EEC1509 - Machine Learning

Lesson #10 Random Forest

Ivanovitch Silva

November, 2018

Update repository

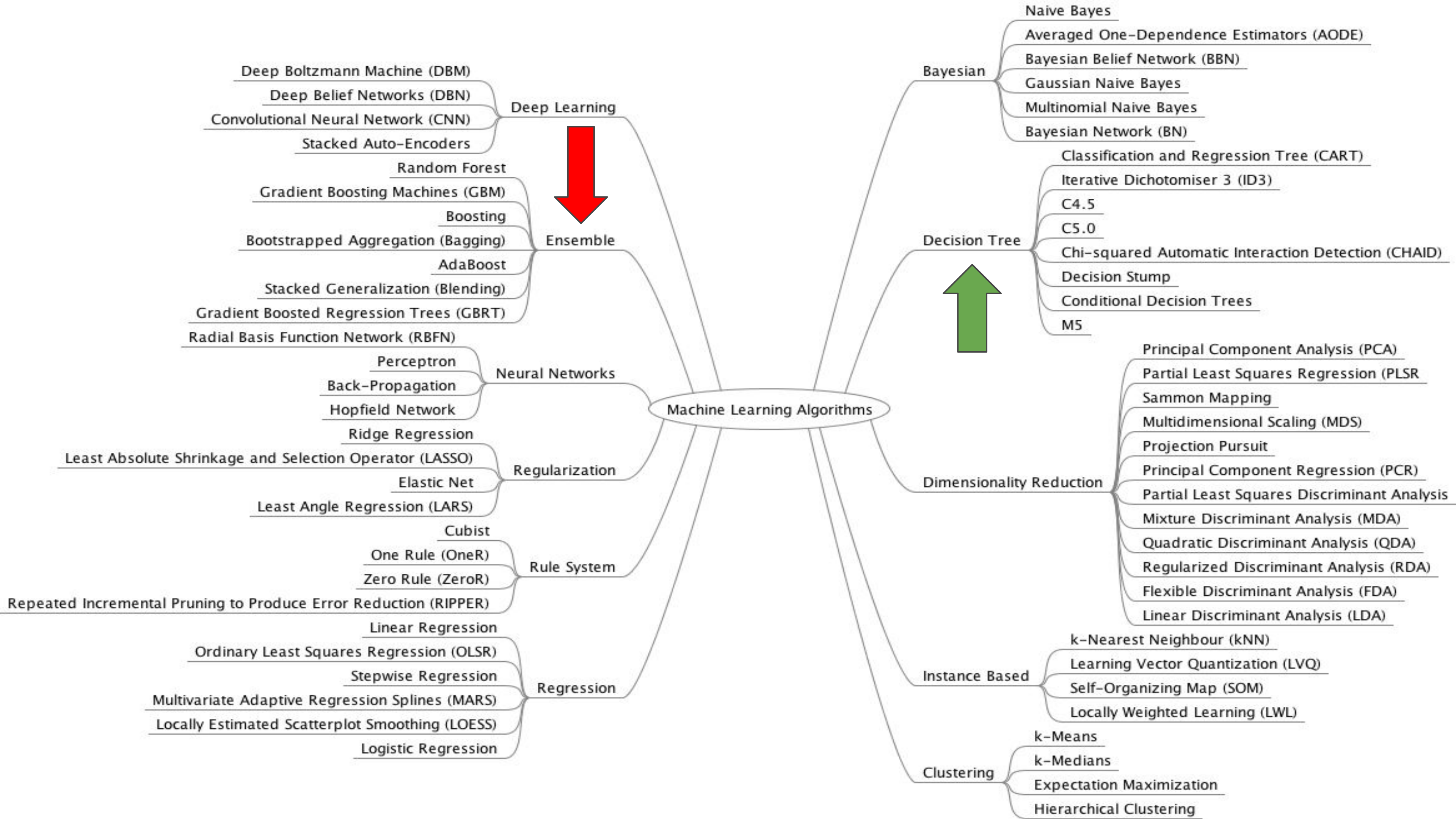
```
git clone https://github.com/ivanovitchm/EEC1509_MachineLearning.git
```

Ou ….

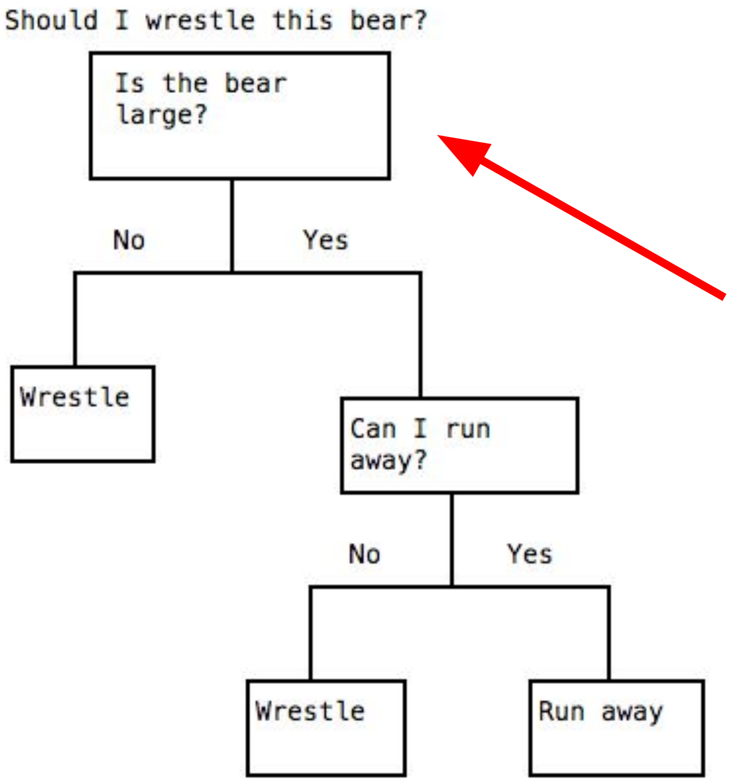
```
git pull
```

Agenda

- Previously on last class (Decision Trees)
- Ensembles (Random Forest)
- Combining predictions
- Why Ensembling works
- Introduction variation with bagging and random features
- Reducing overfitting using Random Forest


Machine Learning Algorithms
Deep Learning
Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders
Ensemble
Random Forest
Gradient Boosting Machines (GBM)
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (Blending)
Gradient Boosted Regression Trees (GBRT)
Neural Networks
Radial Basis Function Network (RBFN)
Perceptron
Back-Propagation
Hopfield Network
Regularization
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least Angle Regression (LARS)
Rule System
Cubist
One Rule (OneR)
Zero Rule (ZeroR)
Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
Regression
Linear Regression
Ordinary Least Squares Regression (OLSR)
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Logistic Regression
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Gaussian Naive Bayes
Multinomial Naive Bayes
Bayesian Network (BN)
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
C5.0
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
Conditional Decision Trees
M5
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Mixture Discriminant Analysis (MDA)
Quadratic Discriminant Analysis (QDA)
Regularized Discriminant Analysis (RDA)
Flexible Discriminant Analysis (FDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)
Clustering
k-Means
k-Medians
Expectation Maximization
Hierarchical Clustering

Decision Tree (classification)

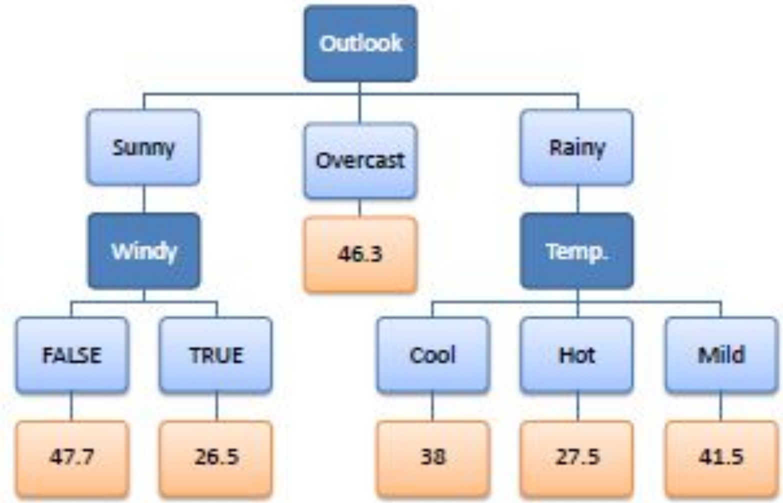


Bear name	Size	Escape possible?	Action
Yogi	Small	No	Wrestle
Winnie	Small	Yes	Wrestle
Baloo	Large	Yes	Run away
Gentle Ben	Large	No	Wrestle

Decision Tree (regression)

Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	46
Sunny	Mild	High	False	45
Sunny	Cool	Normal	False	52
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	35
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	46
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30

Predictors: Outlook, Temp., Humidity, Windy — Target: Hours Played

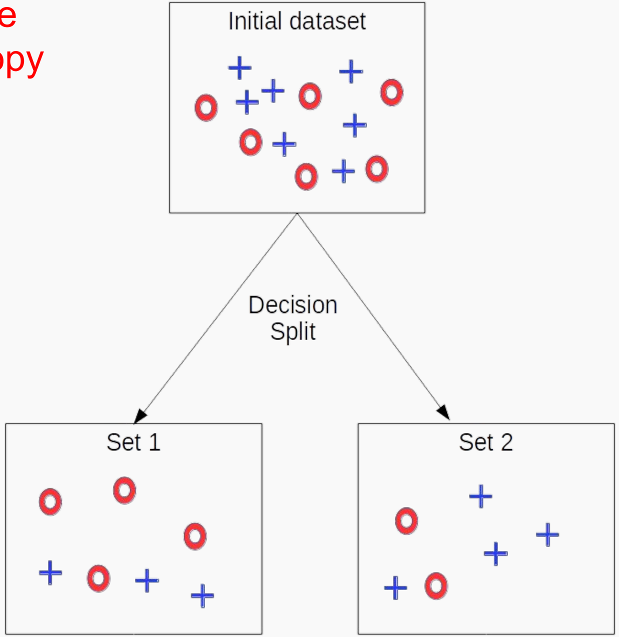


Entropy

Entropy is an indicator of how messy your data is.

Why Entropy in Decision Trees?

Decrease
the entropy



- The goal is to tidy the data.
- You try to separate your data and group the samples together in the classes they belong to.
- You maximize the purity of the groups as much as possible each time you create a new node of the tree
- Of course, at the end of the tree, you want to have a clear answer.

Applying Decision Tree

sklearn.tree.DecisionTreeClassifier

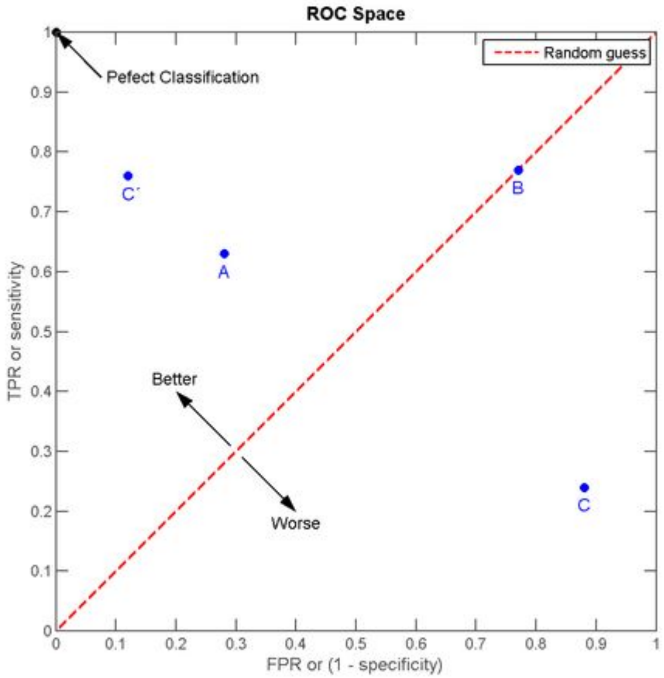
class `sklearn.tree.` **`DecisionTreeClassifier`** (*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False*)

[source]

sklearn.tree.DecisionTreeRegressor

class `sklearn.tree.` **`DecisionTreeRegressor`** (*criterion='mse', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort=False*) ¶ [source]

Receiver Operating Characteristic (ROC)

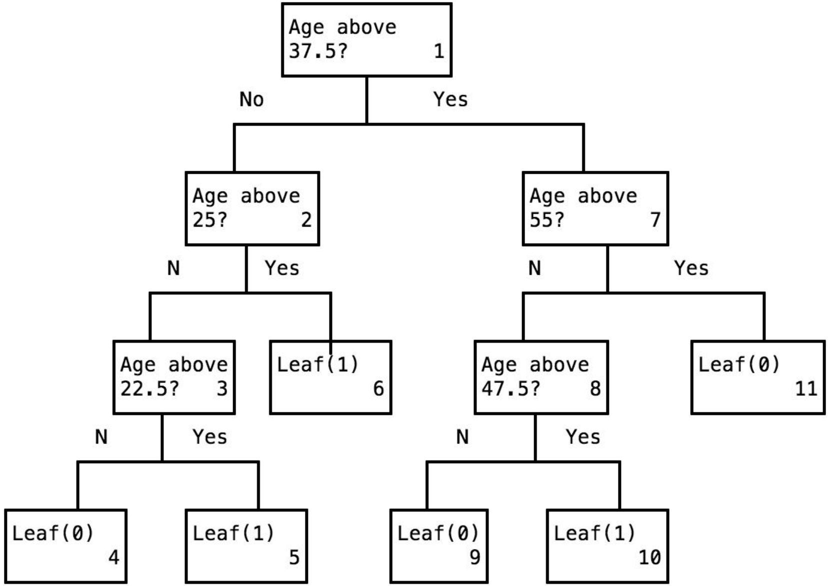


		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$

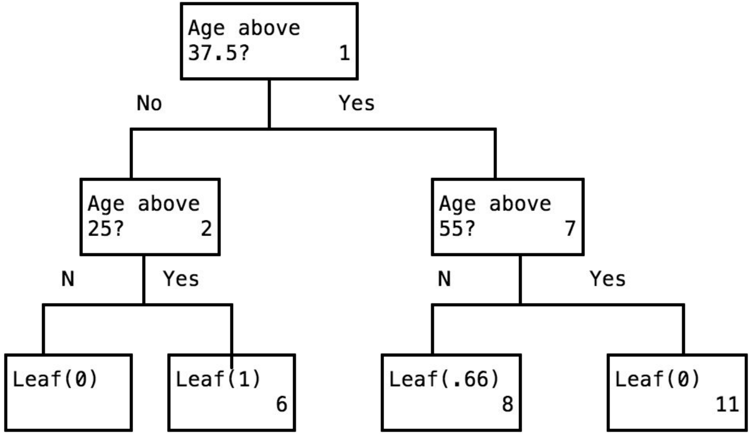
AUC - Area Under Curve

Decision Tree Overfitting

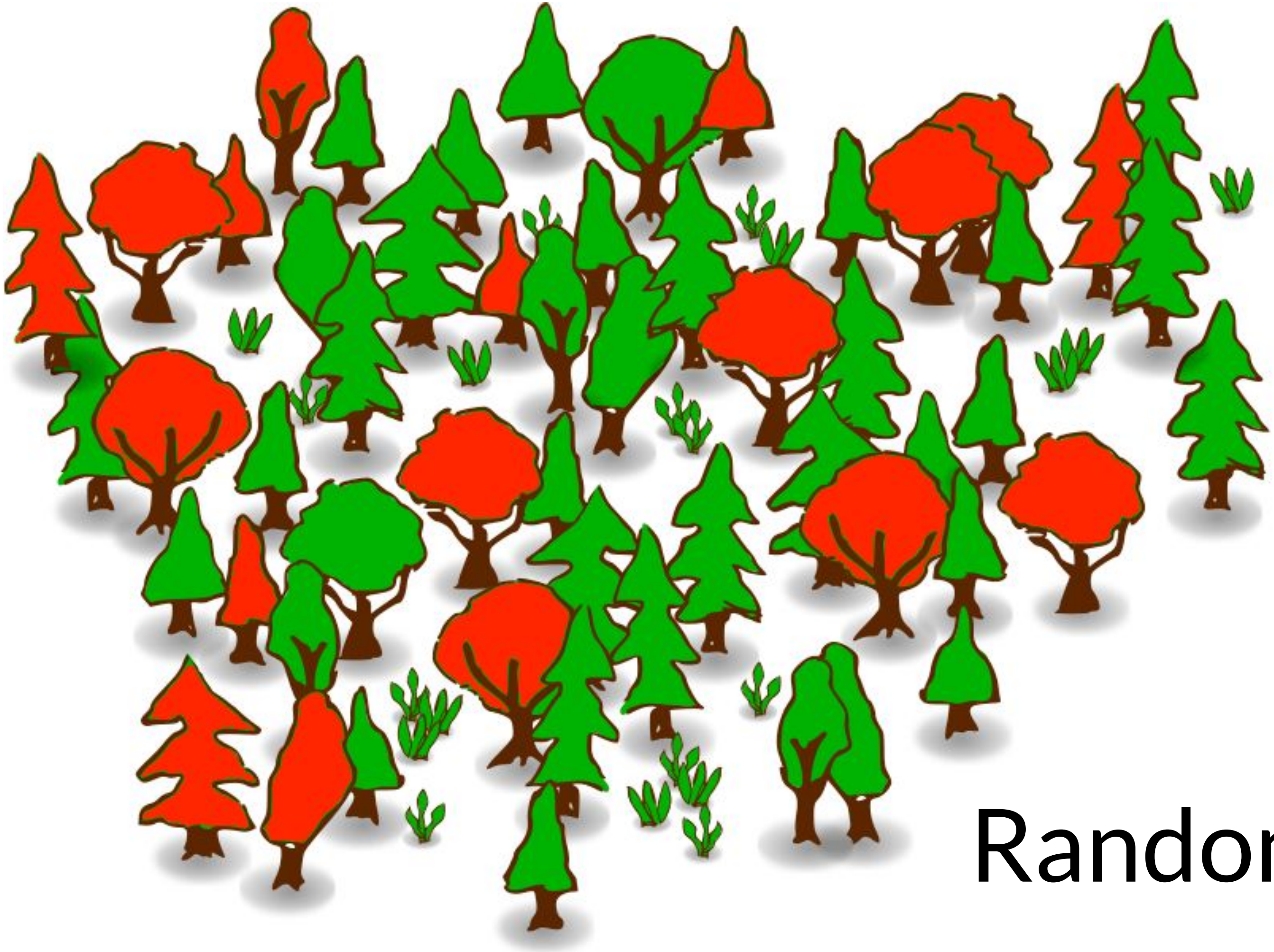
Full tree



Smaller tree



settings	train AUC	test AUC
default (min_samples_split: 2, max_depth: None)	0.947	0.694
min_samples_split: 13	0.842	0.699
min_samples_split: 13, max_depth: 7	0.748	0.743
min_samples_split: 100, max_depth: 2	0.662	0.655



Random Forest

The Dataset

The target column, or what we want to predict, is whether individuals make less than or equal to 50k a year, or more than 50k a year.

US Census 1994

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0

Combining Model Predictions with Ensembles

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score

# features
columns = ["age", "workclass", "education_num", "marital_status",
           "occupation", "relationship", "race", "sex",
           "hours_per_week", "native_country"]

# model 1
clf = DecisionTreeClassifier(random_state=1, min_samples_leaf=2)
clf.fit(train[columns], train["high_income"])

# model 2
clf2 = DecisionTreeClassifier(random_state=1, max_depth=5)
clf2.fit(train[columns], train["high_income"])

# prediction on model 1
predictions = clf.predict(test[columns])
print(roc_auc_score(test["high_income"], predictions))

# prediction on model 2
predictions = clf2.predict(test[columns])
print(roc_auc_score(test["high_income"], predictions))
```

```
0.6878964226062301
0.6759853906508785
```

Combining our predictions

Majority Voting

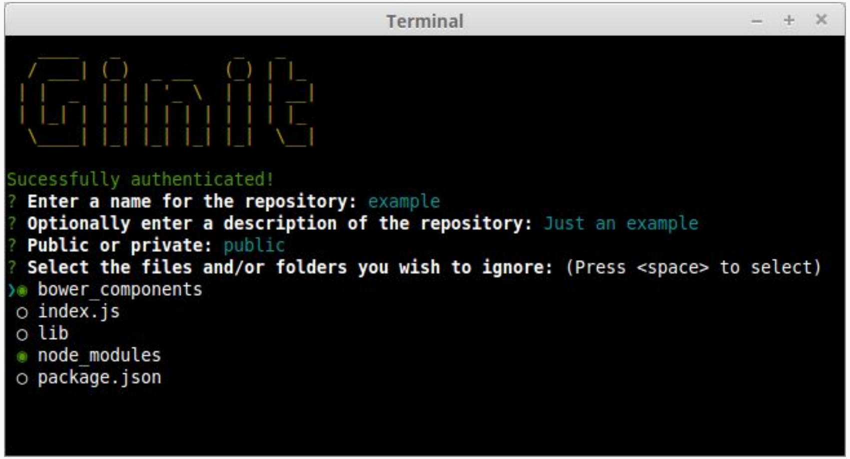
DT1	DT2	DT3	Final Prediction
0	1	0	0
1	1	1	1
0	0	1	0
1	0	0	0

settings	test AUC
min_samples_leaf: 2	0.688
max_depth: 2	0.676
combined predictions	0.715

	#1 Model	#2 Model	Mean	Rounded
0	0.166667	0.288507	0.227587	0.0
1	0.000000	0.288507	0.144253	0.0
2	0.000000	0.180918	0.090459	0.0
3	0.000000	0.354167	0.177083	0.0
4	0.000000	0.041009	0.020504	0.0
5	0.000000	0.006875	0.003437	0.0
6	0.000000	0.006875	0.003437	0.0
7	0.333333	0.146179	0.239756	0.0
8	0.000000	0.006875	0.003437	0.0
9	0.666667	0.777431	0.722049	1.0
10	0.000000	0.041009	0.020504	0.0

Probability Voting

Why Ensembling Works



Introduction Variation with Bagging

```python
# We'll build 10 trees
tree_count = 10

# Each "bag" will have 60% of the number of original rows
bag_proportion = .6

predictions = []
for i in range(tree_count):
    # We select 60% of the rows from train, sampling with replacement
    # We set a random state to ensure we'll be able to replicate our results
    # We set it to i instead of a fixed value so we don't
    # get the same sample in every loop.
    bag = train.sample(frac=bag_proportion, replace=True, random_state=i)

    # Fit a decision tree model to the "bag"
    clf = DecisionTreeClassifier(random_state=1, min_samples_leaf=2)
    clf.fit(bag[columns], bag["high_income"])

    # Using the model, make predictions on the test data
    predictions.append(clf.predict_proba(test[columns])[:,1])
combined = np.sum(predictions, axis=0) / 10
rounded = np.round(combined)

print(roc_auc_score(test["high_income"], rounded))
```

```
0.7329963297474371
```

settings	test AUC
min_samples_leaf: 2	0.688
max_depth: 2	0.676
combined predictions	0.715
min_samples_leaf: 2, with bagging	0.732

Introduction Variation from Random Features

```
# Fit a decision tree model to the "bag"
clf = DecisionTreeClassifier(random_state=1, min_samples_leaf=2,
                             splitter="random", max_features="auto")
clf.fit(bag[columns], bag["high_income"])
```

settings	test AUC
min_samples_leaf: 2	0.688
max_depth: 2	0.676
combined predictions	0.715
min_samples_leaf: 2, with bagging	0.732
min_samples_leaf: 2, with bagging and random subsets	0.734

Put it All Together

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=5, random_state=1,
                             min_samples_leaf=2)
clf.fit(train[columns], train["high_income"])

predictions = clf.predict(test[columns])
print(roc_auc_score(test["high_income"], predictions))
```

```
0.7347461391939776
```

Reducing Overfitting

```
clf = DecisionTreeClassifier(random_state=1, min_samples_leaf=5)

clf.fit(train[columns], train["high_income"])

predictions = clf.predict(train[columns])
print(roc_auc_score(train["high_income"], predictions))

predictions = clf.predict(test[columns])
print(roc_auc_score(test["high_income"], predictions))
```

```
0.8192570489534683
0.7139325899284541
```

Reducing Overfitting

```
clf = RandomForestClassifier(n_estimators=150, random_state=1,
                             min_samples_leaf=5)
clf.fit(train[columns], train["high_income"])

predictions = clf.predict(train[columns])
print(roc_auc_score(train["high_income"], predictions))

predictions = clf.predict(test[columns])
print(roc_auc_score(test["high_income"], predictions))
```

```
0.7917047295143252
0.7498874343962398
```

When to use Random Forest

- Strengths of a Random Forest
  - Very accurate predictions
  - Resistance to overfitting
- Weakness
  - They are difficult to interpret
  - They take longer to create

Lesson #10 - Random Forest.ipynb