Machine Learning and Programming in Python

Lecture for Master and PhD students

Chair of Data Science in Economics

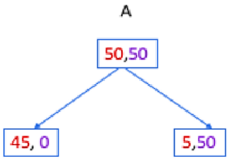
Ruhr University Bochum

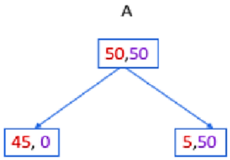
Summer semester 2024

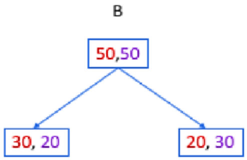
Lecture 9

Classification Trees

Examples







- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, the residual sum of squares (RSS) cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the classification error rate. This is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - max_k(p_{mk})$$

- p_{mk} represents the proportion of training observations in the mth region that are from the kth class.
- The classification error is the fraction of training observations that do not belong to the modal class
- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

- The Gini index is defined by

  $$G = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) = 1 - \sum_{k=1}^{K} p_{mk}^2$$

  a measure of total variance across the K classes. The Gini index takes on a small value if all of the p_{mk}'s are close to zero or one.
- For this reason the Gini index is referred to as a measure of node purity - a small value indicates that a node contains predominantly observations from a single class.

- An alternative to the Gini index is cross-entropy, given by

  $D = -\sum_{k=1}^{K} p_{mk} log_2(p_{mk})$

- It turns out that the Gini index and the cross-entropy are very similar numerically
- a small value indicates that a node contains predominantly observations from a single class.

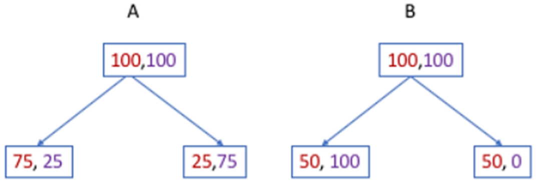
Splitting criteria for classification trees

- Split to extract the most information at that point.
- Information gain from splitting a parent node (R_m) with n_p training observations into two child nodes (R_1 and R_2 with $n_1 + n_2 = n_p$):

$$IG(R_m) = I(R_m) - [\frac{n_1}{n_p} I(R_1) + \frac{n_2}{n_p} I(R_2)]$$

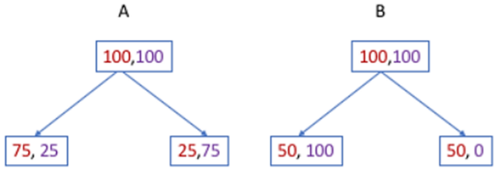
- $I(R)$ is a measure of 'impurity' - how mixed up are the classes in region R?
- Three commonly used measures of impurity:
  - Classification error
  - **Gini Index**
  - **Entropy**

Example of information gain with classification error



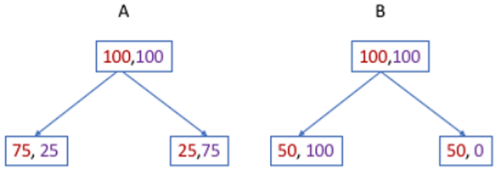
- E of parent node: $E = 1 - 0.5 = 0.5$
- In split A:
  - $E_{left} = 1 - 0.75 = 0.25$ and $E_{right} = 1 - 0.75 = 0.25$
  - $IG_{E,A} = 0.5 - (\frac{1}{2}0.25 + \frac{1}{2}0.25) = 0.25$
- In split B:
  - $E_{left} = 1 - \frac{2}{3} = \frac{1}{3}$ and $E_{right} = 1 - 1 = 0$
  - $IG_{E,B} = 0.5 - (\frac{3}{4}\frac{1}{3} + \frac{1}{4}0) = 0.25$
- Same IG using classification error

Example for information gain with Gini



- G of parent node: $G = 1 - (0.5^2 + 0.5^2) = 0.5$
- In split A:
  - $G_{left} = 1 - (0.75^2 + 0.25^2) = 0.375$ and $G_{right} = 1 - (0.25^2 + 0.75^2) = 0.375$
  - $IG_{G,A} = 0.5 - (\frac{1}{2}0.375 + \frac{1}{2}0.375) = 0.125$
- In split B:
  - $G_{left} = 1 - (\frac{1}{3}^2 + \frac{2}{3}^2) = \frac{4}{9}$ and $G_{right} = 1 - 1^2 = 0$
  - $IG_{G,B} = 0.5 - (\frac{3}{4}\frac{4}{9} + \frac{1}{4}0) = 0.167$
- Gini measure favours split B.

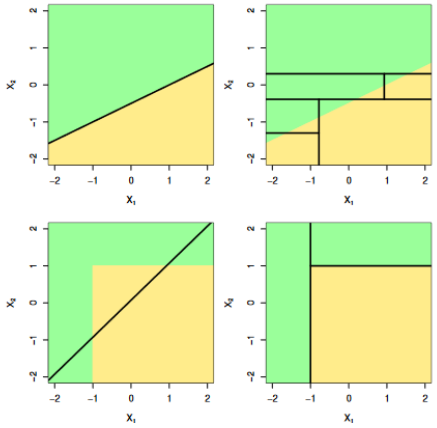
Example for information gain with Entropy



- Use $\log_2()$ and $0 * \log 0 = 0$
- H of parent node: $H = -0.5\log_2 0.5 - 0.5\log_2 0.5 = 1$
- In split A (check this):
    - $H_{left} = 0.81$ and $H_{right} = 0.81$
    - $IG_{H,A} = 0.19$
- In split B:
    - $H_{left} = 0.92$ and $H_{right} = 0$
    - $IG_{H,B} = 0.31$
- Entropy measure favours split B.

Whether to take decision trees or linear regression depends on the underlying functional form between the X variables

Top panels: mostly linear, linear regression better
Bottom panels: mostly non-linear decision trees better



Bagging

- Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- Decision trees suffer from high variance. Random halves of the same training data can yield different trees.
- Recall that given a set of n independent observations $Z_1, ..., Z_n$, each with variance σ_2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.
- In other words, averaging a set of observations reduces variance. Of course, this is not practical because we generally do not have access to multiple training sets

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets.
- Build a deep tree for each sample to get low bias
- Average across trees to lower variance

- This is called bagging

- For regression trees: for each test observation, we record the mean predicted by each of the B trees, and take the mean of them
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions.

Random forests

- Bagging may suffer from correlated trees
- If one strong predictor, most or all trees will split it first
- Resulting trees might be similar
- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees.

- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose $m = \sqrt{p}$, that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors

Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees

- Grow and combine a <u>sequence</u> of decision trees.
- Each subsequent tree uses information from previous trees.
  1. Grow a small decision tree (shallow depth).
  2. Compute the residuals.
  3. Grow another small tree to fit the residuals.
  4. Update the model by adding the new tree.
  5. Repeat steps 2 to 4.
  6. Algorithm on next slide.

1. Pick number of trees, B, interaction depth d ($d+1$ terminal nodes), and shrinkage parameter λ.
2. Set up model $f(\hat{x}) = 0 \implies r_i = y_i, \forall i$ in training data.
3. For $b = 1, 2, \ldots B$ repeat:
    1. Fit a tree of depth d, $\hat{f^b}$ to the training data (X, r).
    2. Update model by adding a shrunken version of the new tree, $\hat{f} \leftarrow \hat{f} + \lambda \hat{f^b}$
    3. Update residuals, $r_i \leftarrow r_i - \lambda \hat{f^b}$
4. Final model $f(\hat{x}) = \sum_{b=1}^{B} \lambda f^b(\hat{x})$

What is the idea behind this procedure?

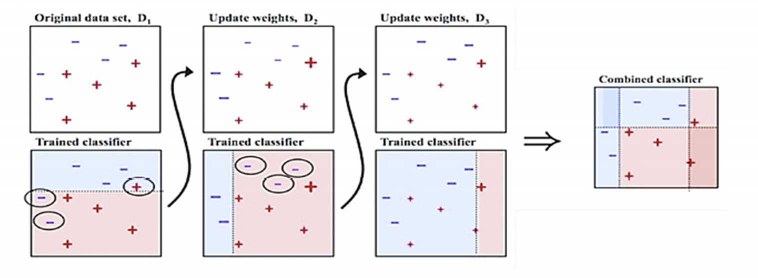
- Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals

Tuning parameters for boosting

- The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
- The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
- The number of splits d in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

AdaBoost Classifier

- Original idea due to Robert E. Schapire in 1990.
- Popular boosting algorithm; often yields good results.
- Uses the complete training dataset in each step.
- The weights on training data points updated each step with more weight on those with larger prediction errors.



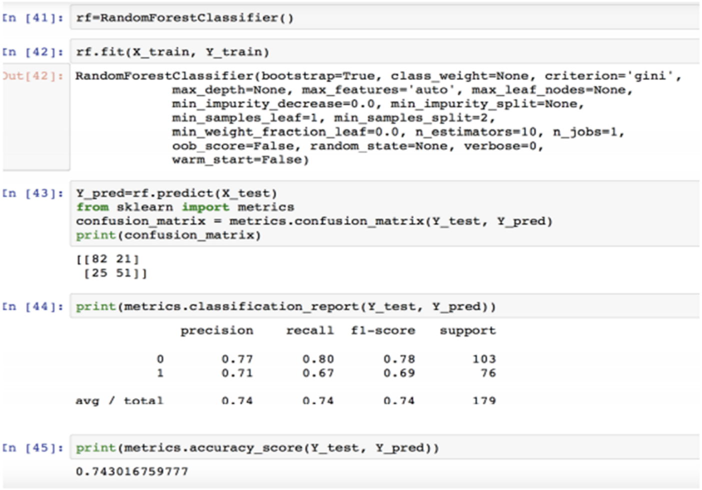
Source: *Valentina Alto. Understanding AdaBoost for Decision Trees.* From `https://towardsdatascience.com/`

In Python

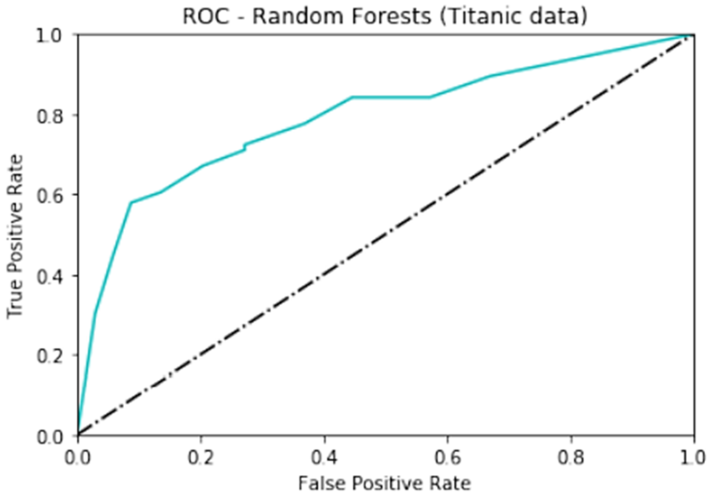
- Scikit-learn has tools for decision trees, random forests, boosting algorithms
- Single decision tree: from sklearn.tree import DecisionTreeClassifier
- Random forests: from sklearn.ensemble import RandomForestClassifier
- AdaBoost: from sklearn.ensemble import AdaBoostClassifier

Many **parameters**, e.g for random forests:

- Number of trees in the forest (*n_estimators*, current default is 100)
- Measure of split quality (criterion, default is ‘Gini’)
- Depth of the tree (*max_depth*, default is build tree until leaves are pure or have less than min samples split data points)
- Minimum sample size to split an internal node (*min_samples_split*, default is 2)
- Minimum data points in a leaf (*min_samples_leaf* , default is 1)
- And many others...



```
In [41]: rf=RandomForestClassifier()

In [42]: rf.fit(X_train, Y_train)

Out[42]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)

In [43]: Y_pred=rf.predict(X_test)
         from sklearn import metrics
         confusion_matrix = metrics.confusion_matrix(Y_test, Y_pred)
         print(confusion_matrix)

         [[82 21]
          [25 51]]

In [44]: print(metrics.classification_report(Y_test, Y_pred))

                      precision    recall  f1-score   support

                   0       0.77      0.80      0.78       103
                   1       0.71      0.67      0.69        76

         avg / total       0.74      0.74      0.74       179

In [45]: print(metrics.accuracy_score(Y_test, Y_pred))

         0.743016759777
```


ROC - Random Forests (Titanic data)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0