Data Science

Marc Tommasi

November 7, 2023

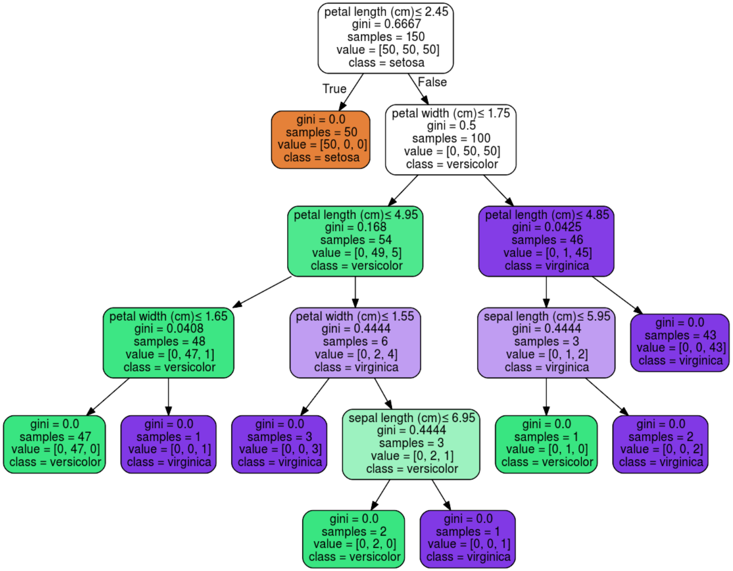
Outline

Outline

Principles

- Classifier: a function from $\mathcal{X}$ to $\mathcal{Y}$
- Model in the form of a set of successive decision rules, represented in a tree
- Model easy to interpret when the tree is small
- Example on iris dataset, 3 classes: setosa, virginica, versicolor; 150 examples; attributes sepal length, sepal width, petal length, petal width.

Example


petal length (cm)≤ 2.45
gini = 0.6667
samples = 150
value = [50, 50, 50]
class = setosa
True
False
gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa
petal width (cm)≤ 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor
petal length (cm)≤ 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor
petal length (cm)≤ 4.85
gini = 0.0425
samples = 46
value = [0, 1, 45]
class = virginica
petal width (cm)≤ 1.65
gini = 0.0408
samples = 48
value = [0, 47, 1]
class = versicolor
petal width (cm)≤ 1.55
gini = 0.4444
samples = 6
value = [0, 2, 4]
class = virginica
sepal length (cm)≤ 5.95
gini = 0.4444
samples = 3
value = [0, 1, 2]
class = virginica
gini = 0.0
samples = 43
value = [0, 0, 43]
class = virginica
gini = 0.0
samples = 47
value = [0, 47, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
sepal length (cm)≤ 6.95
gini = 0.4444
samples = 3
value = [0, 2, 1]
class = versicolor
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica

Inference

- Start from the root, with an example (e.g. 3, 2, 3, 2)
- We go through the tests, each test cutting the space into 2 parts.
- We thus designate a recursive partitioning of the description space
- Each leaf gives a label to a part.
- Good visualization in Jake VandenPlas's book.

Algorithms

- This method introduces two biases: choice of function class + algorithmic bias.
- The algorithmic bias stems from a greedy heuristic:
  - the tree is built from root to leaves,
  - at each stage, a node corresponding to part of the data is developed
  - the best test is selected according to a gain criterion
  - this choice is no longer questioned
- Several algorithms exist: ID3, C4.5, C5, CART, etc.
- Sklearn implements CART

Explanations with ID3

- We consider binary attributes $A = \{0, 1, \ldots, p\}$

```
def id3(S,A):
  """ S: dataset, A: set of attributes """
  if all examples in S have the same class or A is empty:
    retrun a leaf labelled with the majority class

  Let j be the attribute that maximizes the gain
    t_l = id3({(x,y) in S st x_j=0}, A\{j})
    t_r = id3({(x,y) in S st x_j=1}, A\{j})
    return the tree whose root node is the test x_j=1
        and the left children is t_l (case False)
and the right children is t_r (case True).
```

Compute the gain

- Gain: difference observed between absence and presence of the test in the tree.
- Notation: Let $\mathbb{P}_S[F]$ be the probability of F when samples are drawn iid in S
  - (if a node with test x_i corresponds to m examples and m_l go to the left, m_r go to the right, then $\mathbb{P}_S[x_i = 1] = m_r/m$)
- the gain is computed with a function C to be defined:

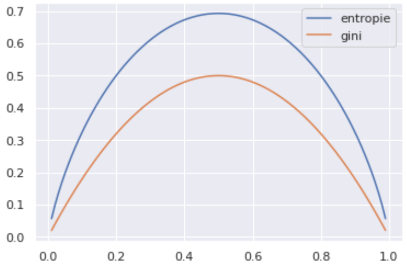
$$\mathrm{Gain}(S, i) = C(\mathbb{P}_S[y = 1]) - \big(\mathbb{P}_S[x_i = 1]C(\mathbb{P}_S[y = 1 \mid x_i = 1]) + \mathbb{P}_S[x_i = 0]C(\mathbb{P}_S[y = 1 \mid x_i = 0])\big)$$

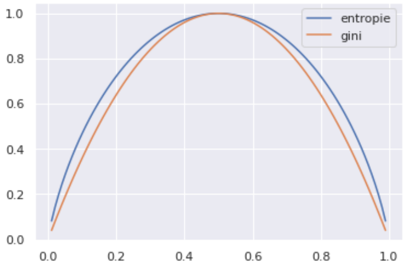
$C(a)$ can be:

- the training error: $C(a) = \min(a, 1 - a)$
- the information gain (entropy): $C(a) = -a \log(a) - (1 - a) \log(1 - a)$
- the Gini score: $C(a) = 2a(1 - a)$

Entropy and Gini

- Will favor tests that achieve the best separation, ie. when $\mathbb{P}_S[y \mid x_i]$ is close to 0 or 1.





- $-a\log(a) - (1-a)\log(1-a)$ et $2a(1-a)$
- $-a\log_2(a) - (1-a)\log_2(1-a)$ et $4a(1-a)$

Limiting overfitting

- Bias/complexity trade-off: the greater the depth, the more complex the function class, the lower the bias but the greater the tendency of decision trees to overfitting.
- depth bound (as in sklearn, parameter `max_depth`)
- pruning consists of removing branches: replacing a node with a class label.
  - Bottom-up approach with a statistical test (ξ^2 or error evaluation).

The case of continuous attributes

- discretization considers all possible thresholds observed on the training sample
- the calculation for these m possible tests could be $O(dm^2)$ but can be reduced to $O(dm\log(m))$.

Regression Trees

- the examples arriving in a leaf are averaged to determine the value to be predicted
- the cost function used to build the tree is, for example, the MSE

Advantages and disadvantages

- model readability
- In theory, high algorithmic complexity to find the best tree, but fast heuristic approach in practice.
- difficulty in setting the max depth or other criteria to avoid overfitting

Outline

Main measures

- Taux d'erreurs (accuracy) : the ratio of the number of wrongly classified examples and the total number of examples.
- Confusion Matrix : number of examples according to prediction and true class.

	Positifs	Negatifs
Prediction Positive	TP	FP
Prediction Negative	FN	TN

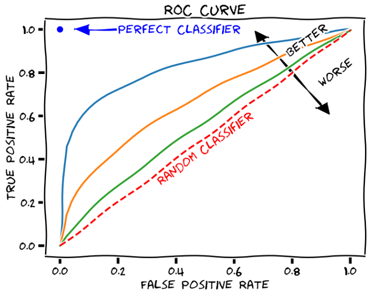
- Precision ($\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$) and Recall ($\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$)
- The harmonic mean between precision and recall is the F1 score and gives a unique score. ($2 \times \frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision}+\mathrm{Recall}}$)
- Other important measures are the True Positive Rate (TPR) and the False Positive Rate (FPR) ($\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$ et $\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$)
- Read the Wikipedia page !!!

Decision function

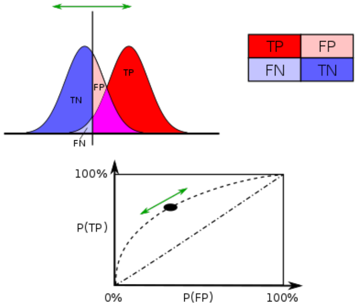
- To obtain predictions, a score is calculated and then the score is compared to a threshold.
- The threshold can be changed and seen as a hyperparameter

ROC curves

- La courbe ROC (Receiver operating characteristic) trace FPR et TPR dans un diagramme en fonction du seuil.



- When the output of the classifier is considered as a random variable



- The area under the ROC curve can be interpreted as the probability of the classifier giving a higher score to a positive example than to a negative example.

In the case of multiclass classification

- The measures also work in a multi-class context, with certain restrictions.
- FPR, TPR, etc are calculated per class
- Averages are calculated across classes
- The average is not a good indicator
- In real problems, macro or micro average techniques can be used to correct the bias introduced by the mean
- For example
  - Class A: 1 TP and 1 FP; Class B: 10 TP and 90 FP; Class C: 1 TP and 1 FP; Class D: 1 TP and 1 FP.
  - macro-average: $(0.5 + 0.1 + 0.5 + 0.5)/4 = 0.4$
  - micro-average: $(1 + 10 + 1 + 1)/(2 + 100 + 2 + 2) = 0.123$
- We can also weight the averages by the proportions in each class.

With sklearn

- Functions are listed in `sklearn.metrics`
  - `confusion_matrix`
  - `precision_score`, `recall_score` et `f1_score`,
  - `classification_report` : un tableau avec les principales mesures
- Most of the classifiers have a method `decision_function` or `predict_proba`
- You can draw curves with `precision_recall_curve` and `roc_curve` and the confusion matrix with `plot_confusion_matrix`, `plot_precision_recall_curve`, `plot_roc_curve`

Outline

Principle

- (weighted) vote of a set of binders: the majority vote wins the decision
- theoretical results given by boosting.
- Intuition: If the basic classifiers are sufficiently independent to be less wrong than a random classifier, then their vote forms a more accurate method than the basic classifiers.
- Numerous methods can be used to combine these classifiers: Bagging, Pasting, Boosting, Stacking, etc.

Bagging

- You can combine classifiers of a different nature or the same classifiers obtained on different data (but from the same distribution).
- The aim is to reduce variance and limit overfitting
- bagging: uniform random selection of a subset of the data (boostrap aggregating);
- pasting: same as above, but drawn without replacement.
- random subspace: similar to boostrap but drawn on attributes and not on examples.
- random patches: when you draw on both examples and attributes.
- Aggregation: by mode (highest frequency of predictions), by majority vote, by the average of decision functions.
- Scaling is easier if you can distribute the calculations

Random Forests

- It's basically bagging with decision trees
- Introducing variety into the construction of a set of decision trees
- The randomness is obtained by
  - drawing a sub-sample of examples, or
  - in the search for the best test: randomly mixed attributes, chosen from a subset of the attributes
- The readability of decision trees is lost
- However, the importance of attributes can be calculated as the average gain over all the trees in the forest, and this importance can be visualised.

Boosting

- modify the distribution of examples by using a weight on each example that varies according to the rate of correct classification
- the weight of poorly classified examples increases gradually
- the decision is a weighted vote. The weight is a function of the accuracy of the classifier

Boosting principles

- Reminder: error in generalisation for ERM $=$ approximation error $+$ estimation error
- We have a class of poor functions with a large approximation error: weak learners whose performance is just slightly better than random.
- How can we iteratively make it richer?
- Boost we aggregate weak learners to form stronger learners, whose performance can be as large as we want.
- Objectives:
  - a method that breaks down the complexity of learning a strong learner directly
  - to answer the theoretical question of how to transform a weak learner into a strong learner.
- Adaboost is an implementation that solves this theoretical problem.

Weak learner

Definition (Weak learner)

A is a γ-weak-learner for a class $\mathcal{H}$ if there exists a function $m_{\mathcal{H}} : (0, 1) \to \mathbb{N}$ such that for any $\delta \in (0, 1)$, for every distribution $\mathcal{D}$ over $\mathcal{X}$ and for every labeling function $f : \mathcal{X} \to \{-1, 1\}$, if the realization assumption holds with respect to $\mathcal{H}$, $\mathcal{D}$, f, then when running the learning algorithm A on $m \geq m_{\mathcal{H}}(\delta)$ iid examples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis h such that, with probability of at least $1 - \delta$, $L_{\mathcal{D},f}(h) \leq 1/2 - \gamma$.

- ϵ (strong learner) is replaced by $1/2 - \gamma$.

Adaboost

- we are given a weak learner
- Adaboost is an iterative algorithm that modifies the distribution $\boldsymbol{D}^{(t)}$ over S at each round t
- the error of the hypothesis h_t built by the weak learner is $\epsilon_t = \sum_i^m D_i^{(t)} \mathbb{1}_{[h_t(\boldsymbol{x}_i) \neq y_i]}$ with probability $1/2 - \gamma$.
- The weight associated with h_t is inversely proportional to the error: $w_t = \frac{1}{2}\log(\frac{1}{\epsilon} - 1)$: wrongly classified examples are given a higher weight.
- the output of Adaboost is a weighted vote of all hypotheses h_t

Algorithm

```
$\mathcal{D}^{(1)} = (1/m, \ldots, 1/m)$
for  $t = 1, \ldots, T$
    $h_t = WL(\mathcal{D}^{(t)}, S)$
    $\epsilon_t = \sum_i^m D_i^{(t)} \mathbb{1}_{[h_t(\boldsymbol{x}_i) \neq y_i]}$
    $w_t = \frac{1}{2} \log(\frac{1}{\epsilon_t} - 1)$
    $D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(\boldsymbol{x}_i))}{\sum_j^m D_j^{(t)} \exp(-w_t y_j h_t(\boldsymbol{x}_j))}$
return  $h_s(\boldsymbol{x}) = \mathrm{sign}\big(\sum_{t=1}^T w_t h_t(\boldsymbol{x})\big)$
```

- The hypothesis class of Adaboost, given a weak learner that builds hypothesis in B is $\{\mathrm{sign}\big(\sum_{t=1}^T w_t h_t(x)\big) \mid \boldsymbol{w} \in \mathbb{R}^T, h_t \in B\}$
- Theorem : The training error of Adaboost is bounded by $exp(-2\gamma^2 T)$

Gradient boosting

- Another way of focusing on errors: the next classifier in the loop attempts to correctly predict residual errors.
- Example with `DecisionTreeRegressor`.
  - An instance `dtr` of this classifier produces a residual `y2 =y - dtr.predict(X)`.
  - A `dtr2` is constructed to predict `y2`.
  - Combine the sum of the predictions from the 2 classifiers.
- The `GradientBoostingRegressor` class does this.

Stacking

- Bias reduction by classifier compositions
- we learn to combine the predictions of basic classifiers
- learning is divided into two parts:
  - part A trains n base classifiers
  - part B is used to create a set P of prediction vectors of dimension n
- the set P is used to train a combination (*blender*)

With sklearn

- a `VotingClassifier` class for simple weighted voting or no voting of classifiers
- the `AdaBoostClassifier` class
- the `BaggingClassifier` class
  - Bagging/pasting setting via the `boostrap` parameter
  - Setting of `max_features` and `max_samples`.
- the `RandomForestClassifier` class
- the `StackingClassifier` class
- In these implementations, the aggregation is the average of the decision functions.
- The same classes exist for regression.