Understanding Random Forests

Marc Garcia

PyData Madrid - April 9th, 2016

Introduction

About me: @datapythonista

- **Python** programmer since 2006
- **Master in AI** from UPC
- Working for **Bank of America**
- http://datapythonista.github.io

Overview: Understanding Random Forests

- How **decision trees** work
- **Training** Random Forests
- Practical **example**
- Analyzing model **parameters**

How decision trees work

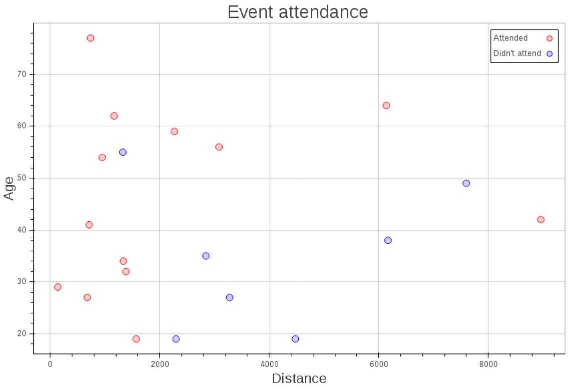
Using a decision tree

Simple example:

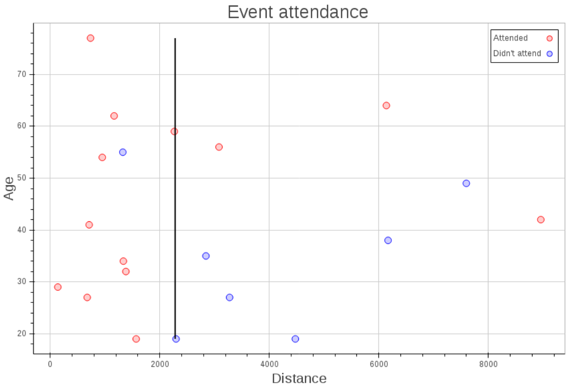
- Event attendance: binary classification
- 2 explanatory variables: age and distance

```
from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()
dtree.fit(df[['age', 'distance']], df['attended'])

cart_plot(dtree)
```

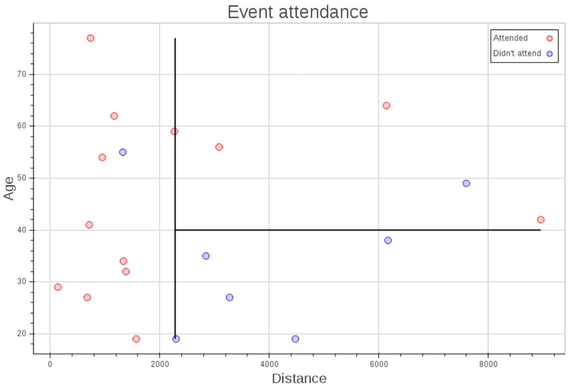
Data visualization



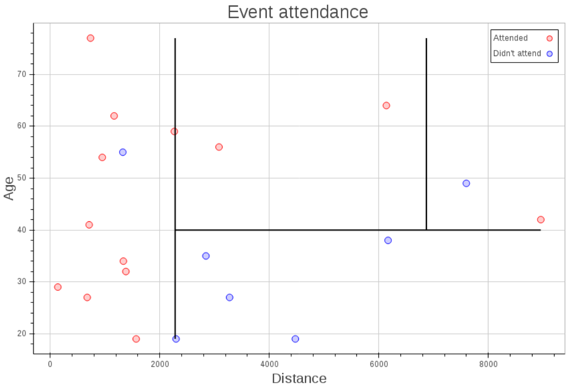
Decision boundary visualization



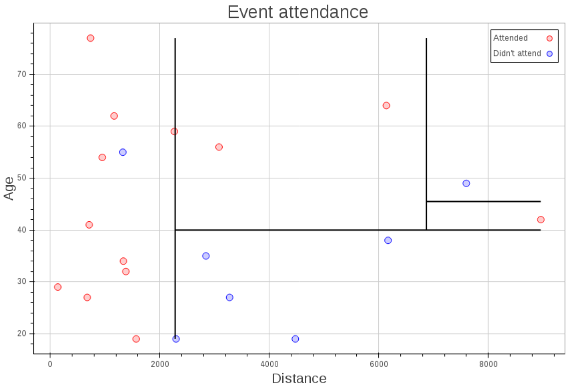
Decision boundary visualization



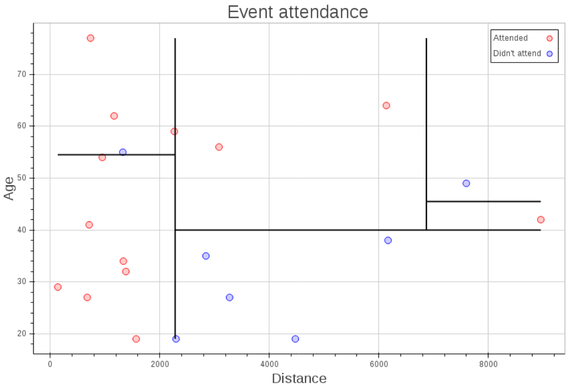
Decision boundary visualization



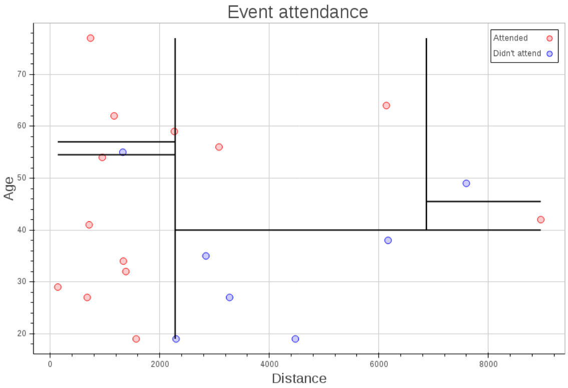
Decision boundary visualization



Decision boundary visualization

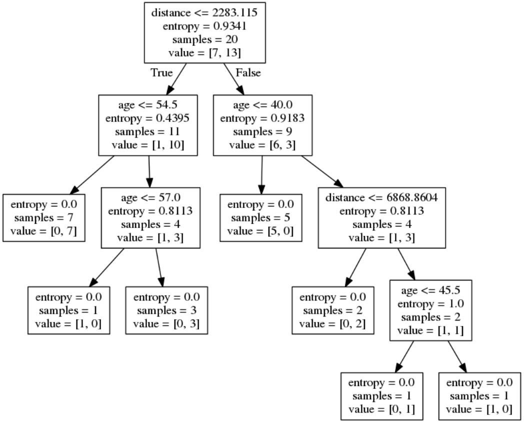


Decision boundary visualization



How is the model?

```python
def decision_tree_model(age, distance):
    if distance >= 2283.11:
        if age >= 40.00:
            if distance >= 6868.86:
                if distance >= 8278.82:
                    return True
                else:
                    return False
            else:
                return True
        else:
            return False
    else:
        if age >= 54.50:
            if age >= 57.00:
                return True
            else:
                return False
        else:
            return True
```



Training: Basic algorithm

```python
def train_decision_tree(x, y):
    feature, value = get_best_split(x, y)

    x_left, y_left = x[x[feature] < value], y[x[feature] < value]
    if len(y_left.unique()) > 1:
        left_node = train_decision_tree(x_left, y_left)
    else:
        left_node = None

    x_right, y_right = x[x[feature] >= value], y[x[feature] >= value]
    if len(y_right.unique()) > 1:
        right_node = train_decision_tree(x_right, y_right)
    else:
        right_node = None

    return Node(feature, value, left_node, right_node)
```

Best split

Candidate split 1

age	18	19	21	27	29	34	38	42	49	54	62	64
attended	F	F	T	F	T	T	F	T	F	T	T	T

Split	True	False
Left	0	1
Right	7	4

Candidate split 2

age	18	19	21	27	29	34	38	42	49	54	62	64
attended	F	F	T	F	T	T	F	T	F	T	T	T

Split	True	False
Left	0	2
Right	7	3

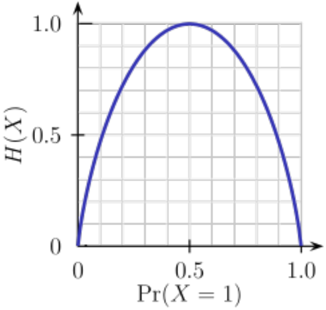
Best split algorithm

```
def get_best_split(x, y):
    best_split = None
    best_entropy = 1.
    for feature in x.columns.values:
        column = x[feature]
        for value in column.iterrows():
            a = y[column < value] == class_a_value
            b = y[column < value] == class_b_value
            left_weight = (a + b) / len(y.index)
            left_entropy = entropy(a, b)

            a = y[column >= value] == class_a_value
            b = y[column >= value] == class_b_value
            right_items = (a + b) / len(y.index)
            right_entropy = entropy(a, b)

            split_entropy = left_weight * left_etropy + right_weight * right_entropy
            if split_entropy < best_entropy:
                best_split = (feature, value)
                best_entropy = split_entropy

    return best_split
```

Entropy

For a given **subset**[1]:

$$entropy = -Pr_{attending} \cdot \log_2 Pr_{attending} - Pr_{\neg attending} \cdot \log_2 Pr_{\neg attending} \quad (1)$$



[1] Note that for pure splits it's assumed that $0 \cdot \log_2 0 = 0$

Entropy algorithm

```
import math

def entropy(a, b):
    total = a + b
    prob_a = a / total
    prob_b = b / total
    return - prob_a * math.log(prob_a, 2) \
           - prob_b * math.log(prob_b, 2)
```

Information gain

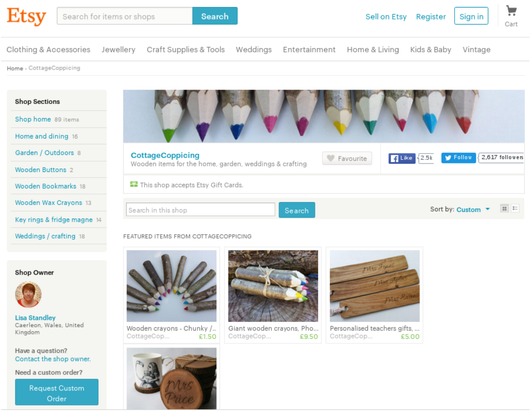
For a given **split**:

$$information\_gain = entropy_{parent} - \left(\frac{items_{left}}{items_{total}} \cdot entropy_{left} + \frac{items_{right}}{items_{total}} \cdot entropy_{right} \right) \quad (2)$$

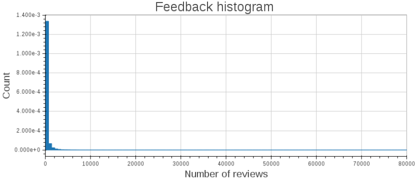
Practical example

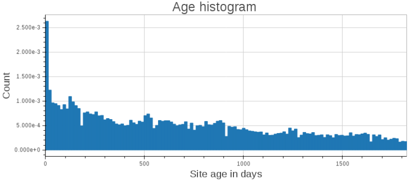
Etsy dataset

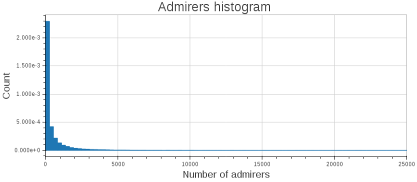
- Features:
  - Feedback (number of reviews)
  - Age (days since shop creation)
  - Admirers (likes)
  - Items (number of products)
- Response variable:
  - Sales (number)
- Samples:
  - 58,092
- Source:
  - www.bigml.com

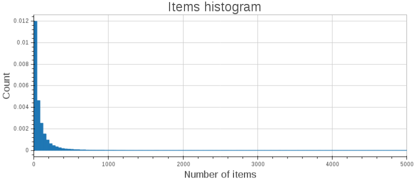


Features distribution

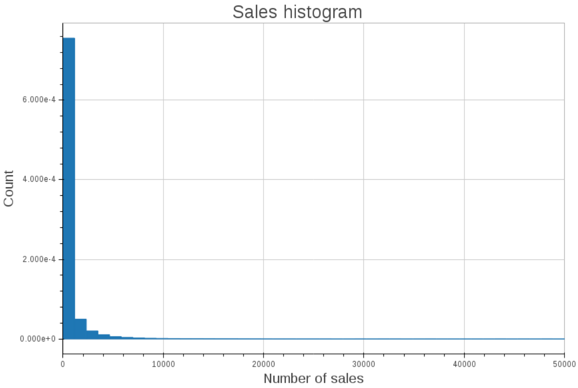








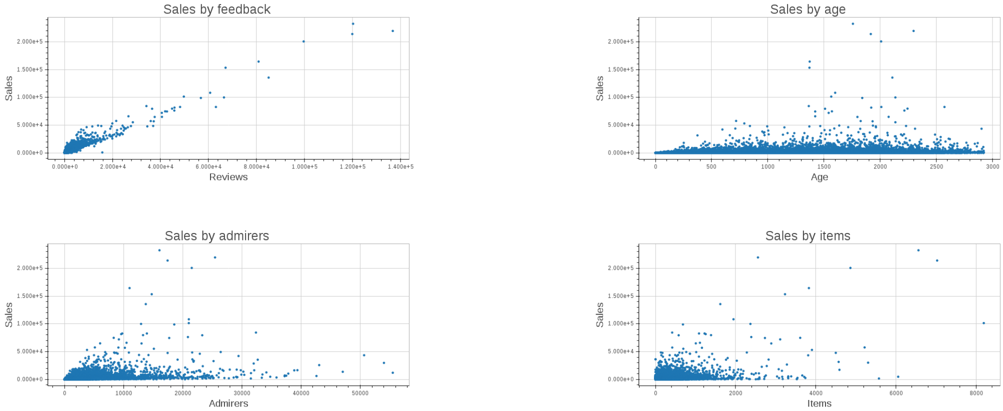
Sales visualization



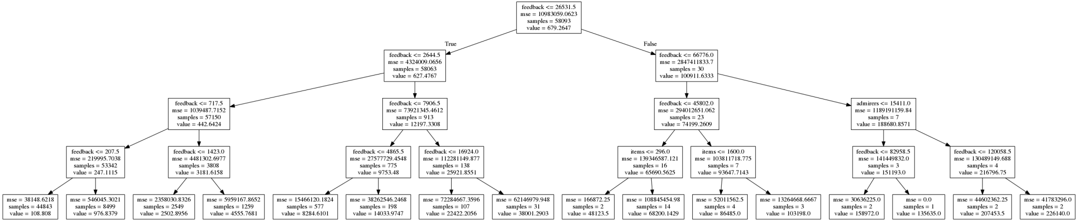
Feature correlation

	sales	admirers	age	feedback	items
sales	1.000000	0.458261	0.181045	0.955949	0.502423
admirers	0.458261	1.000000	0.340939	0.401995	0.268985
age	0.181045	0.340939	1.000000	0.184238	0.167535
feedback	0.955949	0.401995	0.184238	1.000000	0.458955
items	0.502423	0.268985	0.167535	0.458955	1.000000

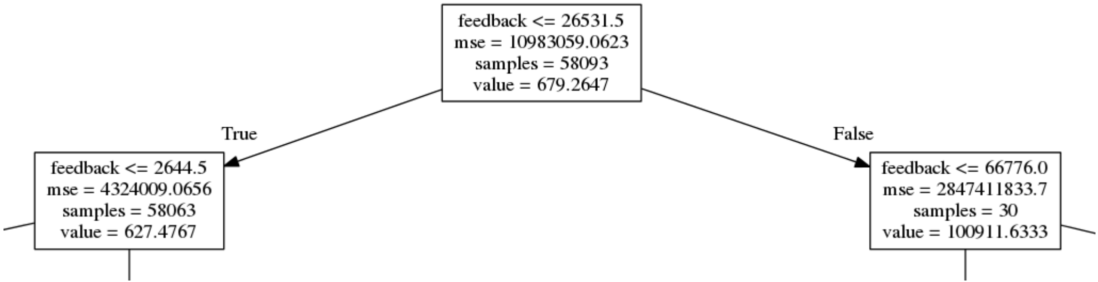
Correlation to sales



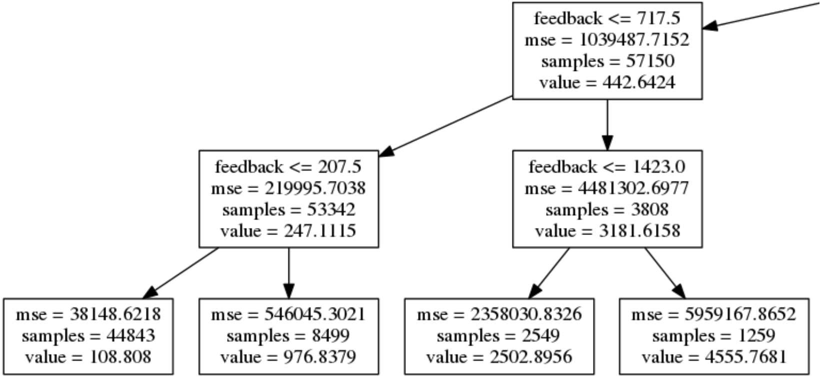
Decision tree visualization



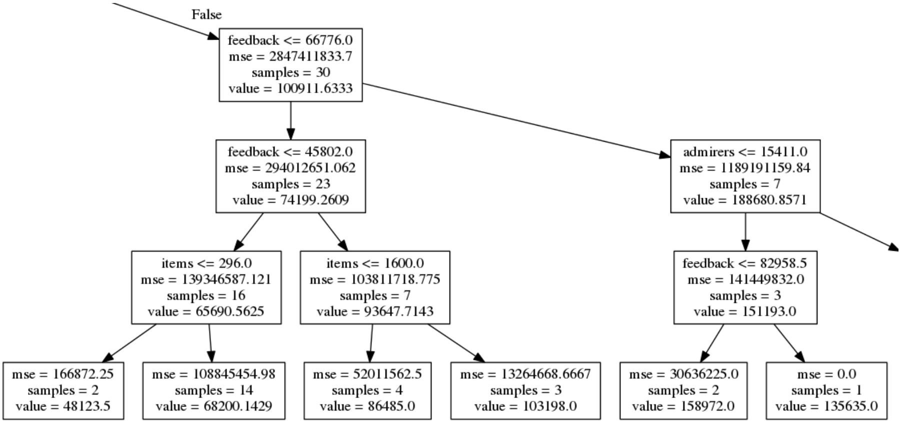
Decision tree visualization (top)



Decision tree visualization (left)



Decision tree visualization (right)



What is wrong with the tree?

- Only **one variable** used to predict in some cases
- Lack of **stability**:
  - If feedback = 207 we predict 109 sales
  - If feedback = 208 we predict 977 sales
  - Our model can change dramatically if we add few more samples to the dataset
- **Overfitting**: We make predictions based on a single sample

Training Random Forests

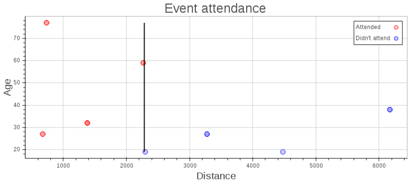
Ensembles: The board of directors

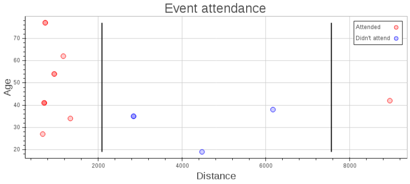
- Why companies have a BoD?
- What is best?
  - The best point of view
  - A mix of good points of view
- How was our best tree?
  - What about a mix of not so good trees?

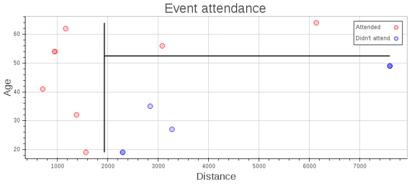
Ensemble of trees

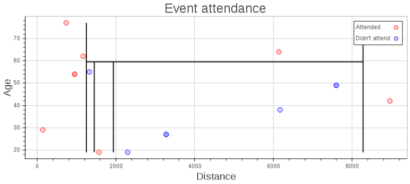
- How do we build optimized trees, that are different?
  - We need to **randomize** them
    - Samples: bagging
    - Features
    - Splits

Bagging: Uniform sampling with replacement









Randomizing features

- Even with bagging, trees will usually make the top splits by the same feature
- **max_features** parameter:
  - Regression: Use all
  - Classification: Use the squared root

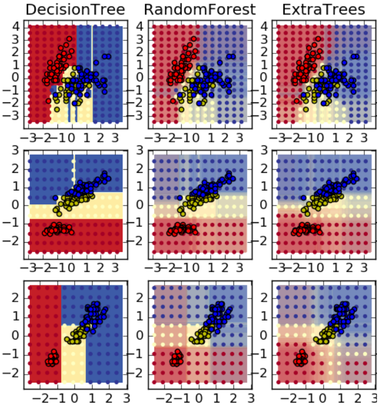
Randomizing splits

- A split per sample is considered in the original decision tree
- We can consider a subset
    - We obtain randomized trees
    - Performance is improved
- **sklearn** implements **ExtraTreeRegressor** and **ExtraTreeClassifier**

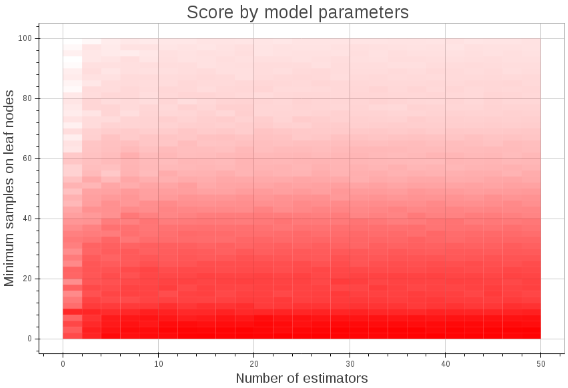
What happened with our example problems

- Only **one variable** used to predict in some cases
  - **We can avoid it by not using all features**
- Lack of **stability**:
  - **Our model is smoother**
- **Overfitting**: We make predictions based on a single sample
  - **Overfitting is mostly avoided**
  - Random forests come with built-in cross validation

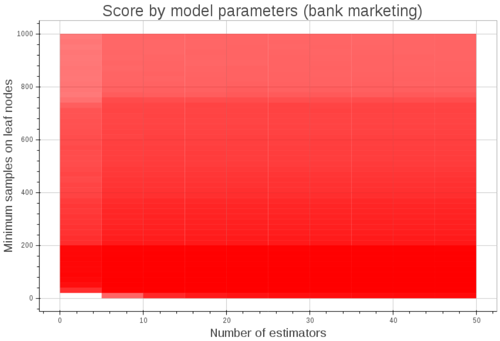
Smoothing



Parameter optimization (Etsy)



Parameter optimization (Bank marketing)



Data source: [Moro et al., 2014] S. Moro, P. Cortez and P. Rita.

A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

When should we use Random Forests?

- Using a CART is usually good as part of exploratory analysis
- Do **not** use CART or RF when the problem is linear
- They are specially good for ordinal (categorical but sorted) variables
- Performance
  - CART is slower than linear models (e.g. logit)
  - RF trains N trees, and takes N times **but it works in multiple cores, seriously**
  - But RF is usually much faster than ANN or SVM, and results can be similar

Summary

- CART is a simple, but still powerful model
  - Visualizing them we can better understand our data
- Ensembles usually improve the predictive power of models
- Random Forests fix CART problems
  - Better use of features
  - More stability
  - Better generalization (RFs avoid overfitting)
- Random Forests main parameters
  - min_samples_leaf (inherited from CART)
  - num_estimators

Thank you

QUESTIONS?