Applied Machine Learning for Business and Economics

AMLBE

Profesor : David Díaz S.

Acknowledgements:
Some slides adapted from
- Tom Mitchell
- Eibe Frank & Ian Witten
- Kan, Steinbach, Kumar
- Ricardo Gutierrez-Osuna
- Gunter Grieser

- J. Fürnkranz
- Scikit learn online material

Some advantages of decision trees are:

- **Simple to understand and to interpret.** Trees can be visualised.
- **Requires little data preparation**. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed.
- **Able to handle both numerical and categorical data.** Other techniques are usually specialised in analysing datasets that have only one type of variable.
- **Able to handle multi-output problems.**
- **Uses a white box model.** If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- **Possible to validate a model using statistical tests.** That makes it possible to account for the reliability of the model.
- **Performs well even if its assumptions are somewhat violated** by the true model from which the data were generated.
- **The cost of using the tree (i.e., predicting data)** is logarithmic in the number of data points used to train the tree.

The disadvantages of decision trees include:

- Disadvantage number 1:
- Decision-tree learners **can create over-complex trees that do not generalize the data well.**
- This is called overfitting.
- To avoid overfitting it is necessary to follow at least one of these strategies:
  - **Pruning (not currently supported in sklearn) (pre or post pruning)**
  - Setting the minimum number of samples required at a leaf node
  - Setting the maximum depth of the tree
  - **Train – test – validation or cross-validation splits (more later)**

Pruning

- Goal: Prevent overfitting to noise in the data
- Two strategies for "pruning" the decision tree:
  - *Prepruning* - stop growing a branch when information becomes unreliable
  - *Postpruning* - take a fully-grown decision tree and discard unreliable parts
- Postpruning preferred in practice—prepruning can "stop too early"

Prepruning

- Based on statistical significance test
  - Stop growing the tree when there is no *statistically significant* association between any attribute and the class at a particular node
- Most popular test: *chi-squared test*
- ID3 used chi-squared test in addition to information gain
  - Only statistically significant attributes were allowed to be selected by information gain procedure

Post-pruning

- First, build full tree
- Then, prune it
  - Fully-grown tree shows all attribute interactions
- Problem: some subtrees might be due to chance effects
- Two pruning operations:
  1. *Subtree replacement*
  2. *Subtree raising*
- Possible strategies:
  - error estimation
  - significance testing

The disadvantages of decision trees include:

- Disadvantage number 2:
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR problems.

	a	b	class
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

- Classic example: XOR/Parity-problem
  - No individual attribute exhibits any significant association to the class

  → In a dataset that contains XOR attributes a and b, and several irrelevant (e.g., random) attributes, ID3 can not distinguish between relevant and irrelevant attributes

  → Prepruning won't expand the root node
  - Structure is only visible in fully expanded tree
- But:
  - XOR-type problems rare in practice
  - prepruning is faster than postpruning

The disadvantages of decision trees include:

- Disadvantage number 3:
- Decision tree learners **create biased trees if some classes dominate**. It is therefore recommended to **balance** the dataset prior to fitting with the decision tree.
- We will use the imblearn library to balance our **original** datasets, using **sub** or **over** sampling.

ID	Class	BinaryClass	Account Balance	SavingsAccount
1	GoodCostumer	0	890	Yes
2	GoodCostumer	0	945	Yes
3	GoodCostumer	0	912	Yes
4	AverageCostumer	0	740	Yes
5	AverageCostumer	0	780	Yes
6	AverageCostumer	0	810	No
7	AverageCostumer	0	692	No
8	AverageCostumer	0	570	No
9	Regular Costumer	0	549	Yes
10	Regular Costumer	0	540	Yes
11	Regular Costumer	0	498	Yes
12	Regular Costumer	0	501	Yes
13	BadCostumer	1	420	No
14	BadCostumer	1	581	Yes
15	BadCostumer	1	564	No

Original dataset

ID	Class	BinaryClass	Account Balance	SavingsAccount
3	GoodCostumer	0	912	Yes
6	AverageCostumer	0	810	No
12	Regular Costumer	0	501	Yes
13	BadCostumer	1	450	Yes
14	BadCostumer	1	581	No
15	BadCostumer	1	564	Yes

Balanced with sub sampling

ID	Class	BinaryClass	Account Balance	SavingsAccount
1	GoodCostumer	0	890	Yes
2	GoodCostumer	0	945	Yes
3	GoodCostumer	0	912	Yes
4	AverageCostumer	0	740	Yes
5	AverageCostumer	0	780	Yes
6	AverageCostumer	0	810	No
7	AverageCostumer	0	692	No
8	AverageCostumer	0	570	No
9	Regular Costumer	0	549	Yes
10	Regular Costumer	0	540	Yes
11	Regular Costumer	0	498	Yes
12	Regular Costumer	0	501	Yes
13	BadCostumer	1	420	No
14	BadCostumer	1	581	Yes
15	BadCostumer	1	564	No
F16	BadCostumer	1	450	Yes
F17	BadCostumer	1	581	No
F18	BadCostumer	1	564	Yes
F19	BadCostumer	1	450	Yes
F20	BadCostumer	1	581	No
F21	BadCostumer	1	564	Yes
F22	BadCostumer	1	450	Yes
F23	BadCostumer	1	581	No
F24	BadCostumer	1	564	Yes

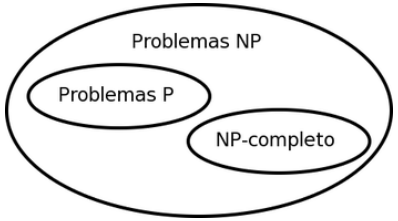
Balanced with over sampling

The disadvantages of decision trees include:

- Disadvantage number 4:
- When dealing with **a small file, the subdivision of data into training and test samples is penalizing**.
- With a small training set, we will have less instances to build an effective model, and the estimate of the error will be unreliable because is based on too few observations.
- The solution is to use **cross-validation:**
- **Cross-validation strategy consist of (1) learning the classifier or regression model using the whole dataset. (2) evaluate the performance of this model using any of the cross-validation mechanisms available in sklearn.**
  - K-folds: divides all the samples in k groups of samples, called folds (if k = n, this is equivalent to the Leave One Out strategy), of equal sizes (if possible). The prediction function is learned using k - 1 folds, and the fold left out is used for test.
  - Repeated k-folds: repeats K-Fold n times. It can be used when one requires to run KFold n times, producing different splits in each repetition.
  - Leave One Out (LOO): k folds with k=n. LOO is more computationally expensive than k-fold cross validation.
  - Leave P Out (LPO): is very similar to LeaveOneOut as it creates all the possible training/test sets by removing p samples from the complete set.
  - Shuffle & Split: will generate a user defined number of independent train / test dataset splits. Samples are first shuffled and then split into a pair of train and test sets.
- Cross-validation **could/should be** done preserving the original stratification and grouping of the data (very important in panel and/or time series data!!)
- *See more here*

The disadvantages of decision trees include:

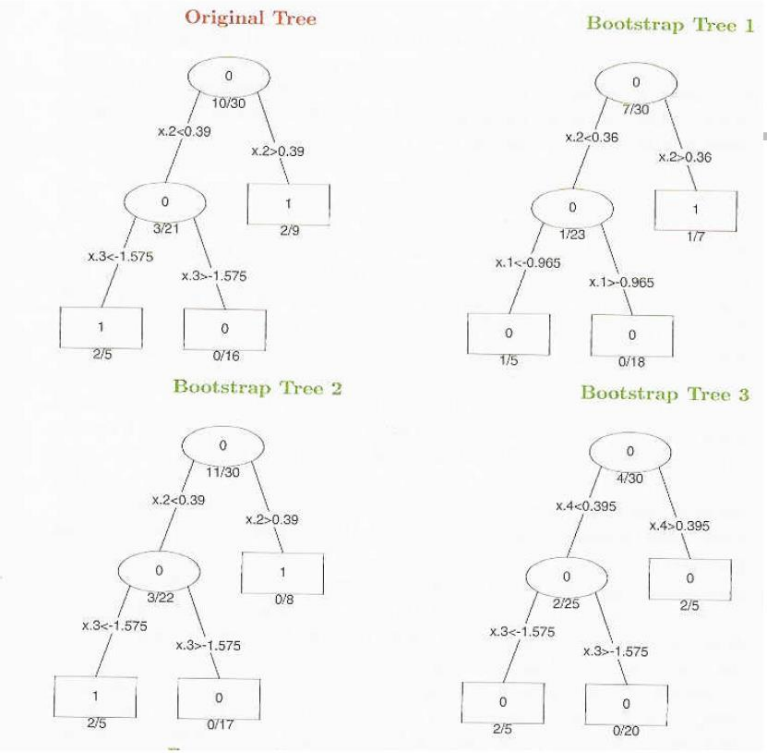
- Disadvantage number 5:
- The problem of learning an optimal decision tree is **known to be NP-complete**.
- In computer science, all problems that could in theory be verified and solved using an algorithm are usually classified according to then time (or resources) it will take the computer to verify and find a solution.



- NP-Complete problems are the hardest ones, and current implementations run in exponential time. P problems are solved in polynomial time. Some NP problems may be solved in polynomial time? Win 1 million dollars, is P=NP?
- Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm **where locally optimal decisions are made** at each node.
- Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an **ensemble** learner, where the features and samples are randomly sampled with replacement.

The disadvantages of decision trees include:

- Disadvantage number 6:
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated.
  - This problem is mitigated by using decision trees within an **ensemble**.



Ensemble Methods

- Basic Idea
- Bagging
- Boosting
- Stacking

Ensemble Classifiers

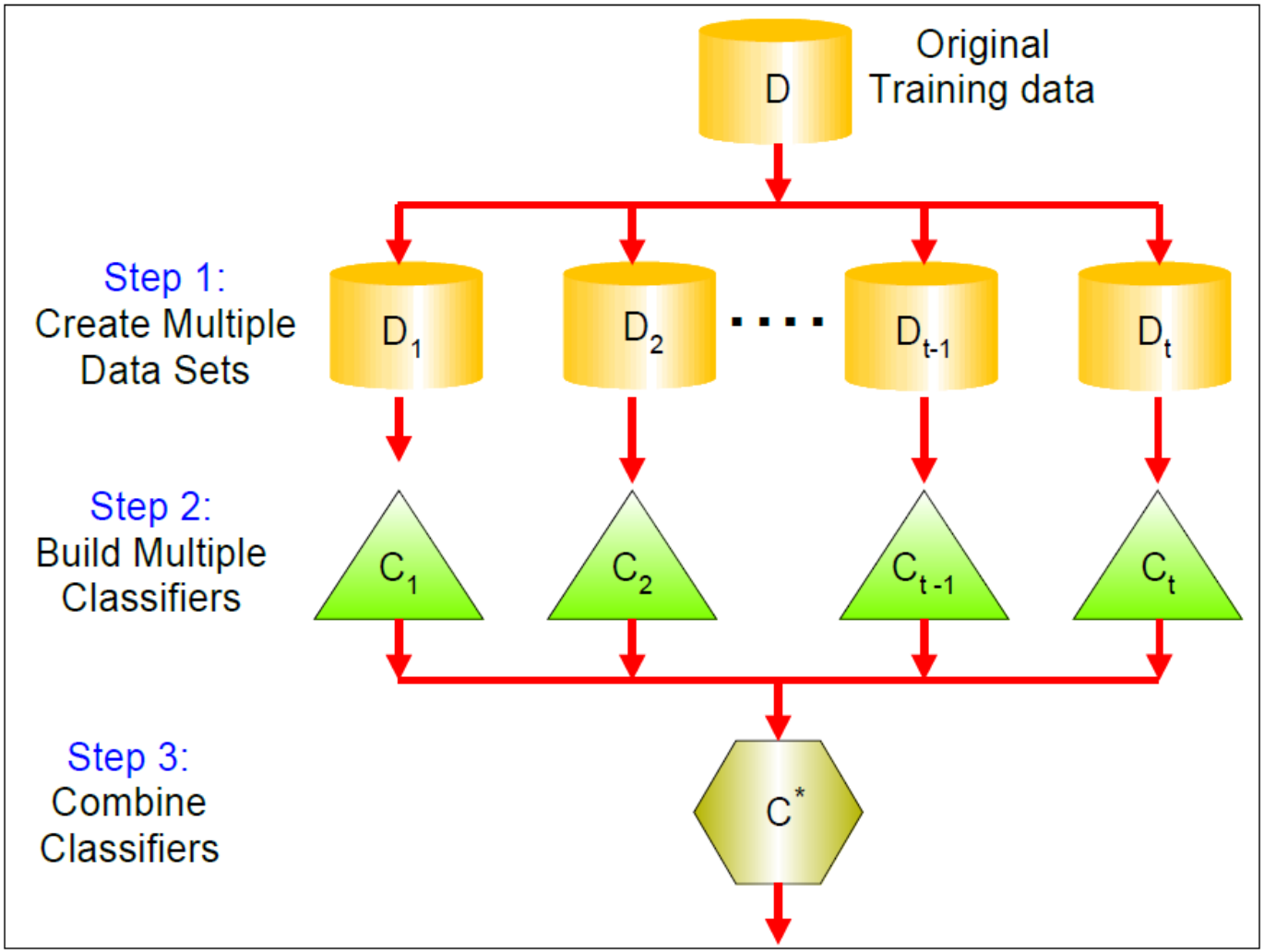
- IDEA:
  - do not learn a *single* classifier but learn a *set of classifiers*
  - *combine the predictions* of multiple classifiers
- MOTIVATION:
  - reduce variance: results are less dependent on peculiarities of a single training set
  - reduce bias: a combination of multiple classifiers may learn a more expressive concept class than a single classifier
- KEY STEP:
  - formation of an ensemble of *diverse* classifiers from a single training set

Why do ensembles work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume classifiers are independent
    - i.e., probability that a classifier makes a mistake does not depend on whether other classifiers made a mistake
    - **Note:** in practice they are not independent!
- Probability that the ensemble classifier makes a wrong prediction
  - The ensemble makes a wrong prediction if the majority of the classifiers makes a wrong prediction
  - The probability that 13 or more classifiers err is

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} \approx 0.06 \ll \varepsilon$$

Bagging: General Idea



Bagging variations

- When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as **Pasting** [R154]
- If samples are drawn with replacement, then the method is known as **Bagging** [R155].
- When random subsets of the dataset are drawn as random subsets of the features, then the method is known as **Random Subspaces** [R156].
- When base estimators are built on subsets of both samples and features, then the method is known as **Random Patches** [R157].
- There are two well known implementations:
  - Random Forests
  - Extremely Randomized Trees

Bagging variations – Random Forests

- In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.
- In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features.
- Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.
- The scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.

Bagging variations – Extremely Randomized Trees

- In extremely randomized trees, randomness goes one step further in the way splits are computed.
- As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule.
- This usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias

Boosting

- Basic Idea:
  - later classifiers focus on examples that were misclassified by earlier classifiers
  - weight the predictions of the classifiers with their error
- Realization
  - perform multiple iterations
    - each time using different example weights
  - weight update between iterations
    - increase the weight of incorrectly classified examples
    - this ensures that they will become more important in the next iterations
      (misclassification errors for these examples count more heavily)
  - combine results of all iterations
    - weighted by their respective error measures

Boosting - AdaBoost

- The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data.
- The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction.
- As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence.

Boosting – Algorithm AdaBoost

1. initialize example weights $w_i = 1/N \quad (i = 1..N)$
2. for $m = 1$ to M // M ... number of iterations
   a) learn a classifier C_m using the current example weights
   b) compute a weighted error estimate $err_m = \frac{\sum w_i \, of \, all \, incorrectly \, classified \, e_i}{\sum_{i=1}^{N} w_i}$ ← $= 1$ because weights are normalized
   c) compute a classifier weight $\alpha_m = \frac{1}{2} \log\left(\frac{1 - err_m}{err_m}\right)$
   d) for all correctly classified examples e_i : $w_i \leftarrow w_i e^{-\alpha_m}$
   e) for all incorrectly classified examples e_i : $w_i \leftarrow w_i e^{\alpha_m}$

   update weights so that sum of correctly classified examples equals sum of incorrectly classified examples

   f) normalize the weights w_i so that they sum to 1
3. for each test example
   a) try all classifiers C_m
   b) predict the class that receives the highest sum of weights α_m

Combining Predictions

- voting
  - each ensemble member votes for one of the classes
  - predict the class with the highest number of vote (e.g., bagging)
- weighted voting
  - make a *weighted* sum of the votes of the ensemble members
  - weights typically depend
    - on the classifiers confidence in its prediction (e.g., the estimated probability of the predicted class)
    - on error estimates of the classifier (e.g., boosting)
- stacking
  - Why not use a classifier for making the final decision?
  - training material are the class labels of the training data and the (cross-validated) predictions of the ensemble members

Stacking

- Basic Idea:
  - learn a function that combines the predictions of the individual classifiers
- Algorithm:
  - train n different classifiers $C_1 \ldots C_n$ (the *base classifiers*)
  - obtain predictions of the classifiers for the training examples
    - better do this with a cross-validation!
  - form a new data set (the *meta data*)
    - **classes**
      - the same as the original dataset
    - **attributes**
      - one attribute for each base classifier
      - value is the prediction of this classifier on the example
  - train a separate classifier M (the *meta classifier*)

Voting Classifier

- Stacking is implemented in sklearn as VotingClassifier with the following variations:

- **Majority Class Labels (Majority/Hard Voting)**
- **Weighted Average Probabilities (Soft Voting)**

Applied Machine Learning for Business and Economics

AMLBE

Profesor : David Díaz S.