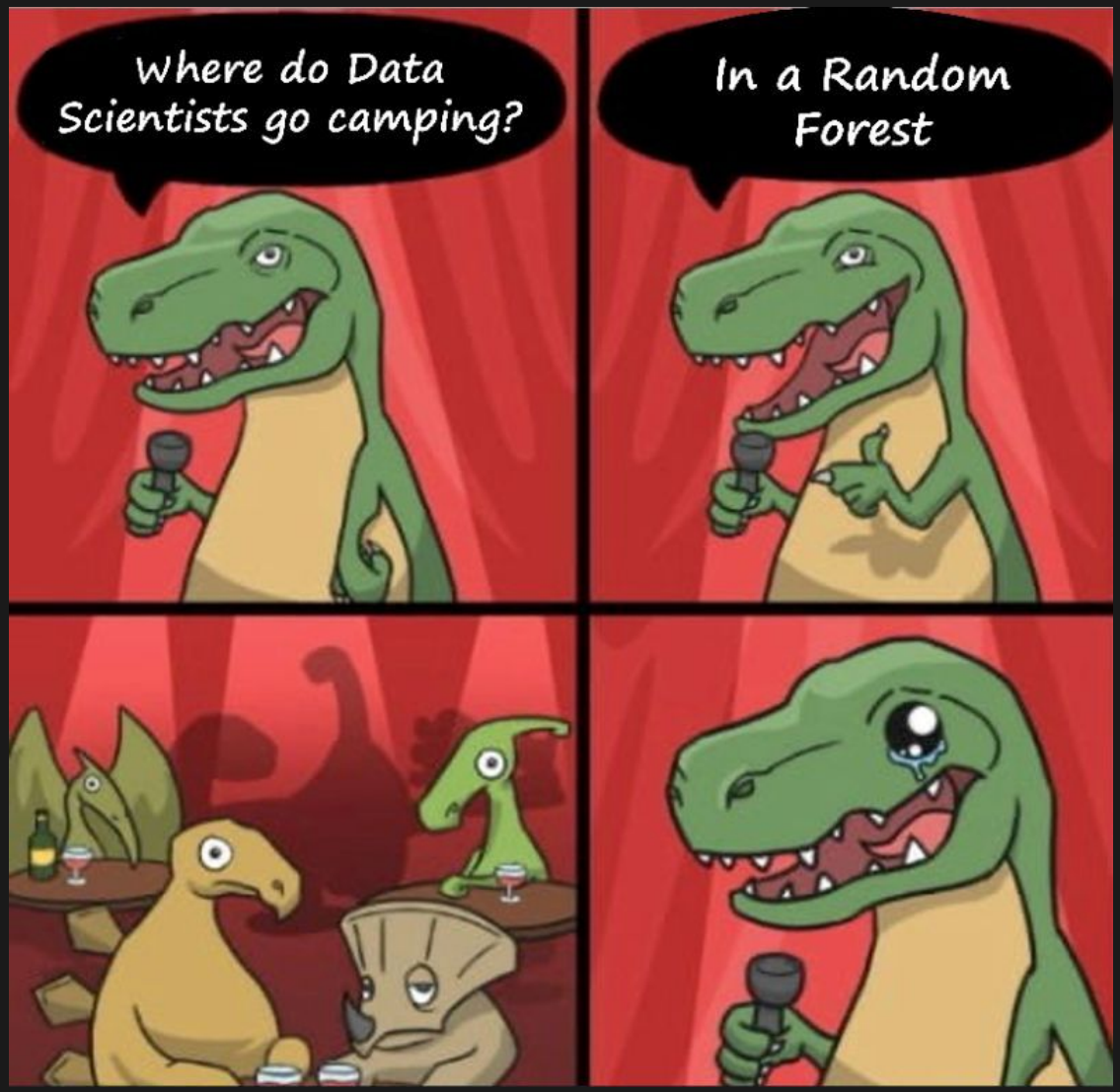
Ensembles, Forests and Boosting

Daniel Lawson University of Bristol

Lecture 03.2 (v4.0.0)



Dinosaur comics meme

Context

- Deep trees are good, could they be better?
  - Can we combine "good" models into a better model?
- Shallow trees are bad but interpretable, could they be better?
  - Can we combine many "bad" into a good model?
- More generally, what are the ways to combine models?
  - Bagging and randomizing samples, boosting

Bagging

- **B**ootstrap **Agg**regat**ing** is is simply running the same algorithm of different random subsets of the data
- In parallel, B times:
  - Form data sample $\{X\}_b$, e.g. by sampling with replacement, or leaving out a random subset of data
  - Learn a classifier $f_b(x)$
- Output: A "bagged" classifier $f(x) = \frac{1}{B}\sum_{b=1} f_m(x)$

Bagging comments

- Bagging reduces overfitting to the data, and therefore works well on complex classifiers
- Same rules for resampling apply as in statistics: e.g. it works well when you respect the correlation structure
- In theory under certain assumptions, the **distribution of bagged learners** give a distribution on:
  - "what I could have seen if I obtained new data"
  - From the same distribution I got my data
- Usually good to try it
  - Except compute cost

Random Forest

- A **random forest** is a set of **decision trees** that are combined together to perform classification.
- For each of T trees, the following steps are run:

  a) Choose which of J variables to include:

  - Choose m_f **random features**. The canonical choice is $m_f = \sqrt{J}$.
  - Like **bagging for features**? (downsampling **without** replacement)

  b) Learn a Tree classifier independently as above.

Boosting

- The general idea of **Boosting** is:
  - Build a classifier, predict the data
  - Treat the residuals as "new data"
  - Repeat
- Boosting sounds like it should work for arbitrary classifiers, but because of the iterative nature it is applied to simple classifiers.
- There are many boosting algorithms, amongst which are:
  - Adaboost[1] (Adaptive boosting - first game-changer)
  - xgboost[2] (exploits sparsity and gradients)
  - LightGBM[3] (parallelizable and efficient)

---

[1] Freund and Schapire in 1996

[2] Chen T and Carlos G (2016) KDD 2016.

[3] Microsoft dev team

Boosted feature splitter

- A very simple way to use boosting is to allow classifiers only from **single features**:
- Initialise weights of each **data sample** (uniformly)
- For T iterations:
  - Normalise weights
  - Train a classifier on every feature individually
  - Choose the best classifier, i.e. feature
  - Update the data weights by upweighting correct decisions and downweighting wrong decisions
- The boosted classifier uses a weighted sum of the selected classifiers

Boosted decision tree

- As noted previously, adaboost is using a **sequence of decisions** to make a boosted classifier.
- By default it uses a boosted depth=1 decision tree, i.e. classifiers were just the features. This is called a **decision stump**.
- You can use deeper trees[4], eg. with xgboost; usually the depth is limited to control learning cost and complexity
- Boosting in theory doesn't need trees so the difference is about **learning rate** and **computational complexity**

[4] J. Elith, J. Leathwick, and T. Hastie "A working guide to boosted regression trees" (2008). British Ecological Society.

Random Forest vs boosted decision tree

- Gradient Boosting Machine (GBM) is the go-to boosted decision tree
- GBM and RF differ in the way the trees are built, the order, and the way the results are combined
- RF can be trivially paralellized
- GBMs seem to outperform RFs under competition conditions, but may do worse with untuned parameters [5]

[5] http://fastml.com/what-is-better-gradient-boosted-trees-or-random-forest/

Stacking

- Stacking is a different way to combine multiple weak learners. It is more appropriate to combining "good classifiers" to make a meta-classifier.
- In theory a stacked classifier will always outperform its constituents if implemented appropriately[6].
  - Cross-validation and asymptotics are required for this guarantee but in practice many approaches work.

[6] van der Laan, M, Polley E, Hubbard A, "Super Learner" (2007) Statistical Applications in Genetics and Molecular Biology, Volume 6.

Super learner

- **Set up** the ensemble:
  - Specify L base classifiers.
  - Specify a metalearning algorithm.
- **Train** the ensemble:
  - Train the L base algorithms on the N training data. Use k-fold cross-validation for these learners.
  - For the $N \times L$ matrix of predictions. Form the "level one" data with this matrix and the raw data.
  - Train the metalearning algorithm on the "level one" data.
- **Predict** on new data:
  - Generate predictions from the base classifiers.
  - Feed those predictions into the meta-learner to generate the ensemble prediction.

More Stacking

- Related approaches:
  - Run any number of classification algorithms
  - Use their predictions as features
  - Use the data in addition to the predictions
- Pass this new feature set to any classification algorithm
- In practice, the best algorithm will be the one that generalises best in the test dataset. Common techniques:
  - Majority vote: use the prediction that most classifiers choose
  - Regularisation
  - Boosting-like prediction combination

Wrapup

- Key to high prediction accuracy are:
  - **Complexity**: Non-linearity helps dramatically
  - **Bias control**: Don't overfit
  - **Meta-learning**: Boosting and stacking are essential for the final few percent.

Random Forest outputs

- The Random Forest **combines decision trees** into a classification by:
  - Weighting each tree according to its performance
  - Report the weighted vote
- It is also possible to extract **feature importance**:
  - How much a feature decreases the score, averaged over all trees
  - Features that are never used will get a score of 0
  - Features that are important in every tree in which they appear will get a high score
  - Features that are correlated will often split their importance

Random Forest algorithm

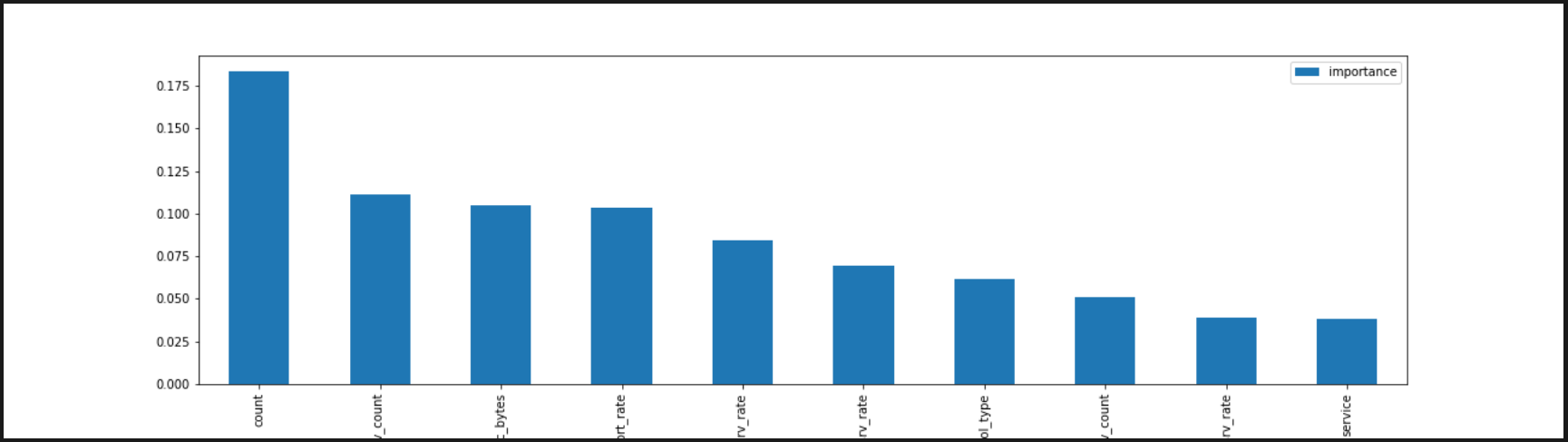
```
from sklearn.ensemble import RandomForestClassifier
clf= RandomForestClassifier(n_jobs=-1,
    random_state=3, n_estimators=102)
trained_model= clf.fit(X_train, y_train)
clf_score=trained_model.score(X_train, y_train)
y_pred = clf.predict(X_test)
```

Random Forest Feature Importance

```
feature_importances = pd.DataFrame(clf.feature_importances_,
    index = X_train.columns,
    columns=['importance']).sort_values('importance',
    ascending=False)

feature_importances.nlargest(10,
    columns=['importance']).plot(kind='bar',figsize=(18, 5))
```

Random Forest Feature Importance



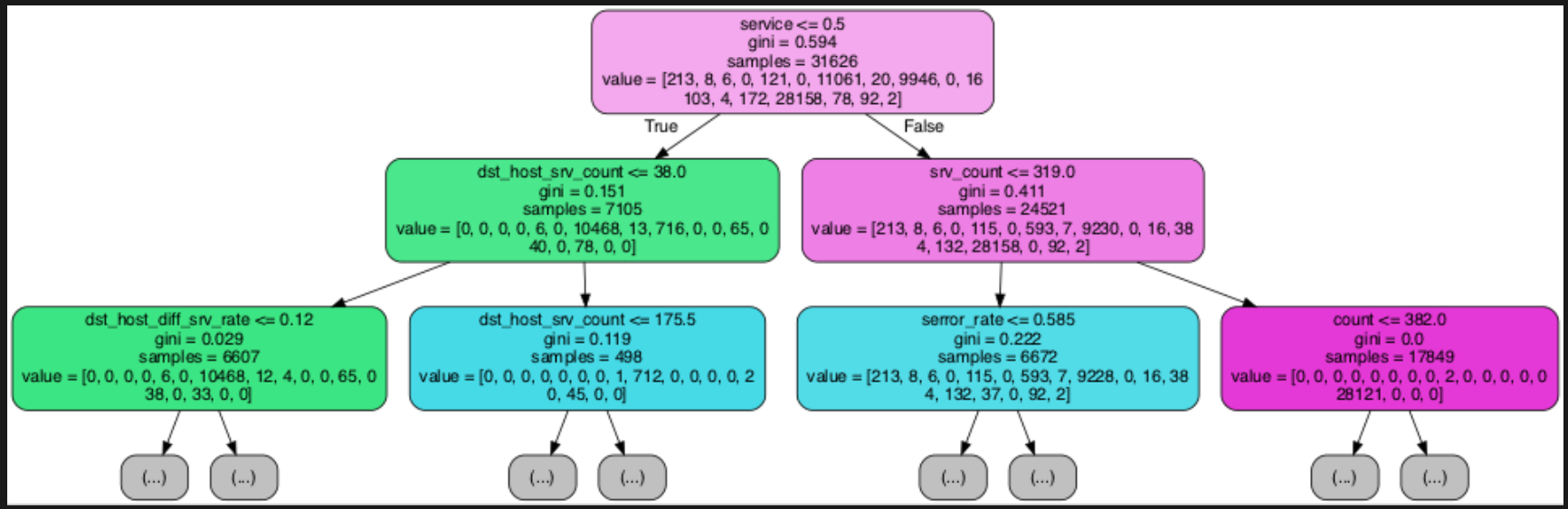
Random Forest Extract single trees

```
estimator5 = clf.estimators_[5]

dot_data = StringIO()
export_graphviz(estimator5, out_file=dot_data,max_depth=2,
                feature_names=X_train.columns.values,
                filled=True, rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Random Forest Feature Trees



Random Forest Feature Trees