MSc - Data Mining

Topic 08 : Advanced Classification

Part 01 : Classification2-Overview

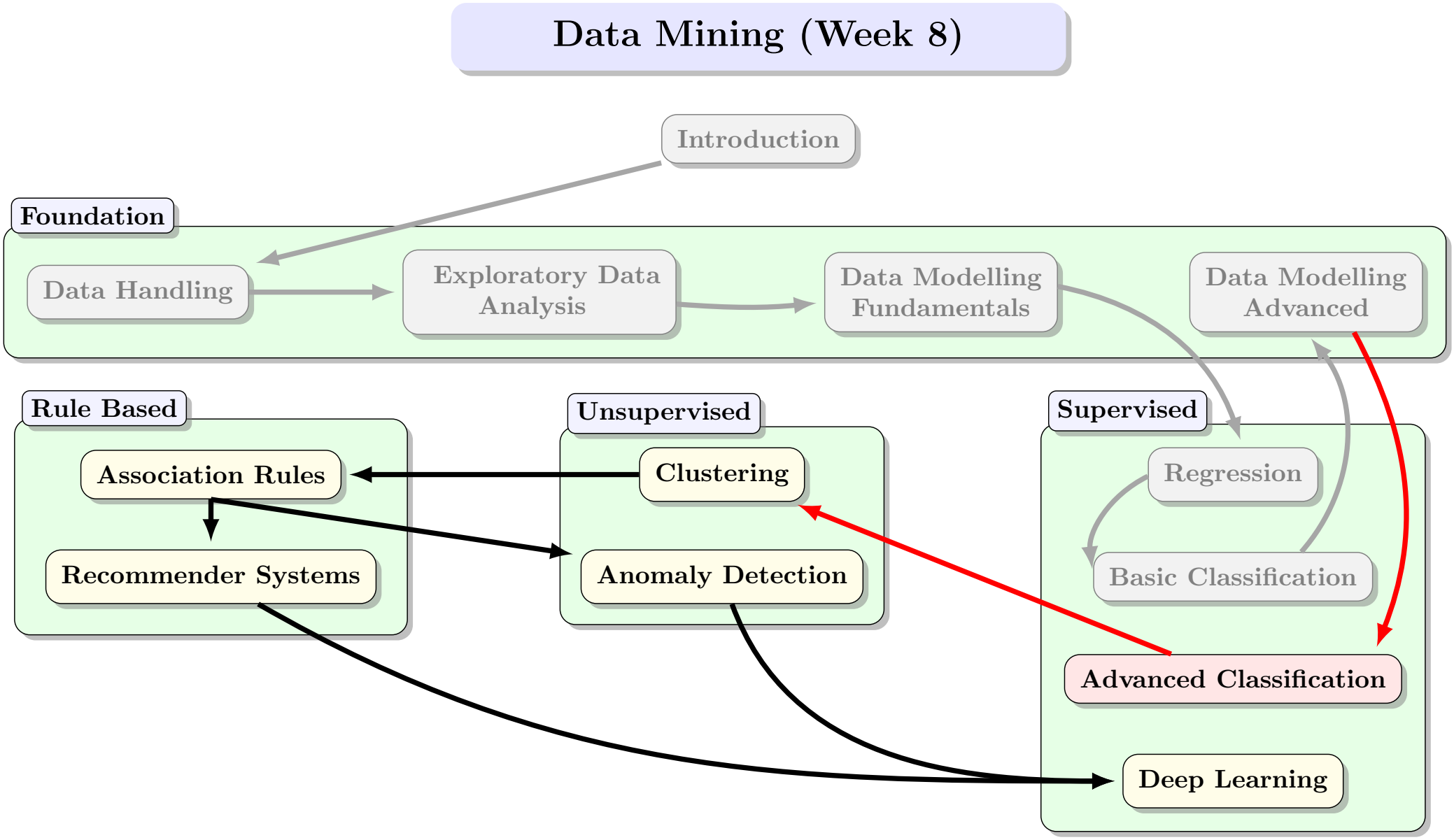
Dr Bernard Butler and Dr Kieran Murphy

Department of Computing and Mathematics, WIT.
(bbutler@tssg.org; kmurphy@wit.ie)

Spring Semester, 2021

Outline

- Decision Trees
- Ensemble classifiers (Bagging and Boosting)
- Support vector machines (SVM)


Data Mining (Week 8)
Introduction
Foundation
Data Handling
Exploratory Data Analysis
Data Modelling Fundamentals
Data Modelling Advanced
Rule Based
Association Rules
Recommender Systems
Unsupervised
Clustering
Anomaly Detection
Supervised
Regression
Basic Classification
Advanced Classification
Deep Learning

Classification2-Overview — Summary

This Week's Aim

This week's aim is to introduce some more advanced techniques used for classification. Remember: you have already met *logistic regression*, *Naive Bayes* and *k nearest neighbours* in Week 6 with Kieran
The new approaches are:

- A technique that uses a series of questions to classify a data set (Decision Trees)
- A technique that combines a suite of weak classifiers into a strong classifier (ensemble classifier)
- A technique that "pushes the boundary": Support Vector Machine (SVM) classifier

These are three of the Top 10 algorithms in data mining (**WuKumarRossQuinlanEtAl2008**), each with its own strengths and weaknesses.

This Week's Data

Remember that Classification is concerned with predicting an entity's class membership (its label) based on attributes of that entity. The following data sets are used (some have already been used by Kieran):

- Iris data: predicting which of three species a given flowering plant is, based on measurements of its sepals and petals.
- NIST handwritten digits data: recognising a digit based on its scanned raster image of pixel intensities
- Pima diabetes dataset: predicting whether someone has diabetes or not, based on their BP, BMI, etc.

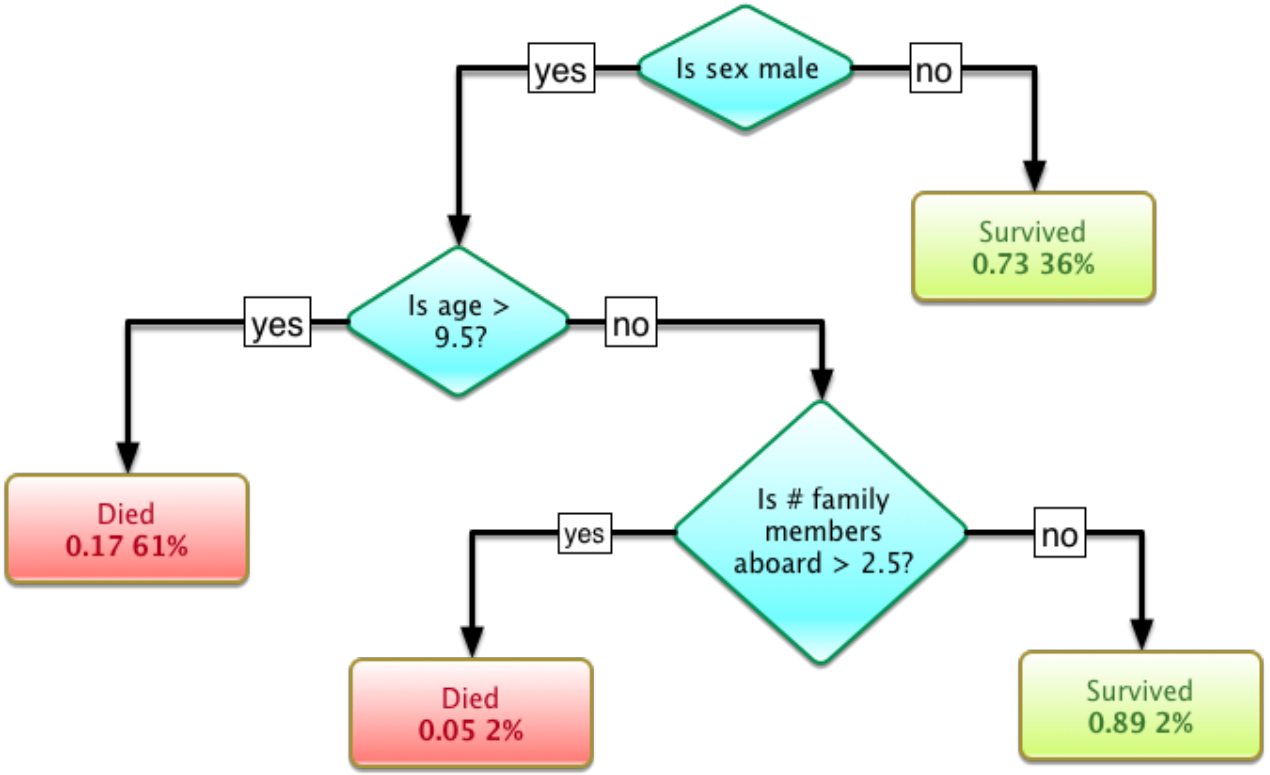
Motivation

Twenty Questions is a powerful way of learning (identifying something)

Can it be used to predict categorical variables?

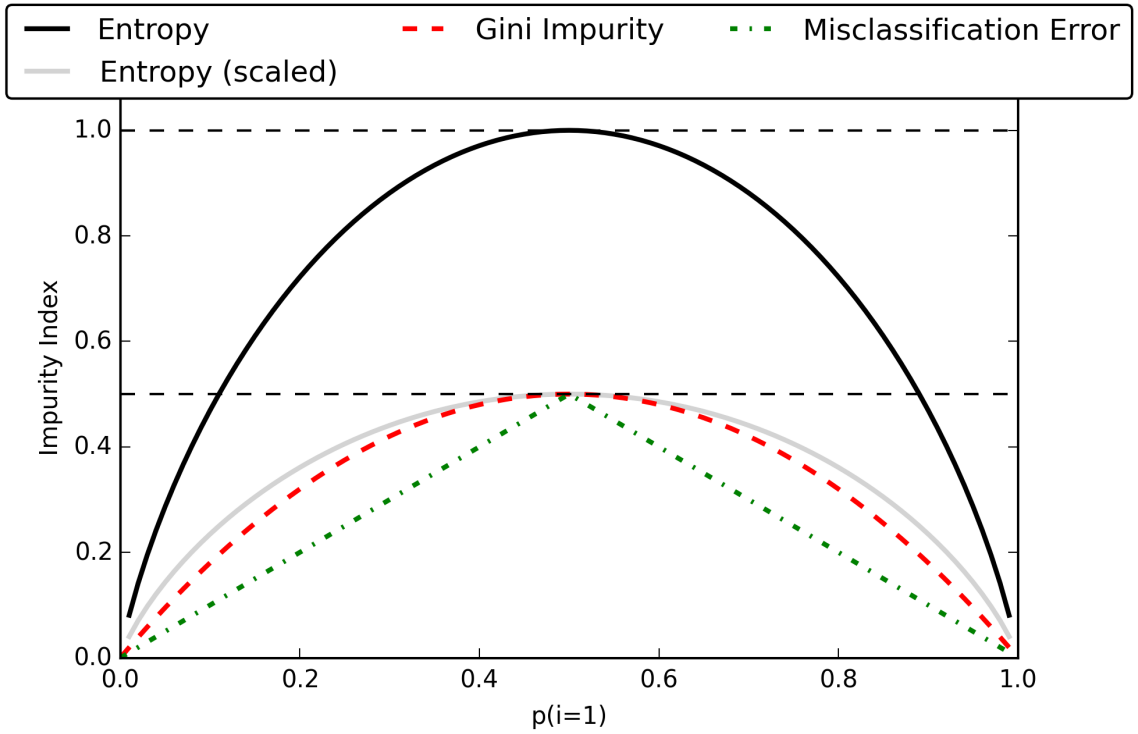
- Assume we have a set of l labels to assign to n_{test} data observations
- During training we repeatedly partition the training set using a sequence of ever finer rules
- The resulting decision tree generates a mapping from *attributes* of an item (the mapping is defined by a path from root to leaf) to conclusions about the item's target value (represented in the leaves).
- The rules which generate the binary splits are applied in a greedy fashion and are intended to reduce the *impurity* in each nodes' children as quickly as possible
- the algorithm proceeds top-down from the root (all data), recursively generating rules as it goes
- Prediction is simple: the rules are applied along the path from root to leaf. The predicted class value is either the most frequent value at the leaf, or the leaf's probability vector.

Classification tree example: Titanic survival



- First split is on Sex, as that attribute was the most important predictor of survival.
- Leaves show the probability of survival and the percentage of training observations in the leaf.
- Percentages sum to 100% (approximately), as the leaves partition the training data.
- Leaf colour indicates p(survival) $\approx$ 1 (green) or p(survival) $\approx$ 0 (red)

Classification tree metrics for rule building



Let p_i be the probability of an item with label $1 < i < J$ being chosen. Then the *GINI* impurity is $1 - \sum_{i=1}^{J} p_i^2$. Worst case (maximum impurity): labels are uniformly distributed by feature.
The alternative *information gain* is a more complex: it is the change in information entropy H from a prior state to a state that takes some information as given.

Sidebar: Entropy and Information Gain

Definition 1 (Entropy)

Entropy is a concept from *thermodynamics* and *information theory*. Here it measures the *impurity* of a collection of items. It ranges from 0 (only 1 item, possibly repeated) to 1 (equal numbers of each item type). Mathematically, it is defined for *one attribute* T as $H(T) = -\sum_{j=1}^{J} p_j \log_2 p_j$, in a collection of size N where there are J unique elements of T, hence $p_j = \frac{n_j}{N}$ where there are n_j elements of type j. For *two attributes* T and X, $H(T,X) = \sum_{c \in X} P(c)E(c)$ where each c represents a level of the X attribute.

Definition 2 (Information Gain)

Information Gain measures the decrease in entropy after a dataset is split on an attribute. It is defined as $G(T,X) = H(T) - H(T,X)$, where $H(T)$ is the entropy at the parent node, and $H(T,X)$ is the entropy after the split by candidate attribute X.

Sidebar: Entropy: PlayTennis example data

outlook	temp	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Source: Mitchell, *Machine Learning*, 1997.

Sidebar: Entropy: PlayTennis example calculations

Example 3 (H(play))

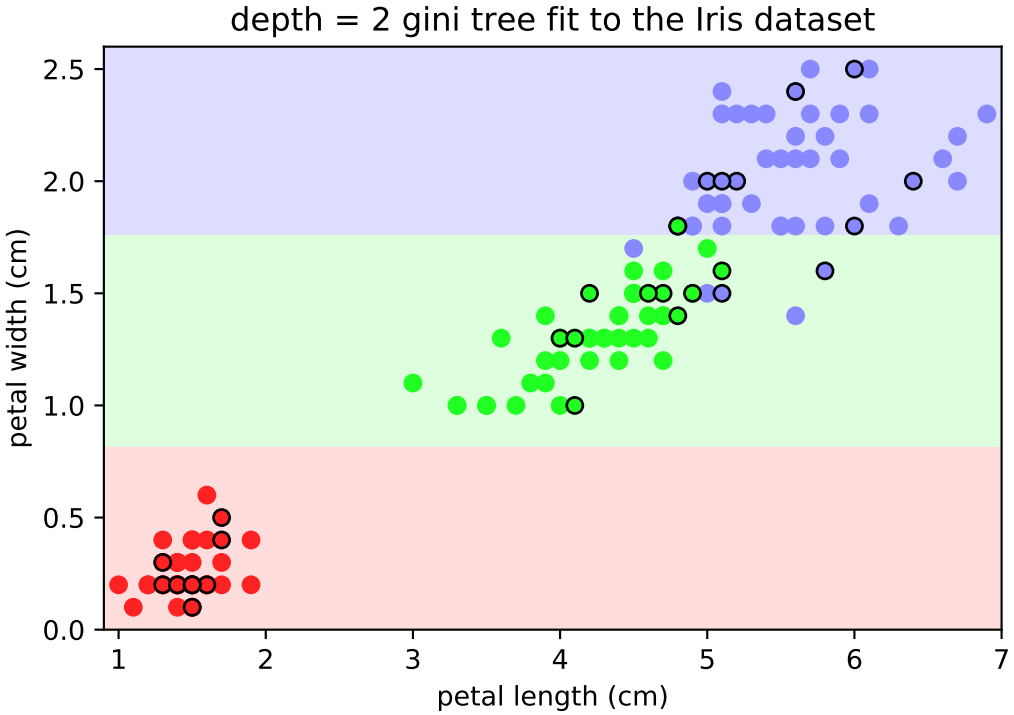
$$\begin{aligned} H(\text{play}) &= -\left(p(\text{play}=\text{yes})\log_2 p(\text{play}=\text{yes}) + p(\text{play}=\text{no})\log_2 p(\text{play}=\text{no})\right) \\ &= H_{9,5} \\ &= -\left(\frac{9}{14}\log_2\left(\frac{9}{14}\right) + \frac{5}{14}\log_2\left(\frac{5}{14}\right)\right) \approx 0.94 \end{aligned}$$

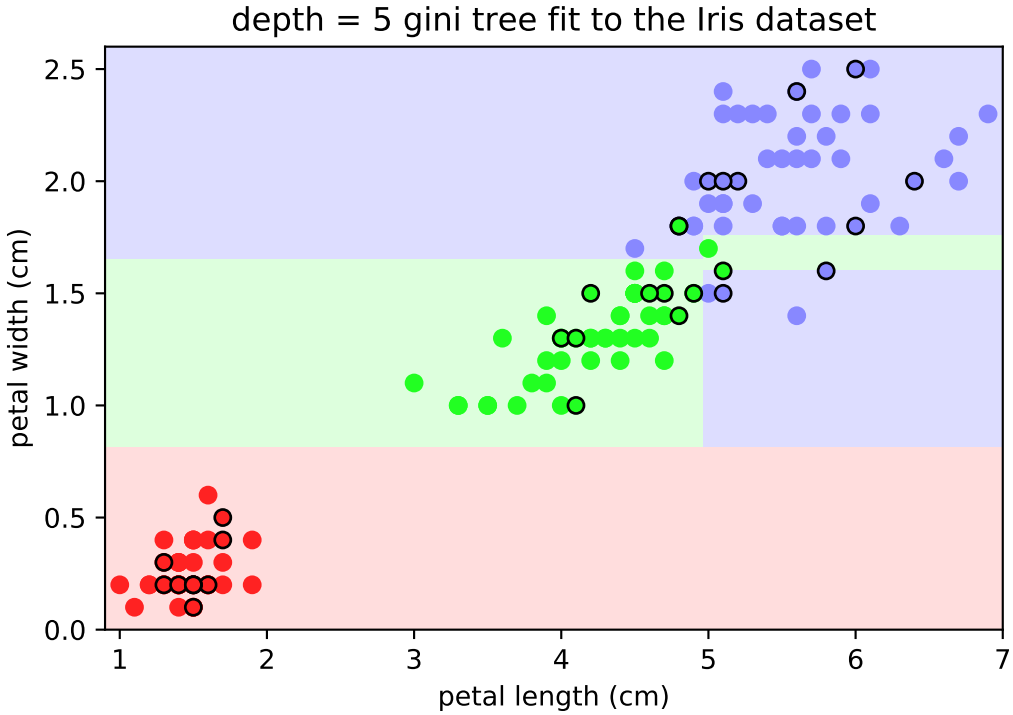
Example 4 (H(play,outlook))

$$\begin{aligned} H(\text{play},\text{outlook}) &= p(\text{outlook}=\text{sunny})H(\text{play}\&(\text{outlook}=\text{sunny})) + \ldots \\ &= p(\text{outlook}=\text{sunny})H_{3,2} + p(\text{outlook}=\text{overcast})H_{4,0} + \ldots \\ &\approx \frac{5}{14}0.97 + \frac{4}{14}0 + \frac{5}{14}0.97 \\ &\approx 0.69 \end{aligned}$$

When growing decision trees, at a given node we search over the attributes for splitting, and choose the one that gives the maximum information gain, until we reach a leaf, which has an entropy of zero.

Classification tree examples: Iris Data





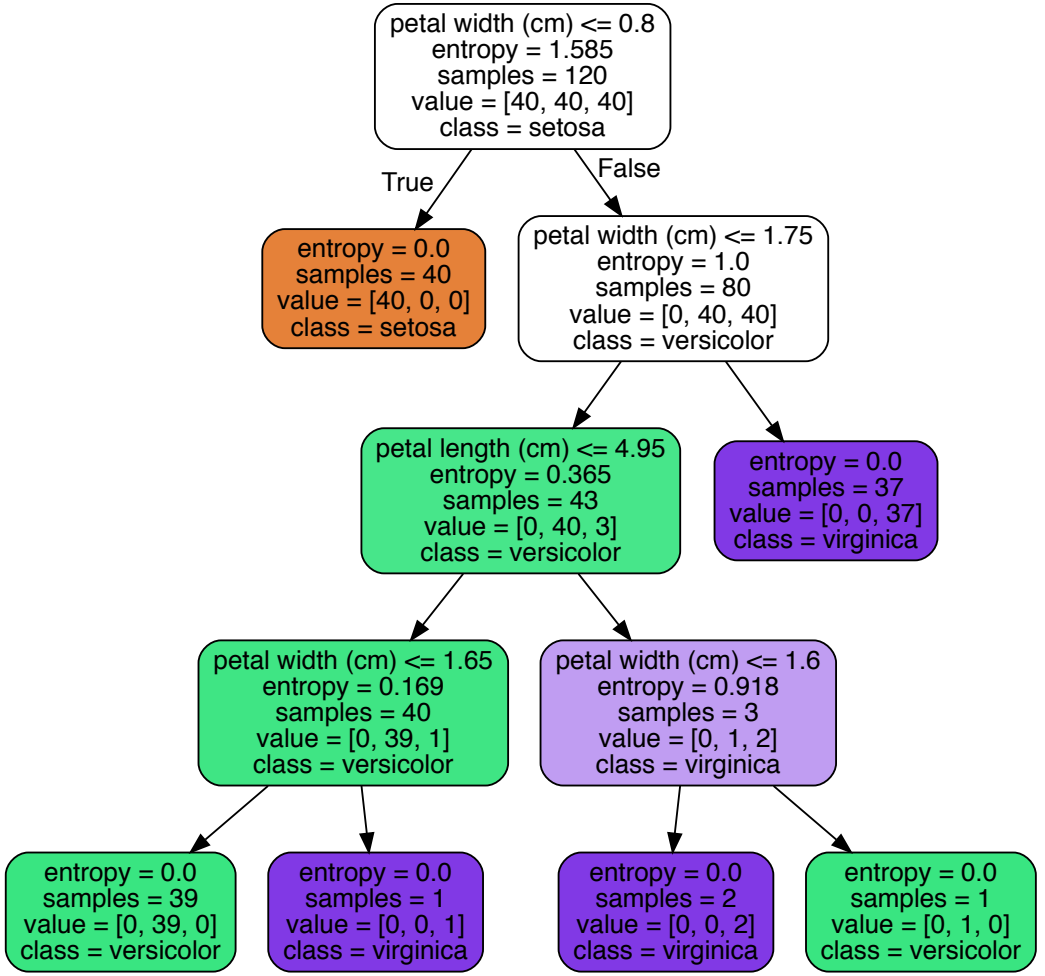
Note the rectangular regions (because each split is over one variable) and the greater complexity when the maximum depth of the tree increases.

Points within a dark circle represent test data, with the main colour of the point indicating its species label.

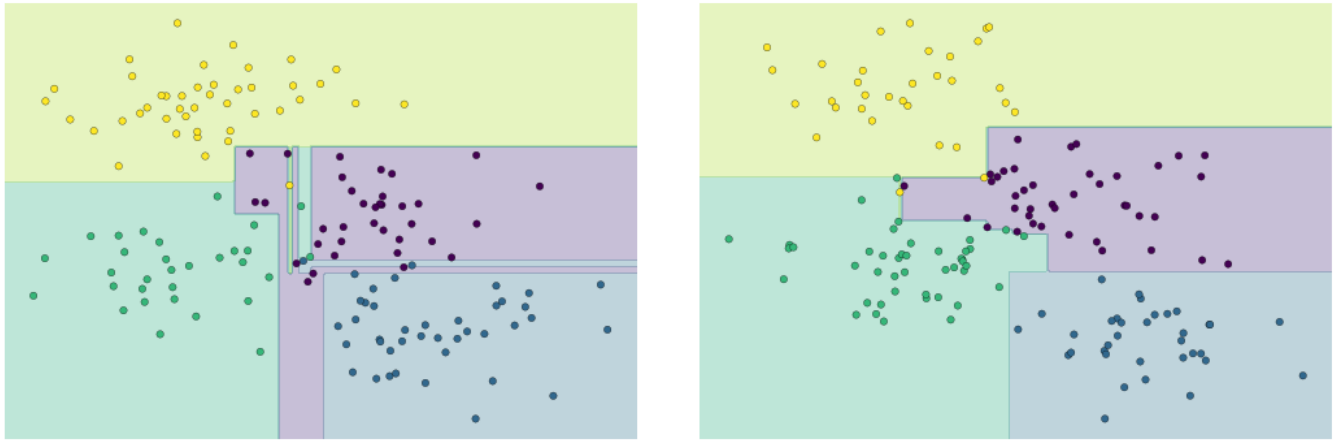
The choice of metric (Gini impurity or Information Gain) makes only slight changes to fit.

Classification tree view: Iris Data

Note that the leaf nodes are pure (entropy=0) and are coloured according to predicted value (species label): brown for *I. setosa*, green for *I. versicolor* and purple for *I. virginica*.



Be careful of overfitting. . .



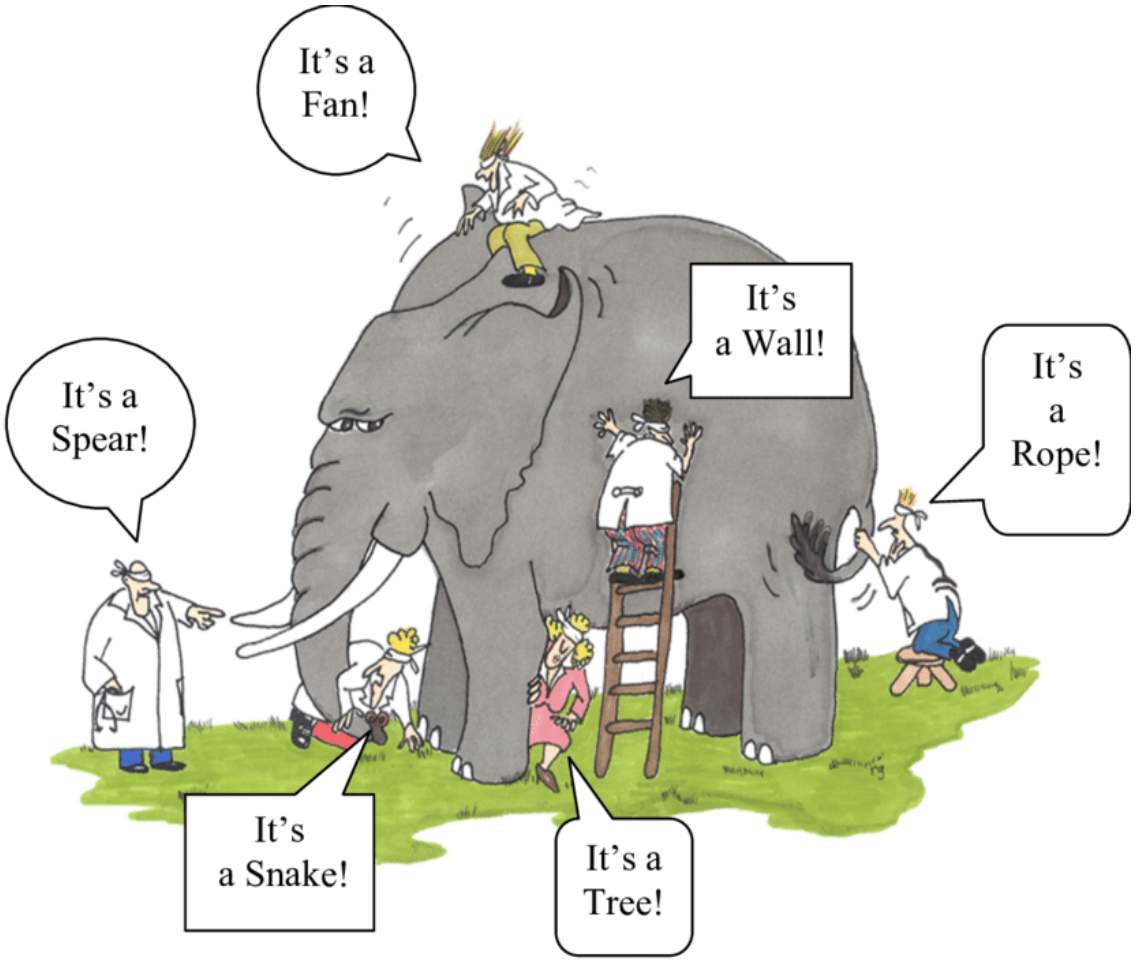
- Given two very similar (generated) data sets, all leaves in each fitted decision tree are *pure*.
- The resulting trees look very different.
- This sensitivity to the noise in the data is characteristic of *overfitting* (high variance).
- Control by a) limiting depth or b) limiting number of leaves.

Classification trees in python

```
tree = DecisionTreeClassifier(criterion=criterion, max_depth=treeDepth, random_state=0)
tree.fit(Xtrain, ytrain)
y_treeTest = tree.predict(Xtest)
print(accuracy_score(ytest, y_treeTest))
print(confusion_matrix(ytest, y_treeTest))
print(classification_report(ytest, y_treeTest, digits=3))
```

After creating the classifier object, fit the training data and then use the fit to predict yTest from xTest. I have also shown how to get some diagnostic output. Similar diagnostics can be obtained for other classifiers.

Motivation: Combining Classifiers



By combining classifiers, we can get a better result than each classifier on its own...

Ensemble learning overview

Intuition

- Training a discriminative classifier is equivalent to searching for a function mapping features (X) to classes (Y).
- The function search space is enormous!
- Rather than finding a single *best* classifier, is it possible to find several *good* classifiers and combine them into a classifier that outperforms each of its components?

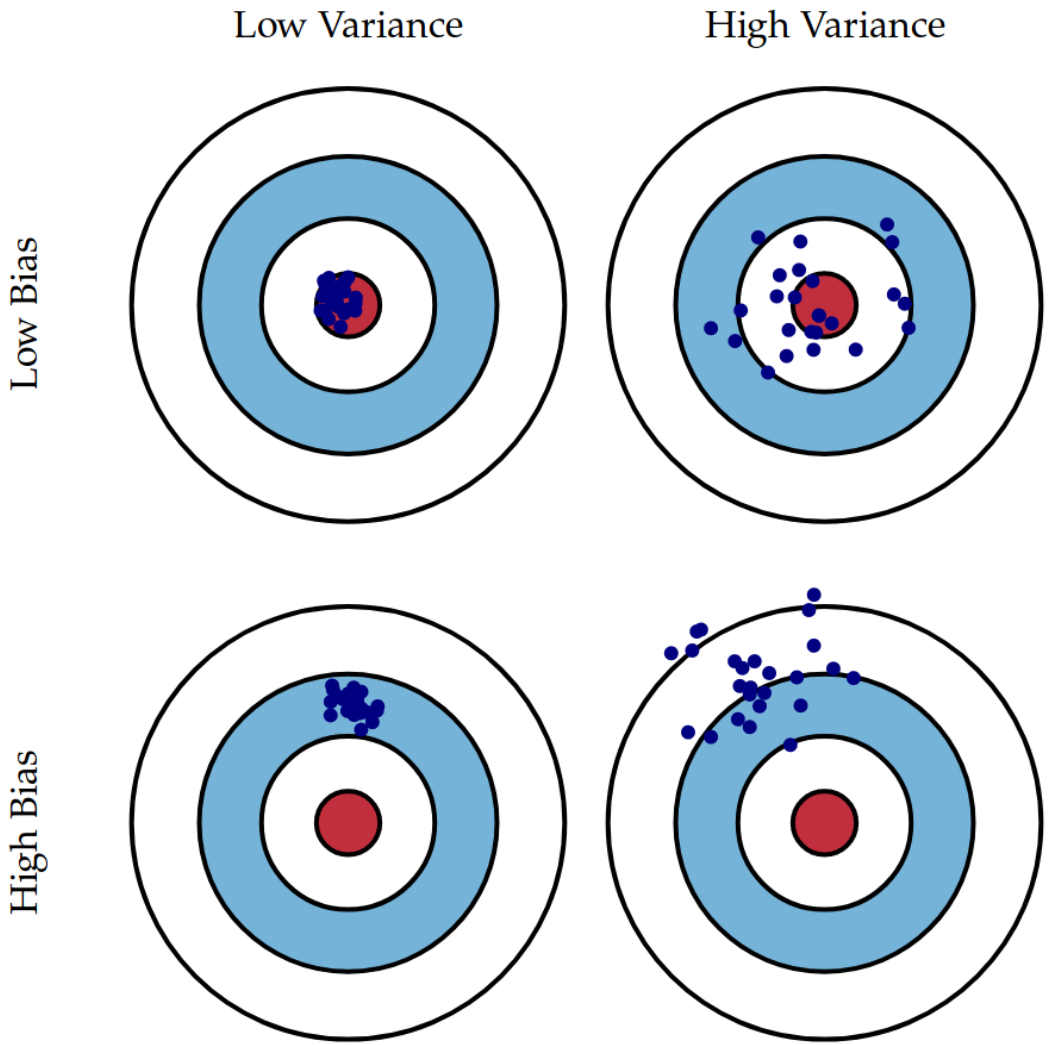
This is an example of a divide and conquer approach. For example, *edge cases* can be handled by relatively simple classifiers, but they need to be combined with more general classsifiers for good overall performance.

General considerations

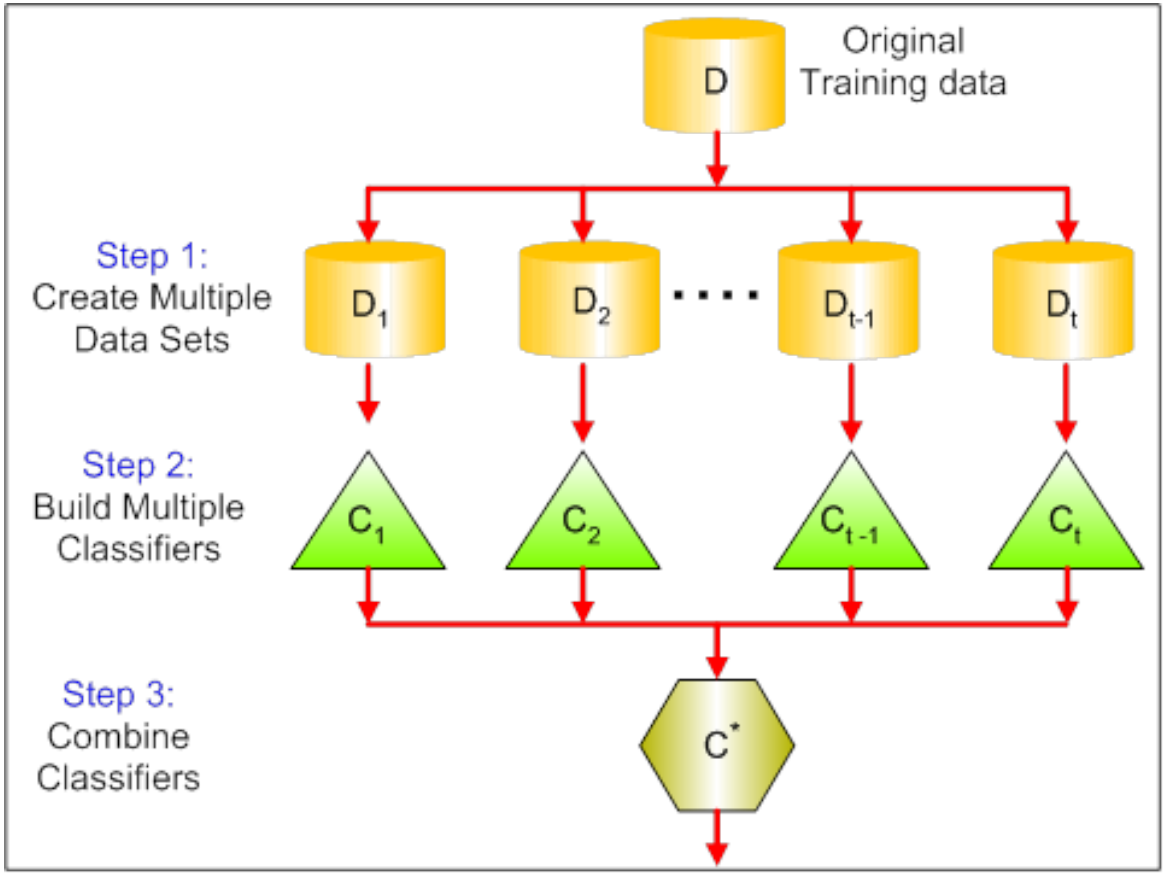
- ensemble diversity and its tradeoffs
- weak versus strong learners
- ensemble size: when do we have enough learners?

Weak learners and ensemble learning

- A *weak learner* has high variance, high bias or both.
- Example: KNN with k set too high or too low, decision tree with too many or too few leaves
- Sometimes it is just slightly better than random guessing.
- But under some circumstances, a suite of weak learners can be combined to act as a composite *strong learner*.
- Can be used for regression and classification, but classification is more common.
- Ensemble algorithns are *bagging*, *boosting* and variants.



A canonical ensemble model



Source: www.analyticsvidhya.com

This structure is used in bagging and boosting classifiers

Bagging background

Definition 5 (Bootstrapping)

Given a sample (training data with N_0 observations, say), bootstrapping (**Efron1979**) draws B samples, each of size N_s, with replacement from the original sample. Because of replacement, B is not limited by N_0, and an observation can appear in more than one bootstrap sample. By the *Law of Large Numbers*, statistics combined from the bootstrap samples can estimate those of the population.

Definition 6 (Weak learner)

A weak learner is quick to learn and predict (e.g., a decision tree with restricted height, or a logistic regressor with few terms) and its error rate is strictly less than 0.5 for all training inputs.

Definition 7 (Model Combination)

Given a set of predictions from many weak learners, the weighted average prediction generally is more correct and the *combined predictor* has lower variance than any of the individual predictors, c.f., "Wisdom of the crowd" phenomenon(??).

Bagging algorithm

Definition 8 (Bootstrap aggregation (Bagging))

$k \leftarrow$ numberOfBootstrapSamples
$N \leftarrow$ bootstrapSampleSize
$X \leftarrow$ trainingData
for $i \leftarrow 1 : k$ **do**
    $D_i \leftarrow$ deriveBootstrapSample(X, N)
    $C_i \leftarrow$ trainClassifier(D_i)
end for
$C^* \leftarrow \text{argmax}_y \sum_i \delta(C_i(x) = y)$

where $\delta(\cdot) = 1$ if $\cdot$ is true and is 0 otherwise.

- The $\delta(\cdot)$ aggregation function is used for classifiers and simple averaging is used for regression.
- The best known bagging algorithm for classification is RandomForest (equivalent to `BaggingClassifier` with `DecisionTrteeClassifier` as the weak learner).
- In `sklearn`, it has basically the same API as other classifiers and is invoked using `from sklearn.ensemble import RandomForestClassifier` and `clf = RandomForestClassifier(n_estimators=10)`, where $k = 10$ in this example.

Boosting algorithm

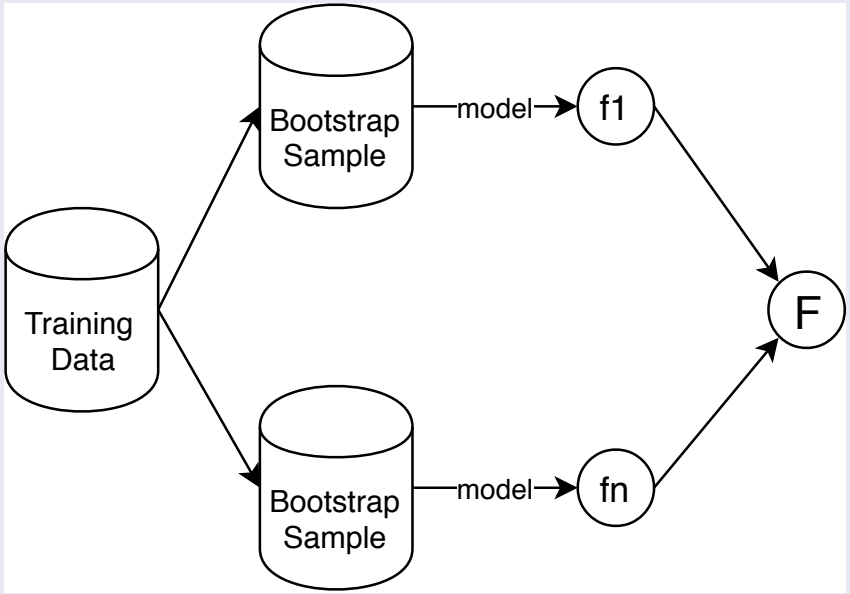
Definition 9 (Boosting)

Boosting (**FreundSchapire1999**) is another ensemble classifier. Unlike boosting, observations are reweighted rather than resampled. Weak classifiers are applied sequentially to the training data. Records that were misclassified by the previous iteration/model are given more weight. Successive models are weighted according to their success (AdaBoost) or gradient performance (GradientBoost). Variants include Adaboost, XGBoost, etc.

- The leading classification algorithms for the Netflix prize, many kaggle competitions, etc. tend to be boosted ensemble classifiers.
- They are available in `sklearn`: `from sklearn.ensemble import AdaBoostClassifier` and `clf = AdaBoostClassifier(n_estimators=100)`

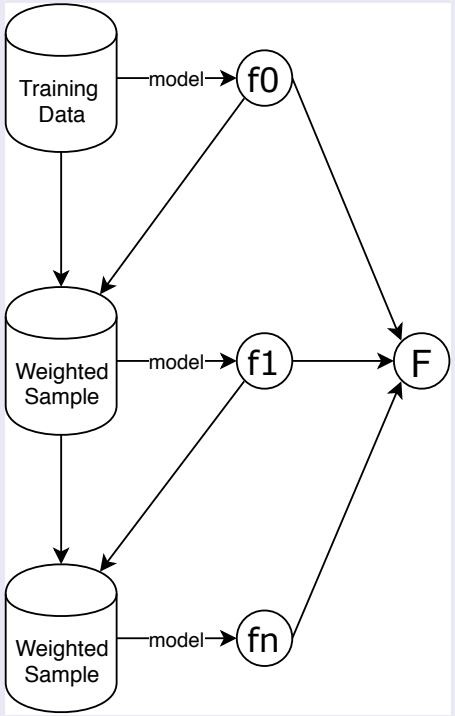
Comparison of bagging and boosting

Bagging



- Resamples the training data
- Observations are unweighted (uniform distribution)
- Classifiers work in parallel

Boosting



- Reweights the training data
- Observations are weighted by “difficulty”
- Classifiers work in sequence

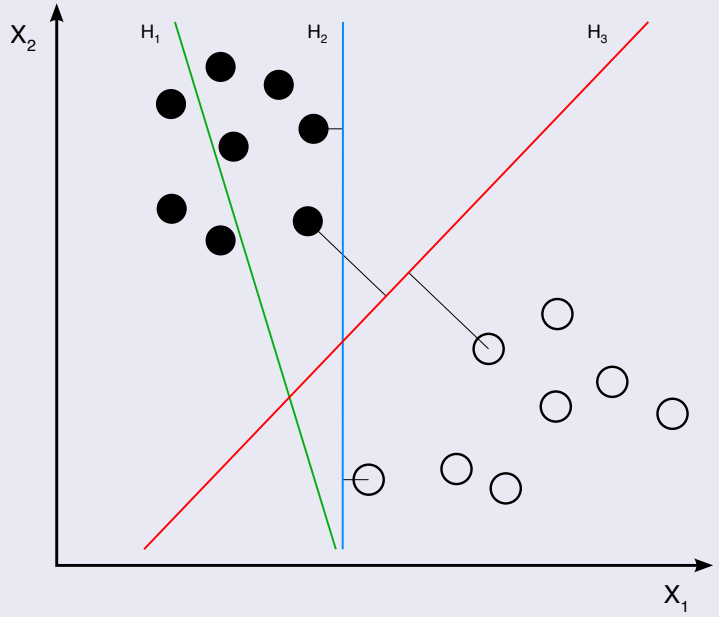
Review of bagging and boosting

- Bagging uses weak learners with *high variance and low bias*, and *bagging reduces the variance* by statistical aggregation
- Boosting uses weak learners with *low variance and high bias*, and *boosting reduces the bias* by iteratively reweighting the learners
- Hyperparameters for each include: choice of weak learner type and number of estimators (weak learners)
- Hyperparameters for boosting include: learning rate (aka shrinkage factor); if less than 1, updated models are downweighted; stabilises the iteration but might slow convergence or even converge to local minimum
- **AdaBoost**: iterative reweighting classifiers by putting more weight on misclassified samples so that they tend to be tackled by the next classifier, keep doing this until convergence.
- **GradientBoosting**: Start with a constant learner, fit a weak learner to the negative gradient of the (approximate) loss function, take a downward step so that it minimises the loss function in that direction, and continue.
- In Adaboost, “shortcomings” of the fit at any stage are identified by high-weight data points.
- In Gradient Boosting, “shortcomings” of the fit at any stage are identified by gradients of the loss function.

Classifier boundaries

Up to now, when considering classifiers, our focus has been on assigning data to classes, not on the boundaries between those classes.

Comparison of separating planes

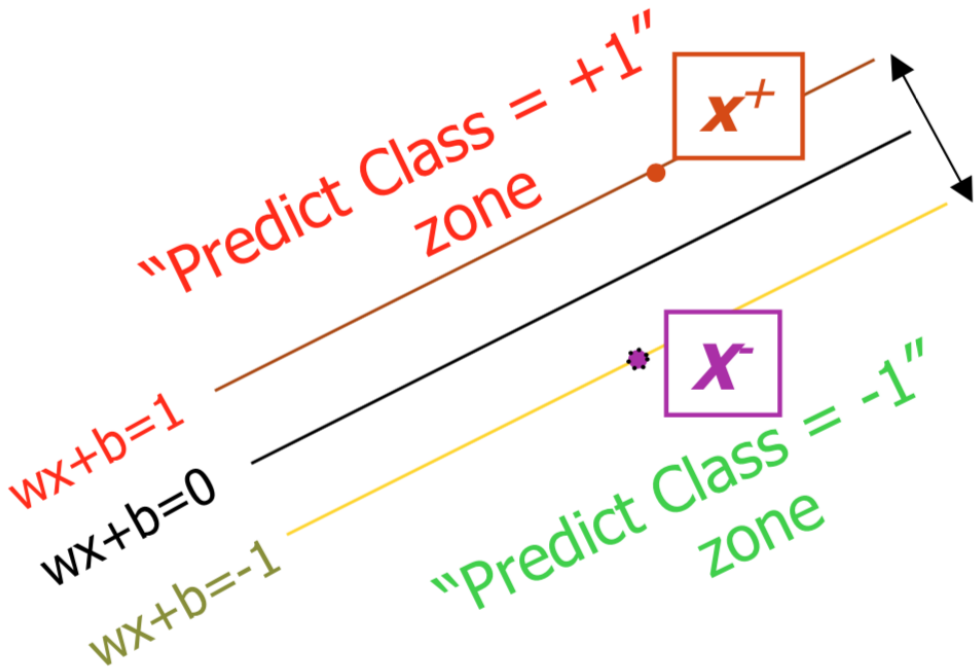


- H1 does not separate the two classes
- H2 does separate them, but only just!
- H3 is the maximum margin separating hyperplane (line in 2-D) in this case
- the corresponding classifier is a maximum margin classifier that *minimises the generalisation error* when classifying new data from the test set

Source: wikipedia

Consequently, H3 is defined by a relatively small number of observations (points) which are known as the support vectors of the classifier

Linear SVM geometry



- $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$ are the support vectors
- Suitably scaled and shifted, the margin lines are $wx + b = 1$ and $wx + b = -1$
- Note that $(wx^+ + b = 1) - (wx^- + b = -1) = w(x^+ - x^-) = 2$.
- Also the Margin $M = (\boldsymbol{x}^+ - \boldsymbol{x}^-).\boldsymbol{w}/|\boldsymbol{w}|$, i.e., the projection of the vector between the two support vectors onto a line perpendicular to the separating hyperplane
- Collecting terms, we have $M = 2/|\boldsymbol{w}|$

Linear SVM optimisation

We wish to find w and b so that each observation is assigned correctly to one of two classes, and the margin is maximised

Definition 10 (SVM iteration)

Feasibility Start with an estimate of w and b, iterate to find new feasible w and b that ensure that all training data are classified correctly: $y_i(wx_i + b) \geq 1, \forall i$, where $y_i = \pm 1$ depending on whether the point is in the +1 or -1 class.

Optimality Starting with feasible w and b, iterate to maximise $M = \frac{2}{|\boldsymbol{w}|}$ or equivalently, minimise $\boldsymbol{w}'\boldsymbol{w}/2$ while maintaining feasibility.

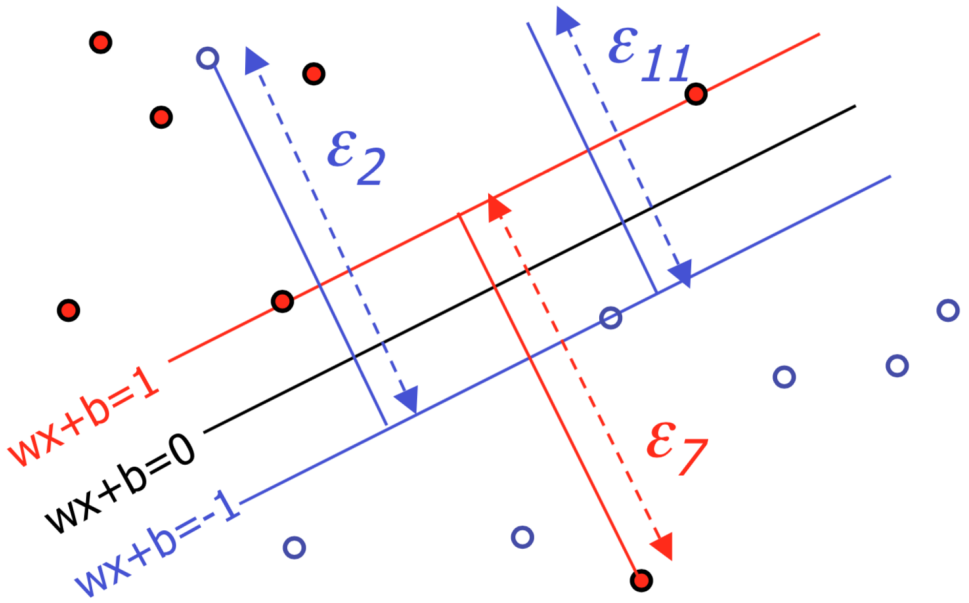
This is a constrained convex optimisation problem. Library software exists to solve it. Either there is no solution (because the classes are not linearly separable) or the solver will find the *unique* SVM classifier.

Review of basic SVM

- The SVM solver we described can be extended to higher dimensions easily
- It can be extended to multiclass (not just binary classification), by running n SVMs and aggregating (this happens transparently)
- However, the use of a hard margin makes it sensitive to noisy support vectors (data that strays across the "true" separating hyperplane)
- Solution is to introduce a soft margin, so some points are allowed to be classified as "+", say, even if they lie a small way inside the "-" region.

The soft margin is defined in terms of a set of ε vector lengths, which are nonzero for a small number of "noisy" points, some of which could otherwise be treated as support vectors and/or make the hard margin SVM problem infeasible.

Soft margin linear SVM



ε_2 allows one point to be classified as blue even though it is on the "red side" and ε_7 allows another point to be classified as red even though it is on the "blue side".

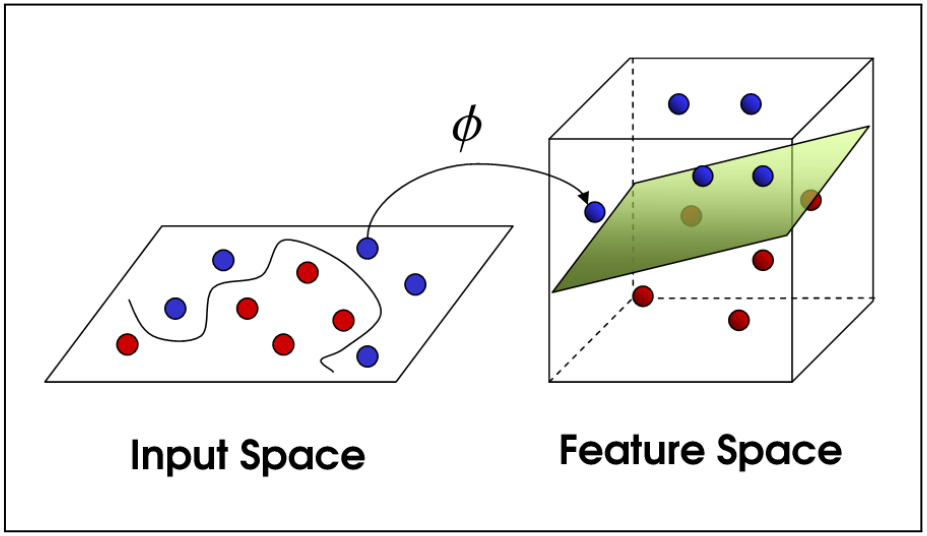
Hard margin formulation is

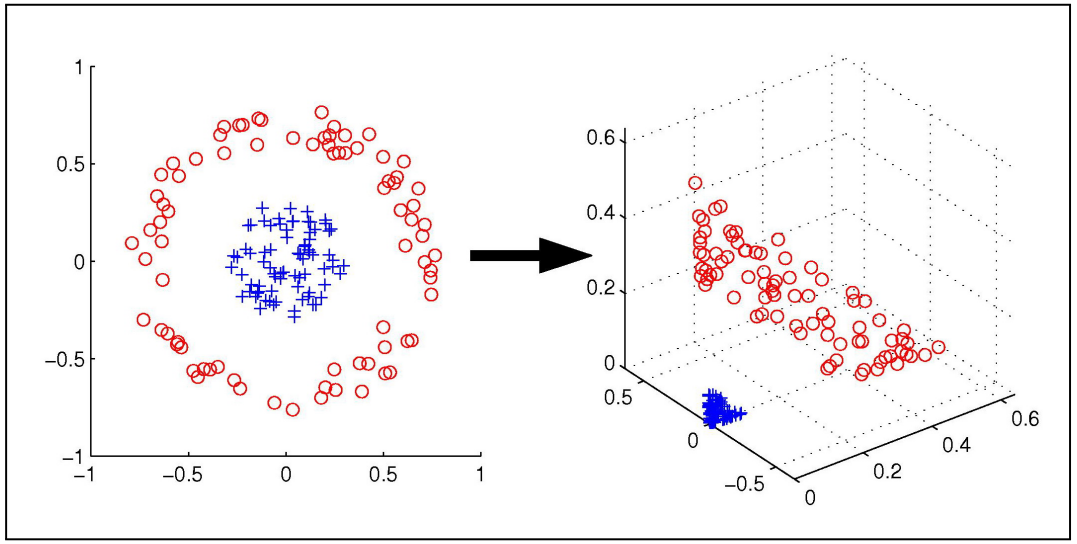
- Find $\boldsymbol{w}$ and b that minimise $\frac{1}{2}\boldsymbol{w}'\boldsymbol{w}$
- Subject to $y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) \geq 1$ (hard margin feasibility)

Soft margin formulation is

- Find $\boldsymbol{w}$ and b that minimise $\frac{1}{2}\boldsymbol{w}'\boldsymbol{w} + \lambda \sum_i \varepsilon_i$
- Subject to $y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) \geq 1 - \varepsilon_i$ (soft margin feasibility), and
- $\varepsilon_i \geq 0$ for all points indexed by i, and
- $\lambda \geq 0$ is like a regularisation parameter

Nonlinear boundaries: Kernel SVM





Definition 11 (Kernel trick)

When the data is not linearly separable in the input space, a suitable transformation into a new feature space (often with more dimensions) can make it so.

- Example kernels include *linear* as seen already, *polynomial* (for curved boundaries), *radial basis function* (for classes "enclosing" others, etc.)
- By inspecting the data, the user should identify a suitable transformation, hence kernel and SVM can then work, as before, with the transformed data/generalised kernel
- SVM is available `sklearn`: `from sklearn import svm` and `clf = svm.SVC()`.

Review of SVM

- Although Euclidean distance is frequently used, there is flexibility in choosing a similarity function
- Since the solution is defined in terms of (a relatively small number of) support vectors
  - even with large data sets, the solution is sparse
  - suited to high dimensional data because we calculate the inner product over the dimensions ("squashing them") so complexity is largely independent of dimension
- Soft margin approach can help to deal with noisy data and to minimise overfitting
- Convex optimisation problem ensures the optimisation process leads to just a single global solution
- Very flexible regarding nonlinear boundaries and feature selection (kernel SVM)
- Often a good fit with text classification.
- *Kernel trick* also used in artificial neural networks, to enable them to classify even with nonlinear/piecewise-linear boundaries.

Summary

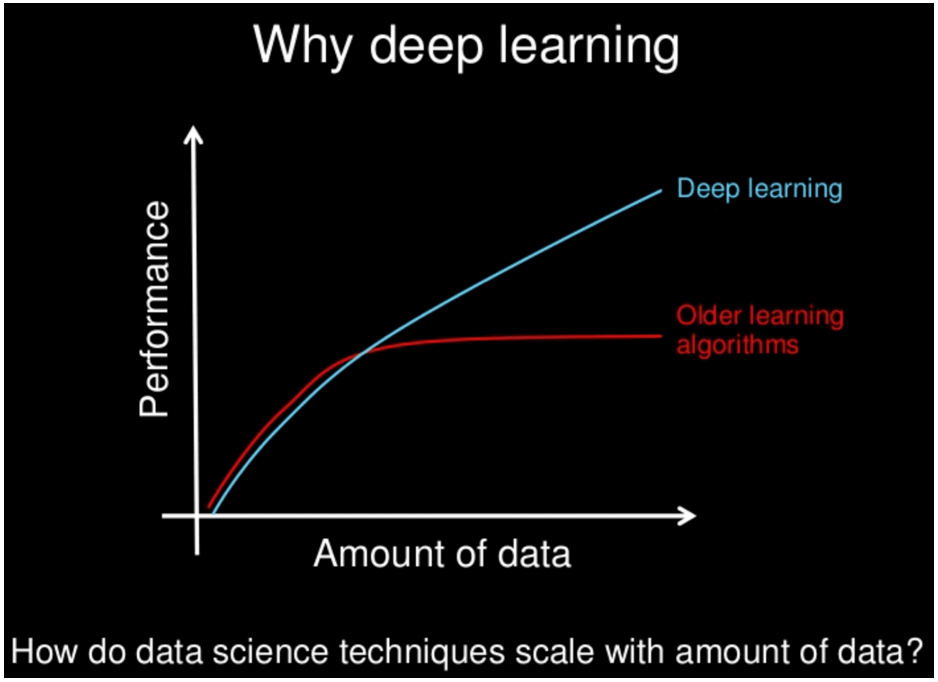
- Classification is one of the main tasks in data mining, and is a mature and well-studied field
  - Logistic regression is an extension of linear regression and benefits from its strengths
  - k-nearest-neighbours is conceptually simple (based on voting) and the lack of a model (lazy learning) means it responds better to data drift
  - Naïve Bayes offers a probability-based generative model, able to work from data summaries, ideal for text and email classification
- More advanced classifiers have their own advantages, especially in relation to high dimensional data:
  - Decision trees learn a representation that is often easily interpretable, but works better with linear boundaries
  - Ensemble methods were state of the art (2000-2012, say) and sacrifice interpretability for good performance with high dimensional data
  - SVM was state of the art (1985-2000, say) and is still extremely effective for very high dimensional problems like document classification

Other considerations

- KNN uses *lazy* learning, all other techniques above use *eager* learning (derive model from training data)
- Naive Bayes uses *generative* learning to learn how the data was generated, all other techniques above use *discriminative* learning to derive the function that assigns class labels
- For KNN and Decision Trees, the representation grows with the size of the data - that is not generally true in all other techniques above

Classification is sometimes confused with clustering - will cover this in Week 9.

But is that the last word on Classification?



Source: Andrew Ng, *Why Deep Learning*

Learning from big data

- Traditional classification algorithms eventually run out of steam as data size increases
- Shallow neural networks had been discounted in the 1980s and 1990s when trained with small data
- Deep learning to the rescue!
- Kernel SVM and logistic regression lead nicely to perceptron models, hence ANNs, hence deep learning
- Deep learning requires lots of data but the models can scale better to take account of extra data

Kieran will cover Deep Learning as the topic in Week 12.

General References