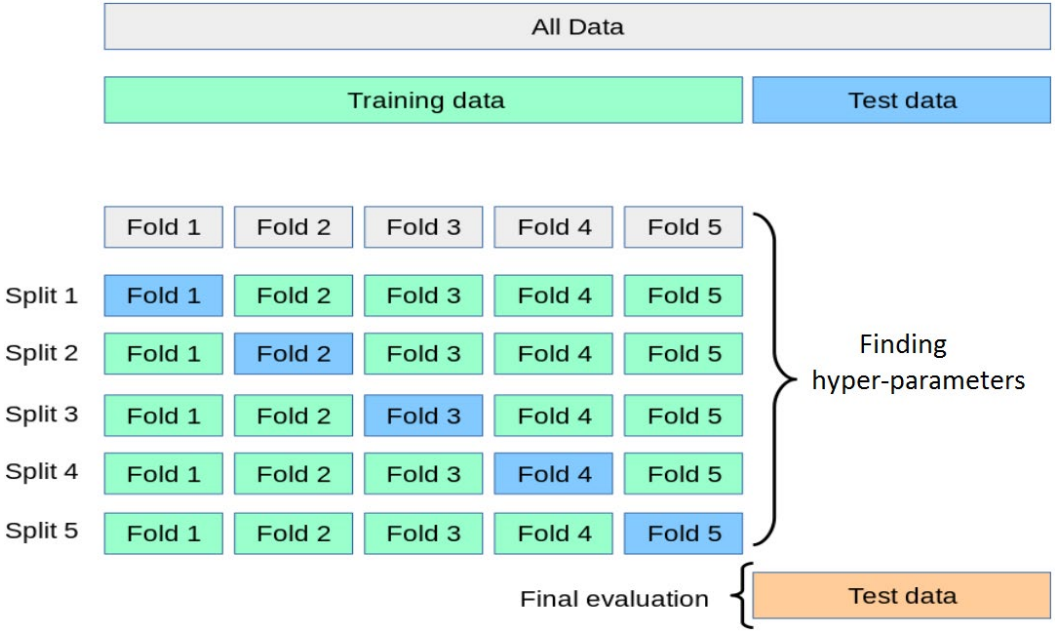
- The **generalization error** is the error of the model h_θ on new (unseen) data.
  - The generalization error is typically higher than the training error.

- How do we estimate the generalization error of some model?
  - Using k-fold Cross Validation (k-CV)
  - Using Leave-One-Out estimate (LOO)
    - Same as k-CV with k = the number of examples.
    - LOO is better when the number of data-points in our dataset is small.

Generalization Error

Example of **k=5-fold cross validation**:

- Split the training dataset into 5 parts (folds).
- Each time, train a model on the training parts (green) and apply it on the remaining part (blue) to estimate the error.
- So, you will finally get 5 estimates of the error. The **generalization error** is the average of these 5 errors. In the case of classification, you can also compute the generalization accuracy the same way (accuracy% = 100 – error%)



Leave-one-out cross validation:

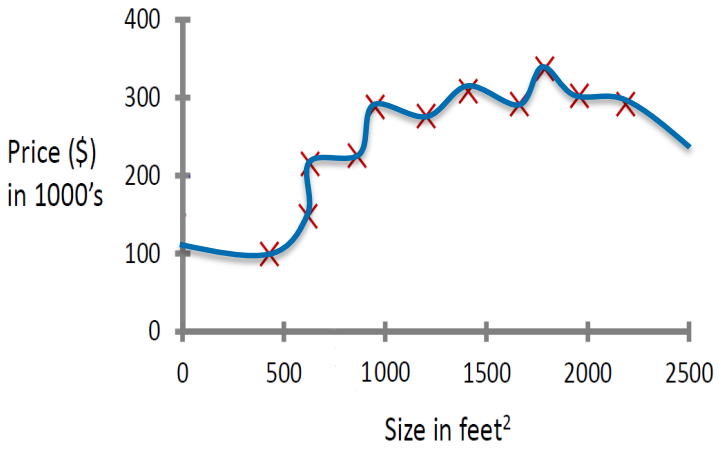
- It is similar to the k-fold cross validation, but the training dataset is split into n parts, where n is the number of training samples. i.e., we leave one sample each time.
- It is useful when you have a small training dataset (i.e. when n is small).

Addressing overfitting

Addressing overfitting

- It becomes hard to visualize the data and the model when we have more than 2 or 3 features.
  - So how can we avoid overfitting?

$x_1 =$ size of house
$x_2 =$ no. of bedrooms
$x_3 =$ no. of floors
$x_4 =$ age of house
$x_5 =$ average income in neighborhood
$x_6 =$ kitchen size
$\vdots$
x_{100}



Addressing overfitting

1. Model selection / hyperparameters tuning **(this lecture)**
   - You can try various models of different complexity (e.g. with various hyperparameters values), compute the generalization error for each of them (as explained previously), and keep the best hyperparameters.

2. Reducing the number of features **(this lecture)**
   - We are more likely to overfit when the number of features is high (relatively to the size of the dataset).
     - Manually select which features to keep / remove
     - Or using feature selection algorithms

3. Using an ensemble methods **(this lecture)**

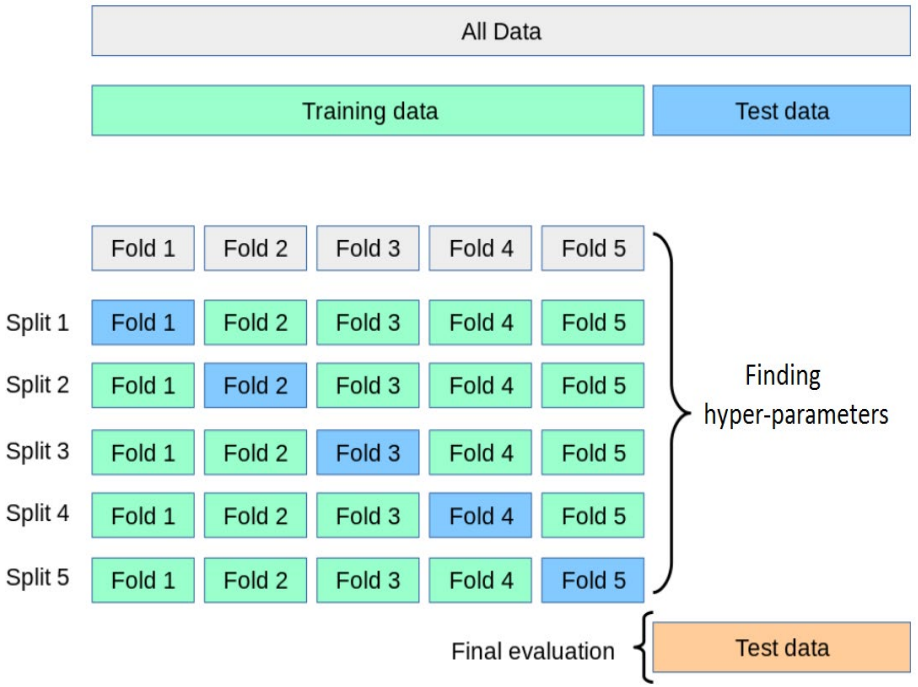
4. Using regularization **(next lecture)**
   - Keep all features, but reduce the magnitude / values of parameters θ_j
   - Works well when we have a lot of features, and each feature contributes a bit to predicting y

1. Hyper-parameters Tuning

Hyperparameters Tuning

- Most ML algorithm has hyperparameters that lead to a more or less complex model.
  - e.g. the K in the KNN algorithm, is a hyperparameter.

- To find the values of hyperparameters that lead to a good model (not too simple, not too complex), we need to:
  1. Pick some values for our hyperparameters.
  2. Compute the generalization error using a 10-fold-cross-validation.
  3. Repeat steps 1 and 2 (with various values for the hyperparameters).
  4. Keep the hyperparameters that gave you to the smallest generalization error; and train a model on the whole training set using those hyperparameters.
  5. You can now test your trained model on the test data (see figure) to see how it will perform when deployed in real-world.



Comparing two models

- Suppose that you want to compare two regression algorithms A and B.
- Suppose that, when you performed a 10-fold-cross-validation on A and B, you got the following MSE error estimates:
  - For A: $[0.6, 0.1, 1.1, 1.3, 0.3, 3.2, 3.2, 0.9, 1.3, 1.1]$. The average is $\mu_A = 1.31$. The stdev is $\sigma_A = 1.01$
  - For B: $[3.3, 0.6, 5, 12.4, 1.2, 5.8, 8.6, 0.4, 0.5, 4.35]$. The average is $\mu_B = 4.21$. The stdev is $\sigma_B = 3.75$

- Which model is the best?
  - Model A is better than model B as it achieves a smaller generalization error than B ($\mu_A < \mu_B$).

- Is A significantly better than B?
  - To answer this question we need to do some statistical tests. One such test (Wilcoxon test) uses the standard deviations σ_A, σ_B and compares the means μ_A, μ_B to check if:

$$|\mu_A - \mu_B| > 1.96 \frac{\sigma_{AB}}{\sqrt{K}}$$

where $\sigma_{AB} = \sqrt{\sigma_A^2 + \sigma_B^2}$

and K=10 folds in this example.

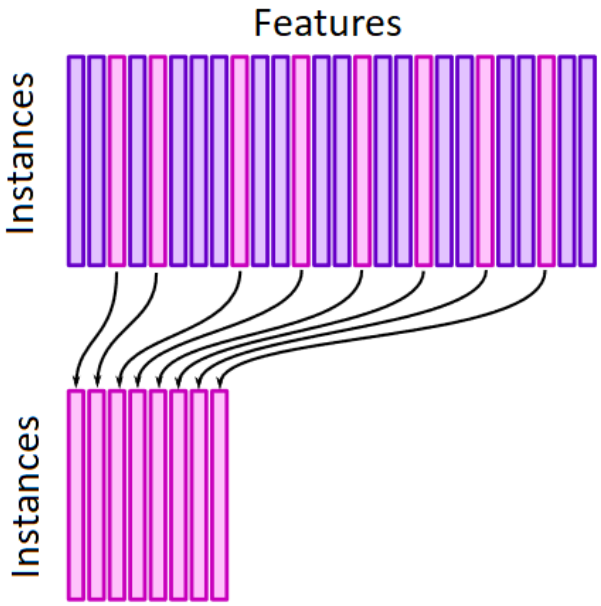
➔ If true, then the results achieved by A and B are significantly different (i.e. A significantly better), otherwise not (i.e. A not significantly better).

2. Features Selection

Motivation behind features selection

- Why? To reduce overfitting which arises
  - when we have a regular data set (with enough data-points), but too flexible model
  - or when we have a high-dimensional dataset with not enough data.

- Datasets with thousands or millions of features are quite usual these days. We want to choose only those, which are needed to construct simpler, faster, and more accurate models.



Simple (naïve) feature selection

- Removing features with low variance:
  - One very simple (and unsupervised) way to do feature selection is to remove features with low variance.

- Motivating example:
  - Suppose that we have a feature that have the same value in all samples (i.e. a column of X which has a constant value)
  - Such feature would be useless because it doesn't help differencing between samples.
  - It will have a variance of 0, and will therefore be removed.

Recursive feature elimination

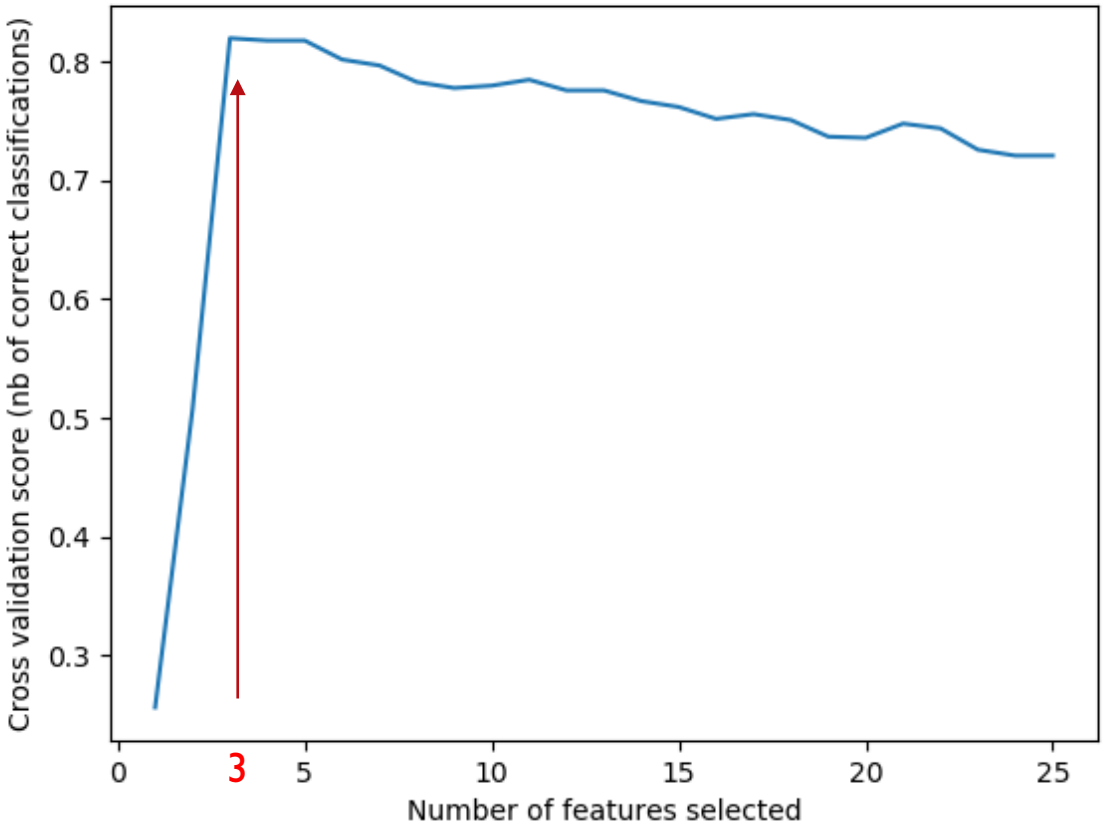
- Suppose that we have a model that assigns weights to features (e.g., the coefficients θ of a linear model)
  - $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_d x_d$ (it's a weighted sum of the features).
  - Features x_j that have small coefficients θ_j are considered less important.

- In the *recursive feature elimination* we select features by recursively considering smaller and smaller sets of features, as follows:
  - We train a model using the initial set of features.
  - We obtain the importance of each feature (e.g. through the coefficients θ if it's a linear model).
  - We remove the least important features from current set of features.
  - We repeat this procedure recursively until the desired number of features to select is reached.

- The selection of the best number of features to keep (i.e. the stopping criteria) can be done using cross-validation.

Recursive feature elimination

- A recursive feature elimination example with automatic tuning of the number of features selected with cross-validation.



In this example, the optimal number of features was 3.

HALMSTAD UNIVERSITY

Tree-based feature selection

- Some classification or regression models such as tree-based models (decision trees, random forest …) can be used to compute features importance.
- This is can be used to discard irrelevant features.
- Example with the scikit-learn (sklearn) library in Python:

```
>>> from sklearn.ensemble import DecisionTreeClassifier
    ...
>>> clf =DecisionTreeClassifier(n_estimators=50)
>>> clf = clf.fit(X, y)
>>> clf.feature_importances_
array([ 0.04...,  0.05...,  0.4...,  0.4...])
```

Ensemble Methods

Ensemble methods

Wisdom of the crowd

- Instead of using a single model, why not using several models at once.
  - An ensemble (also called committee) is a set of models (classifiers or regressors) whose individual decisions are combined in some way to predict the output of new examples.

- Diversity vs accuracy:
  - The individual classifiers composing an ensemble must be
    - **better than random**
    - and **diverse** (i.e. they make different errors on new data points).
  - An ensemble of classifiers must be more accurate than any of its individual members.

Ensemble methods – Motivating example

- We have 3 individual taggers who tag each word as
  - **V** (Verb), **N** (Noun), **DT** (Determiner), or **PN** (Pronoun)
- The taggers are diverse (i.e. they make different errors) but each of them is better than random (i.e. weak predictors).

	Bob	gave	Alice	the	key	ACC.
Tagger 1	V	V	N	DT	N	0.8
Tagger 2	N	N	V	DT	N	0.6
Tagger 3	N	V	N	PN	N	0.8
Majority	**N**	**V**	**N**	**DT**	**N**	**1.0**

- Average accuracy $\simeq 0.73$. Majority accuracy $= 1.0$
- Majority vote is better than individual models

Ensemble methods

- No Free Lunch Theorem:
  - There is **no** algorithm that is always the most accurate in all situations.

- Generate a group of base-learners which when combined has higher accuracy.

Ensemble methods

- How do we generate a diverse set of base-learners (models) that complement each other?
  - e.g. using Bagging and Boosting
    - Generate new datasets by sampling from the original dataset uniformly at random (with replacement), and with different subsets of features …

- How do we combine the outputs of base learner for maximum accuracy?
    - Voting
    - Weighted combination of the outputs …

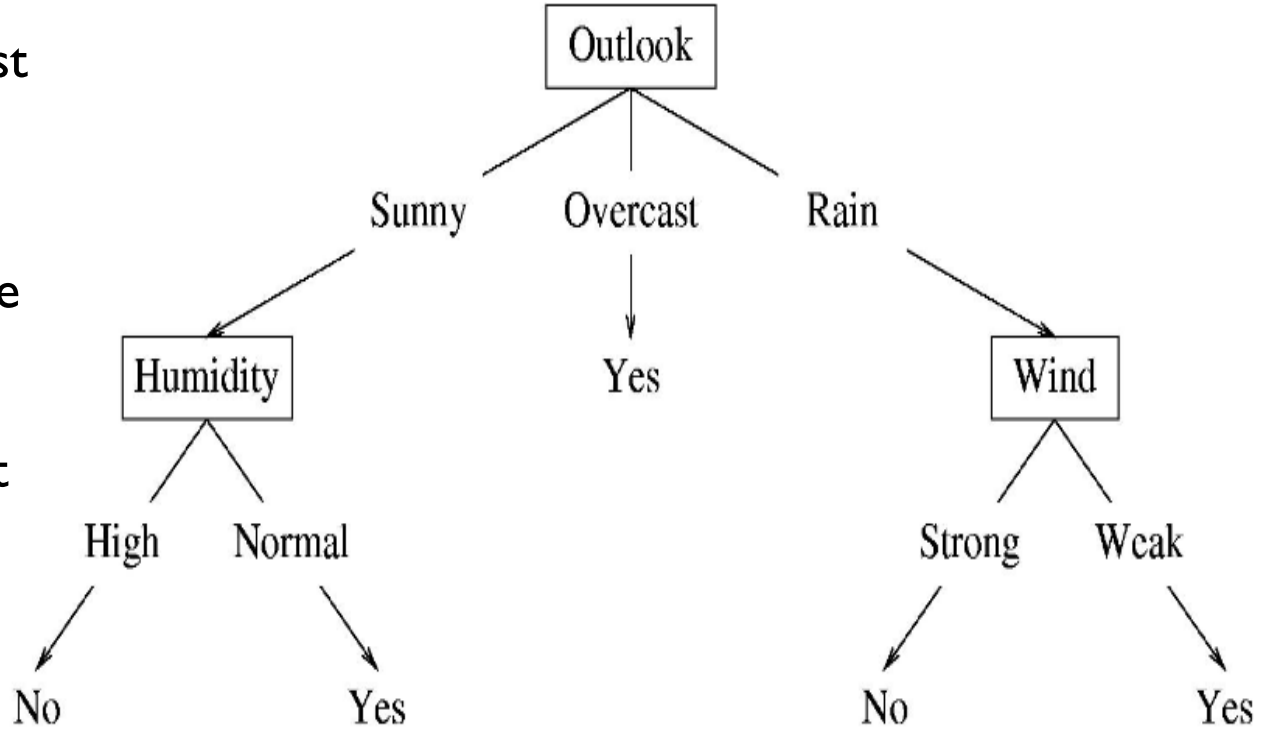
Ensemble Methods:
Random Forest

Random Forest

Reminder about Decision Tree

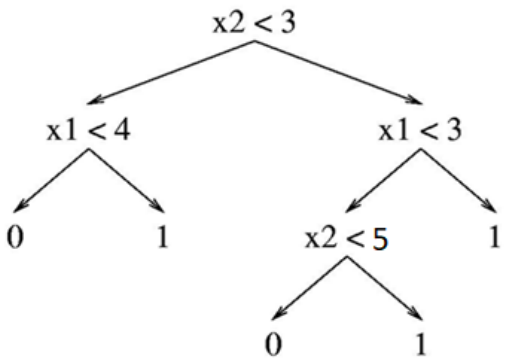
- Random forest is an ensemble of decision trees.
  - Reminder about decision trees:
    - Here is an example of a possible decision tree:

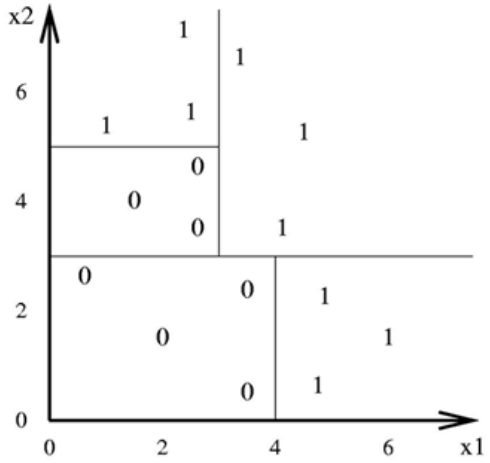
- Each internal node: test one feature x_i
- Each branch from a node: selects one value (or interval) for x_i
- Each leaf node: predict the output.



Random Forest

Reminder about Decision Tree





$$\text{Entropy} = -\sum_{i=1}^{c} p_i \log_c p_i$$

where c is the number of classes (2 in this example), p_i is the proportion of samples from the i^{th} class, and log_c is the with log base c.

- For each node, we find the feature x_i and a threshold value on that feature, that splits the samples assigned to the node into two subsets so that the label purity (i.e. homogeneity) within these subsets is maximized.

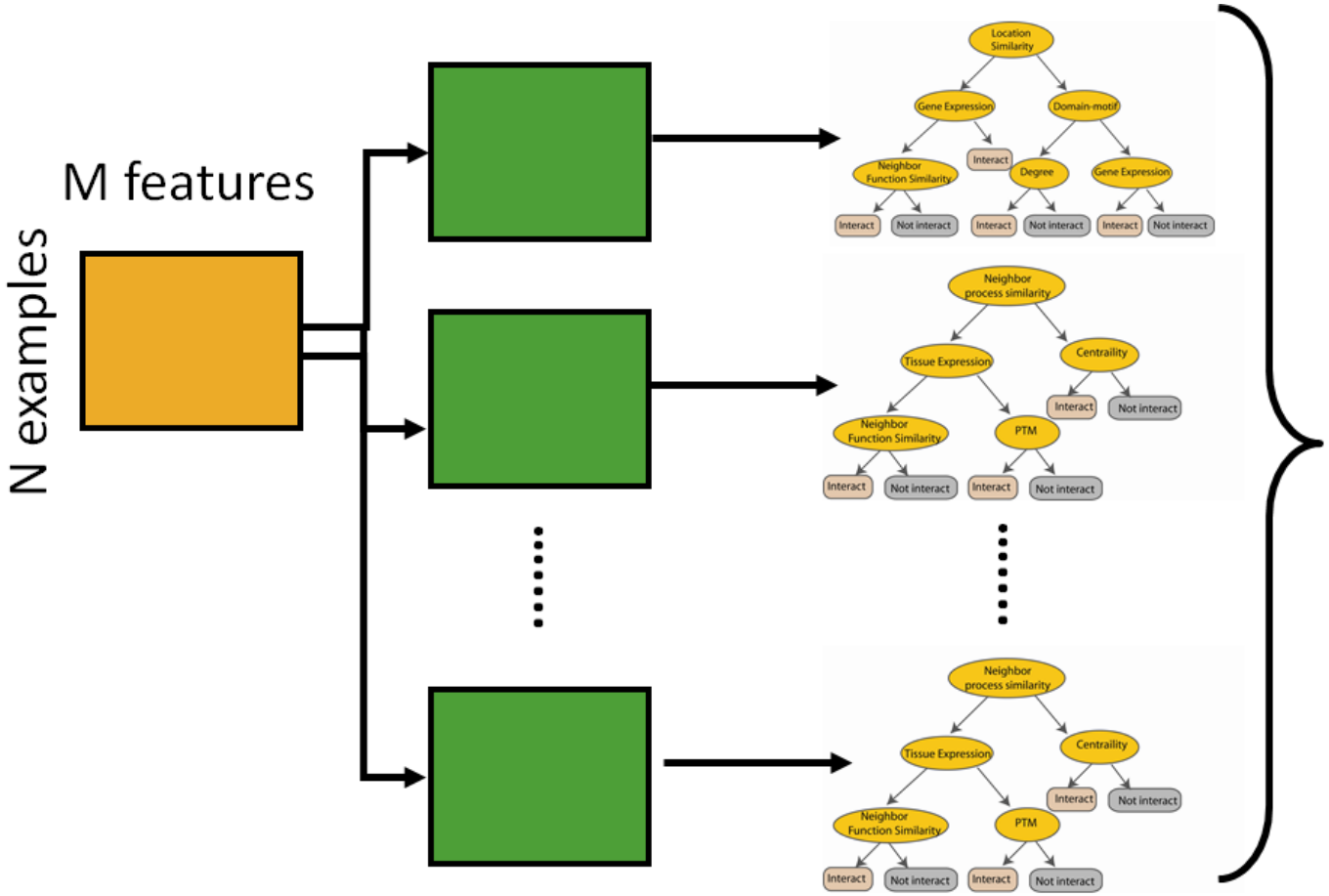
- The homogeneity is measured using (e.g.) entropy:
  - If a subset is completely homogeneous the entropy is zero.
  - If a subset is equally divided (has same proportion of different labels) then it has entropy of one.

Random Forest

- Simplified Algorithm
  - Choose T: the number of trees in the ensemble.
  - Choose $m' < m$ (m is the number of total features) : m' is the number of features used to calculate the best split at each node (typically 20%).
  - For each tree
    - Choose a training set by choosing n times (n is the number of training examples) with replacement from the training set.
    - For each node, randomly choose m' features and calculate the best split (e.g. based on the entropy measure).
  - Use majority voting among all the trees to predict.

Random Forest

Collection of diverse trees based on various subsets of samples and features …

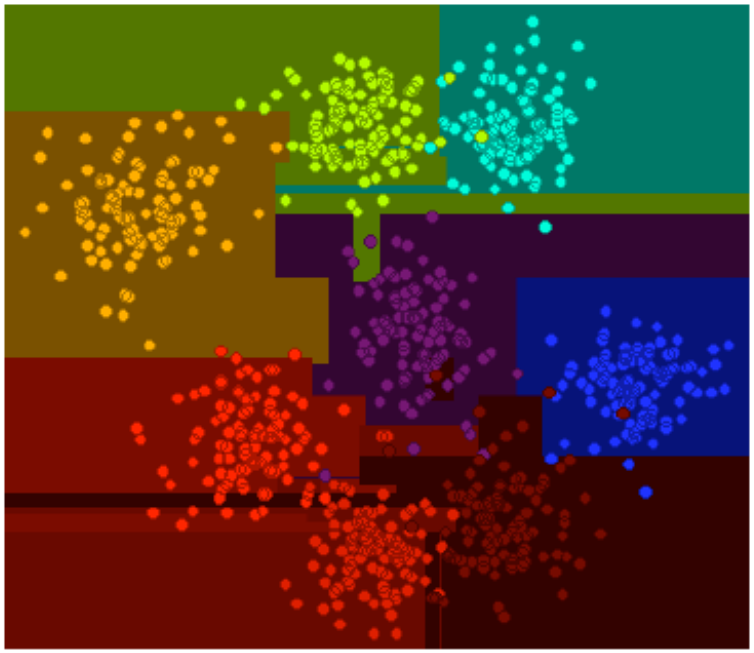


Take the majority vote

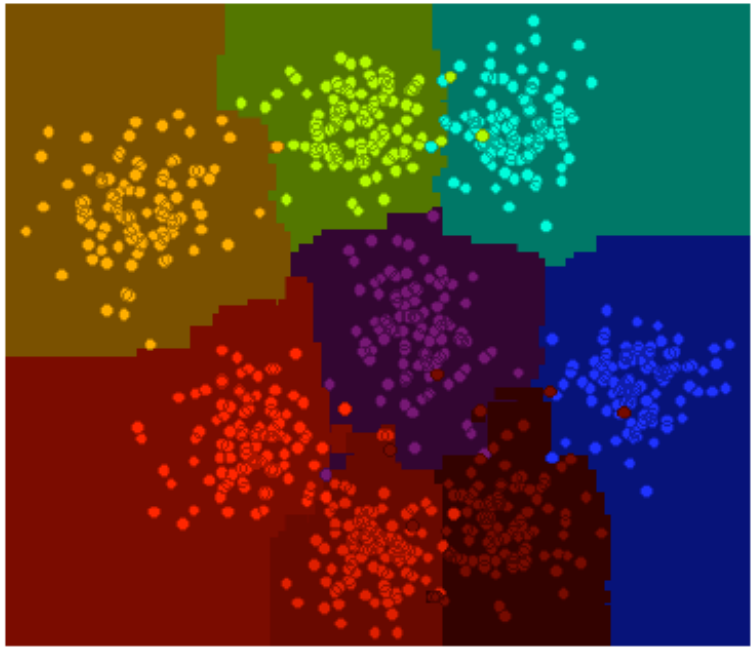
We can also give a probability $P(y|x)$ based on the proportion of trees that predicts the same label y for x.

One Tree vs. Random Forest

Decision boundary:



with one Tree



RF with 100 Trees

Random Forest and Features Importance

- Random Forest can compute the importance of each feature. So, it can be used for feature selection.

- This is done using the mean decrease impurity:
  - As we know, Random Forest consists of a number of decision trees:
    - Every node in the decision trees is a condition on a single feature, designed to split the dataset into two sets.
    - The measure based on which the (locally) optimal split (condition) is chosen is called impurity (e.g. based on entropy, or information gain, or Gini impurity).
    - Thus when training a tree, it can be computed how much each feature decreases the weighted impurity in a tree.
  - For the forest, the impurity decrease from each feature can be averaged and the features are ranked according to this measure.

Summary

- It is often a good idea to combine several learning methods.
- We want diverse classifiers, so their errors cancel out.
- However, remember, ensemble methods do not get free lunch too …