UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

Random Forest (and Boosting)

Machine Learning

Recall Decision Trees

- Use a metric to determine how to divide the data into regions
    - Metrics: entropy, gini (eg. impurity)
    - Recursive (Binary) Splitting: continue dividing data into regions until “convergence”
- Advantages
    - Explanatory power
    - Elegant handling of irrelevant features
- Disadvantages
    - Overfitting, perhaps due to high variance
    - Mediocre performance, perhaps due to greedy nature of algorithm

Random Forest Overview

- Build multiple decision_tree instances
    - Number of trees is one hyperparameter
    - More trees = more accurate hypotheses
    - Typical to see: 10, 30, 100
- For fit:
    - Build multiple independent decision trees
    - Randomly select data for each tree
    - Randomly select features for each tree
- For predict:
    - Get independent predictions from each tree
    - Take the majority (classification) or mean (regression) of all trees

Advantages & Disadvantages

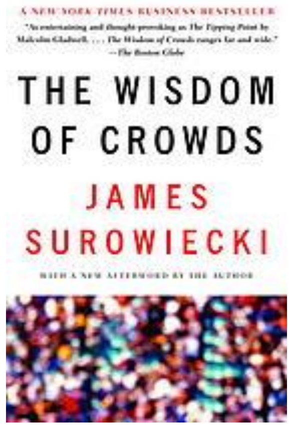
- Advantages
  - Handles missing data well
  - Handles large amounts of data very well (i.e. either large n, large p or both)
  - Handles small data well
  - Does not overfit*
- Disadvantages
  - Cannot predict outside training targets for regression
  - May overfit on noisy datasets, especially if small
  - Drives statisticians (and other theory-based people) insane
- Other random stuff
  - General purpose algorithm: can generate prediction contours of arbitrary shape
  - Used for body part prediction in MicroSoft Xbox Kinect?

Handling High Variance

- Central Limit Theorem
  - Distribution of repeated sampling of (training) data is a normal distribution
  - True even if data has high variance
- Multiple hypotheses are better than one
  - Increasing sample size helps
  - Increasing number of samples helps
  - Applies to regression as well as classification
- Bagging: Bootstrap Aggregation
  - Sample (with replacement) to create B different bootstrapped datasets

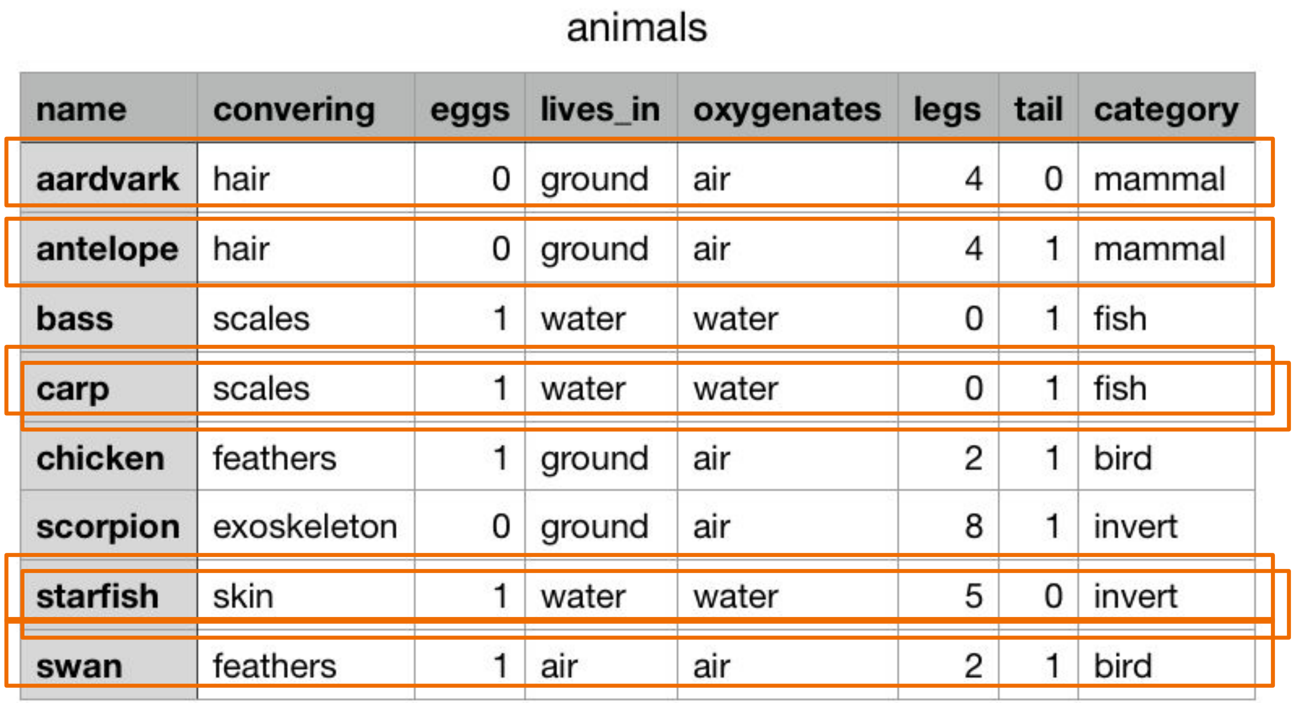
$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

  - … where $f^{*b}(x)$ = the prediction point at x for the Bth training set



By Source, Fair use, https://en.wikipedia.org/w/index.php?curid=25461520

Bagging on Animals



animals

name	convering	eggs	lives_in	oxygenates	legs	tail	category
aardvark	hair	0	ground	air	4	0	mammal
antelope	hair	0	ground	air	4	1	mammal
bass	scales	1	water	water	0	1	fish
carp	scales	1	water	water	0	1	fish
chicken	feathers	1	ground	air	2	1	bird
scorpion	exoskeleton	0	ground	air	8	1	invert
starfish	skin	1	water	water	5	0	invert
swan	feathers	1	air	air	2	1	bird

- Example:
  - Randomly generated "n" samples
  - Set 1:

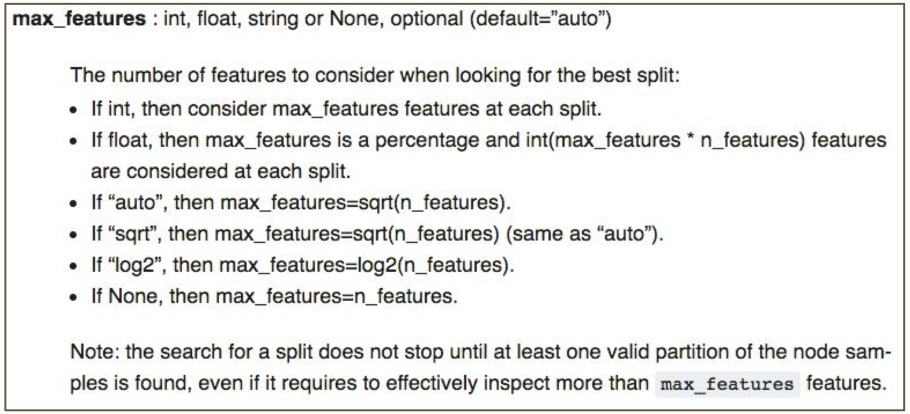
    [4, 0, 3, 3, 6, 6, 7, 1]

  - Not shown: sets 2

Handling Greedy Algorithm

- During "fit", remove one or more features from consideration
- Randomly decide on features to be removed
  - The scikit-learn implementation considers sqrt(p) features

From
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html



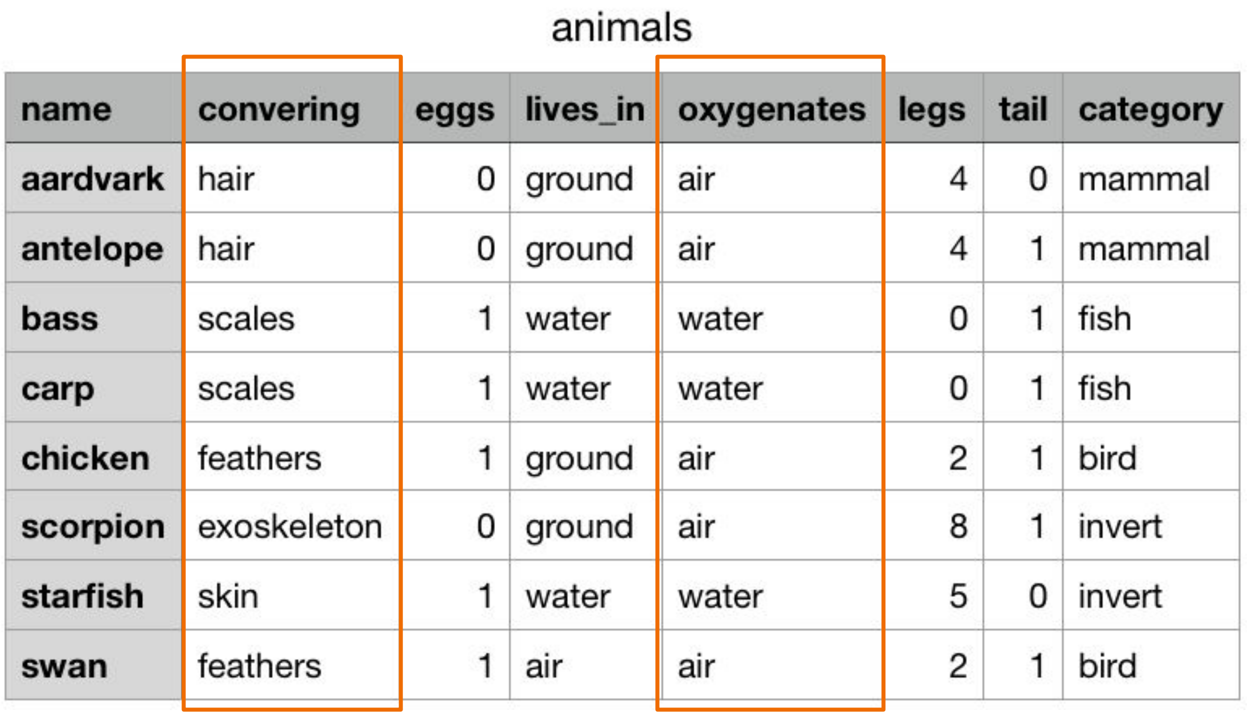
max_features : int, float, string or None, optional (default="auto")

The number of features to consider when looking for the best split:

- If int, then consider max_features features at each split.
- If float, then max_features is a percentage and int(max_features * n_features) features are considered at each split.
- If "auto", then max_features=sqrt(n_features).
- If "sqrt", then max_features=sqrt(n_features) (same as "auto").
- If "log2", then max_features=log2(n_features).
- If None, then max_features=n_features.

Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features.

Random Feature Selection



animals

name	convering	eggs	lives_in	oxygenates	legs	tail	category
aardvark	hair	0	ground	air	4	0	mammal
antelope	hair	0	ground	air	4	1	mammal
bass	scales	1	water	water	0	1	fish
carp	scales	1	water	water	0	1	fish
chicken	feathers	1	ground	air	2	1	bird
scorpion	exoskeleton	0	ground	air	8	1	invert
starfish	skin	1	water	water	5	0	invert
swan	feathers	1	air	air	2	1	bird

- Example:
  - Randomly generated “sqrt(p)” features
  - Estimator 1:

    [3, 0]

Random Forest — fit()

```
def fit(X, Y, num_trees)
    tree_list = create_list(num_trees)  # decision_tree list
    for t in tree_list:
        subsample_x, subsample_y = subsample(X, Y) # Bagging
        feature_list = sample_of_features(X) # Random features
        t.fit(subsample_x, subsample_y, feature_list)
```

Random Forest — predict()

```
def predict(X):
    from collections import defaultdict
    hypothesis_list = [t.predict(X) for t in tree_list]
    counts = defaultdict(int)
    for h in hypothesis_list:
        counts[h] += 1
    return sorted(counts.items(), reverse=True, key=lambda
tup: tup[1])[:len(tree_list)][0][0]
```

Odds & Ends

- Classically, the random forest algorithm does prune trees
    - Practically, this can be driven by the data
    - In sklearn, this is a hyperparameter (max_depth)
- In sklearn, we can determine feature importances on fitted data, eg:

```
clf.fit(X, Y)

importances = clf.feature_importances_
```

- The BaggingClassifier class in sklearn is a closely-related class
- Ensemble learning
    - One of many techniques
    - Several weak learners (performing slightly better than chance) may create a strong learner

Boosting Overview

- Observation
    - What if we train a learner (classifier) on some data then have it predict the training data?
    - Answer: The classifier makes errors even on the training set
- Can you learn from mistakes?
    - Create several (sequential) learners
    - Each learner focuses on the previous learner’s errors

AdaBoost

- Adaptive Boosting
- Algorithm overview
  - Create “B” data subsets using bagging and “B” learners, one for each bag
  - 1: Train a learner on its bag
  - 2: Test the learner against its own (training set) bag
  - 3: Add probability to training instances which were misclassified so that it’s more likely to be selected for subsequent bags
  - 4: Repeat (1) for next bag

Exercise — Occupancy Detection

- Download the Occupancy Detection dataset
  - https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+
  - Target (last column) = 1 (occupied) vs. 0 (unoccupied)
  - Features = Everything except target
  - Ignore instance number and date columns (for now)
- Without finding optimal hyperparameters, try three sklearn classifiers
  - DecisionTreeClassifier (sklearn.tree)
  - RandomForestClassifier (sklearn.ensemble)
  - AdaBoostClassifier (sklearn.ensemble)
- Answer the following:
  - What is the performance vs. datatest?
  - What is the performance vs. datatest2?
  - Can you improve the performance for each?

Next Time

- Hands-on with Random Forest (and AdaBoost / XGBoost?)
- Perceptron & Multi-Layer Perceptron