

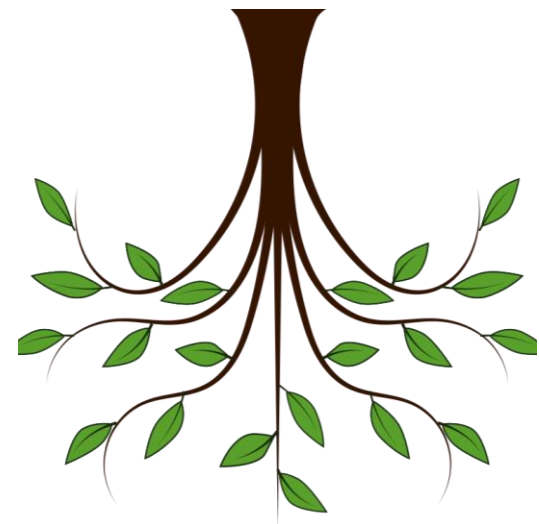
Tree and Ensemble Methods

Making decisions... better

Yordan Darakchiev

Technical Trainer

iordan93@gmail.com





sli.do

#MachineLearning

Table of Contents

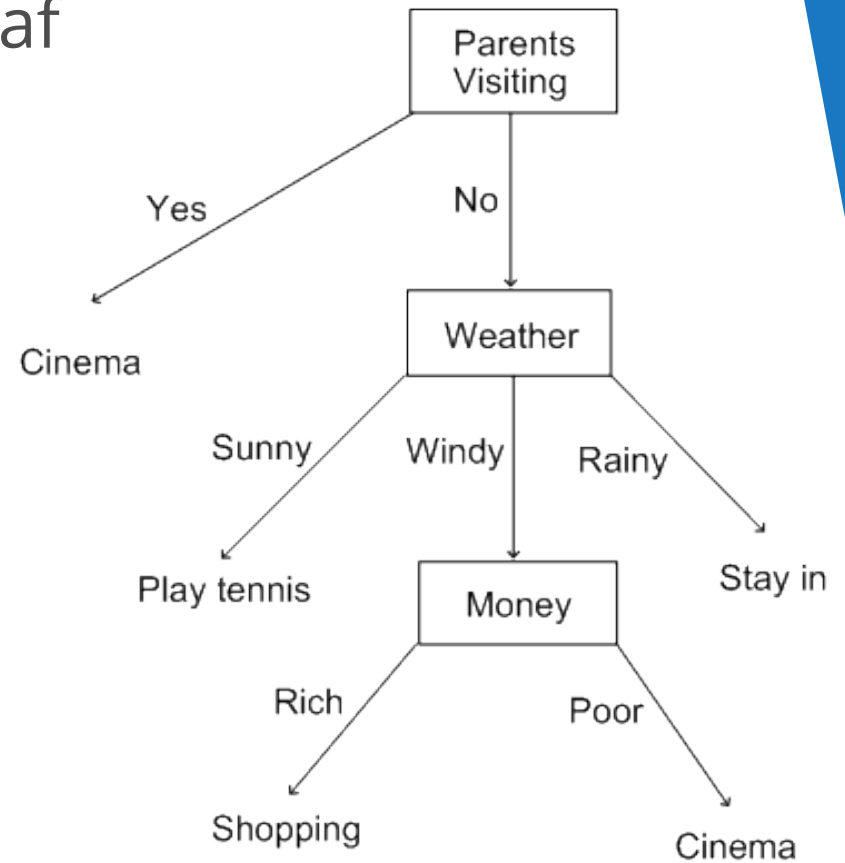
- Decision trees
- Ensemble algorithms
 - Random decision forests
 - AdaBoost

Decision Trees

"To be or not to be..."

Decision Trees

- Can be used for classification or regression
 - **Root**: top node (always a single root)
 - **Leaves**: bottom nodes
 - Getting an answer: path from root to a leaf
- Biggest advantage: easy to interpret
- Answer a series of yes / no questions to get the data model
 - Similar to the way we decide what to do
- We can construct our own decision trees using if-statements
 - Machine learning problem: construct the tree without involving "brain power"

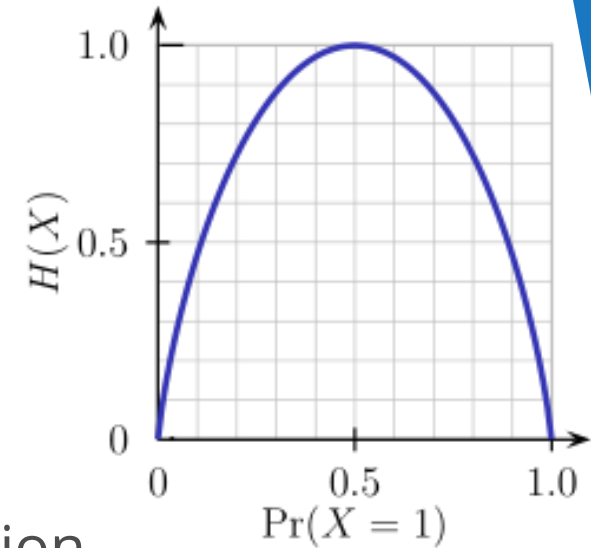


Decision Trees (2)

- Start at the root
- At each step decide how to split the data
 - Choose the feature (column) that results in the largest **information gain** (IG) ([example](#))
- Iterate until every leaf node contains only one class
 - To avoid overfitting \Rightarrow **pruning** (limiting the max depth)
- Objective function: maximize IG:
$$IG(D_p, f) = I(D_p) - \sum_{j=1}^m \frac{N_j}{N_p} I(D_j)$$
 - f – feature to perform the split on
 - D_p, D_j – datasets of the parent and child nodes
 - N_p, N_j – number of samples (at parent / child nodes)
 - I – **impurity measure**
 - More simply, difference between parent and child impurities
 - Greater difference = more IG

Impurity Measures

- Most libraries implement binary decision trees
 - Each node can have 0, 1 or 2 children
 - Reasons: simplicity, reducing the search space
- Three common impurity measures
 - Entropy – measure of classification uncertainty
 - Probability 0 or 1 = no uncertainty
 - Probability 0,5 = max uncertainty
 - Gini index – similar to entropy
 - Criterion to minimize the probability of misclassification
 - We usually use one of the measures, as they provide similar results
 - Misclassification error
 - Linear measure (0 at $p = \{0; 1\}$, max at $p = 0,5$)
 - **Good for pruning** a tree but worst measure for growing

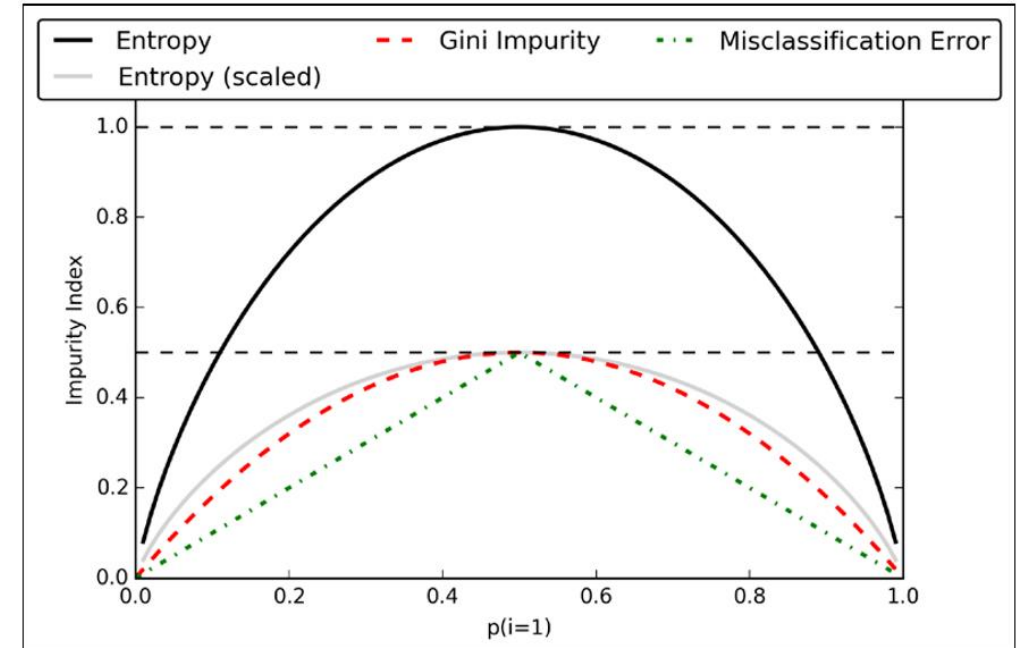


Visualizing Impurity Measures

- For a two-class classifier, visualize the measures
 - Parameter p – probability of class 1 ($0 \leq p \leq 1$)

```
def entropy(p):  
    return -p * np.log2(p) - (1 - p) * np.log2((1 - p))  
def gini_index(p):  
    return p * (1 - p) + (1 - p) * (1 - (1 - p))  
def misclassification_error(p):  
    return 1 - np.max([p, 1 - p])
```

- Scaled entropy:
entropy / number of classes
- Observations
 - Min: $p = \{0, 1\}$, max: $p = 0,5$
 - Entropy and Gini are very similar



Decision Trees in scikit-learn

- Creating and fitting a classifier – as usual

```
from sklearn.tree import DecisionTreeClassifier  
decision_tree = DecisionTreeClassifier()  
decision_tree.fit(attributes, labels)
```

- Model hyperparameters
 - criterion: "gini" (default), "entropy"
 - max_depth
 - max_features (usually we don't change this)
- Outputs
 - feature_importances_ – Gini scores for all features
 - n_classes_, n_features_

Visualizing Decision Tree Boundaries

- For simplicity, let's use the Iris dataset

```
from sklearn.datasets import load_iris  
iris = load_iris()
```

- This method can be applied to all classifiers, not only trees
 - Select 2 features (for a 2D plot)
 - Predict class values for a "mesh" of evenly-spaced samples
 - Plot the test data and predicted values in different colors (classes)

```
X = iris.data[:, :2] # Sepal length, sepal width  
y = iris.target  
h = 0.02 # Step size  
color_dict = {0: "blue", 1: "lightgreen", 2: "red"}  
colors = [color_dict[i] for i in y]  
  
depth_2 = DecisionTreeClassifier(max_depth = 2).fit(X, y)  
depth_4 = DecisionTreeClassifier(max_depth = 4).fit(X, y)  
titles = ["Max depth = 2", "Max depth = 4"]
```

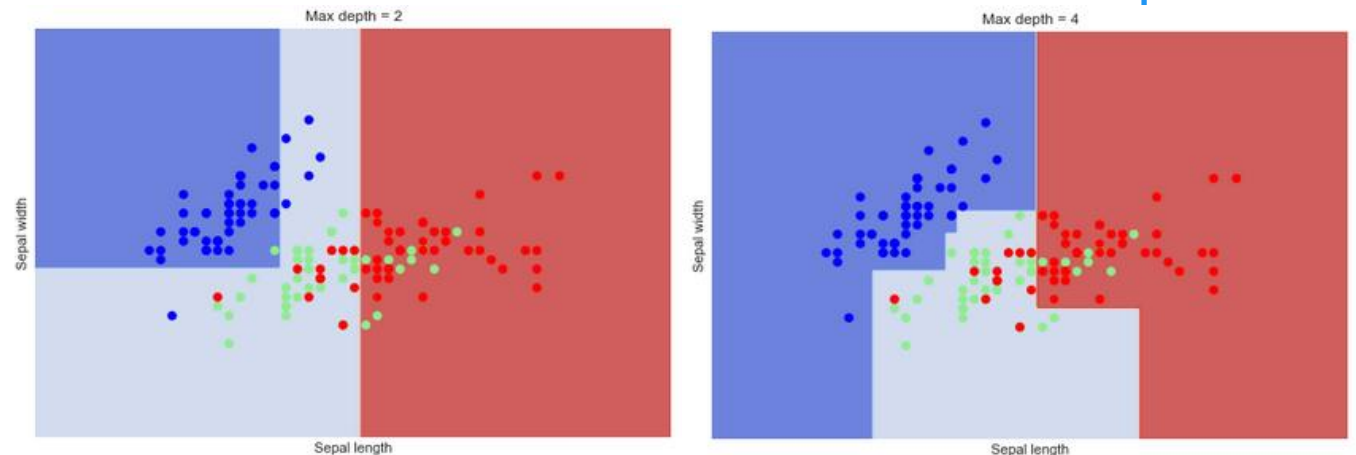
Visualizing Decision Tree Boundaries (2)

- Create a mesh

```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

- Create and evaluate predictions for all classifiers

```
for i, classifier in enumerate((depth_2, depth_4)):
    plt.figure()
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap = plt.cm.coolwarm, alpha = 0.8)
    plt.scatter(X[:, 0], X[:, 1], c = colors)
    plt.xlabel("Sepal length")
    plt.ylabel("Sepal width")
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[i])
plt.show()
```





Decision Forests

It's even harder to decide...

Random Forests

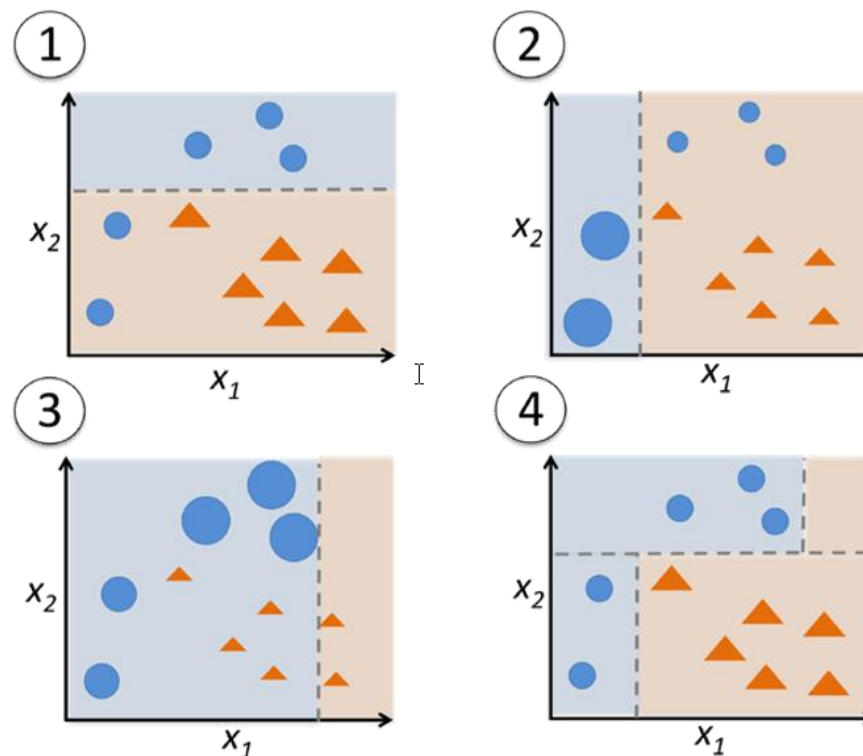
- Combinations (ensembles) of decision trees
- Idea: combine many weak learners (models that perform slightly better than random)
 - Draw a bootstrap sample (random with replacement) of size n
 - Grow k decision trees on the bootstrap sample
 - At each node, **randomly select d features** and split based on max IG
 - Aggregate the prediction by majority vote
- Differences with decision trees
 - Forests use a random subset of features (trees use all features)
 - A little harder to interpret than decision trees :(
- Advantages :)))
 - Better (lower) generalization error
 - Less susceptible to overfitting
 - Less hyperparameter tuning (in practice, we usually care about k only)

AdaBoost

- Short for "**Ada**ptive **Boost**ing"
 - Another method to combine weak learners into a strong one
- Algorithm
 - Train a weak learner on a random subset (without replacement) of the test data
 - Draw another random subset and add 50% of the previously misclassified samples; train another weak learner on that
 - Find the training samples on which both learners disagree to train a third weak learner
 - Combine the three weak learners via majority voting
- Those algorithms tend to overfit the data
 - We have to check variance carefully

AdaBoost (2)

1. All samples have equal weight
 - First classifier: dashed line minimizes an error function
2. Assign larger weights to misclassified samples, lower weights to correctly classified samples
 - Second classifier: "focuses" on misclassified samples
3. The same as step 2 (we can perform many rounds of boosting)
 - Third classifier
4. End result: combination of all weak learners
 - Resulting classifier: combined results
 - Majority vote



Testing AdaBoost

- Use an AdaBoost classifier to combine 100 "decision stumps" (i.e. decision trees with depth 1)
 - Use the [adult income](#) dataset
- Compare the results to only one tree

```
from sklearn.metrics import accuracy_score

# Preprocessing, train / test split

# Single tree
tree = DecisionTreeClassifier(max_depth = 1)
tree.fit(features_train, labels_train)
train_pred = accuracy_score(labels_train, tree.predict(features_train))
test_pred = accuracy_score(labels_test, tree.predict(features_test))
print(
    "Decision tree train / test accuracies: %.3f / %.3f",
    (train_pred, test_pred))
```


Testing AdaBoost (2)

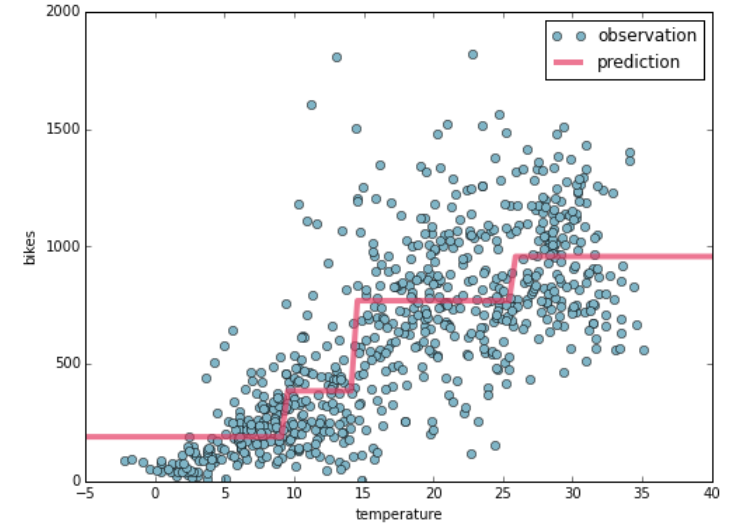
```
from sklearn.ensemble import AdaBoostClassifier
# Boosted tree
tree = DecisionTreeClassifier(max_depth = 1)
ada = AdaBoostClassifier(base_estimator = tree,
    n_estimators = 100, learning_rate = 0.1)
ada.fit(features_train, labels_train)
train_pred = accuracy_score(labels_train, ada.predict(features_train))
test_pred = accuracy_score(labels_test, ada.predict(features_test))
print(
    "AdaBoost tree train / test accuracies: %.3f / %.3f",
    (train_pred, test_pred))
```

■ Results

- AdaBoost is better in most cases
 - Predicts the test and train data better
- AdaBoost has higher variance and reduced bias
 - Better comparison: cross validation; model selection process
 - CV + "hold-out" set

Other Algorithms

- Regression with trees and forests
 - Not commonly used because of the model function
 - Stepwise, not smooth (i.e. no gradients), tends to overfit, etc.
 - Example
- Gradient boosting
 - Boosting algorithm (similar to AdaBoost)
 - Trains on the remaining errors
 - Doesn't modify the sampling distribution
 - Example
 - **XGBoost** – e**X**treme **G**radient **B**oosting
 - Usually faster, performs better on large datasets



Summary

- Decision trees
- Ensemble algorithms
 - Random decision forests
 - AdaBoost

The image features a white background with two blue decorative bars. The top bar is a solid blue band at the very top. Below it is a white space containing the text. At the bottom, there is another white space, followed by a dark blue band and a final solid blue band at the very bottom. The text 'Questions?' is centered in the white space.

Questions?