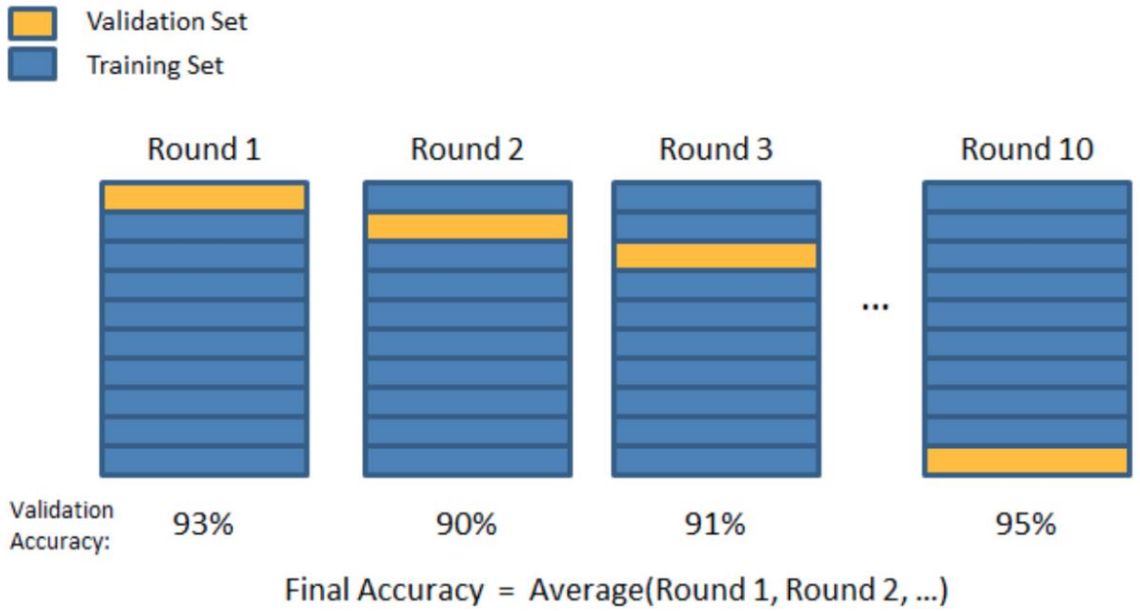
Practical Course DS: Advanced Algorithms – Part I

Dr. David Koll

For Completeness: CrossValidation (CV)

- **CrossValidation** is extremely useful to evaluate model improvement
  - **k-fold** cross validation creates **k** different folds of data
  - Each fold is of size n
  - In each fold, a different portion of size n/k the training set is used as validation set, remainder of data as training set
  - Can pre-validate the model with training set before feeding to test set
  - Question: Most extreme CV?



For Completeness: CrossValidation (CV)

- **CrossValidation** benefits?
  - vs validation set: reduces risk of overfitting a single validation set
  - vs validating with test set: also don‘t need to touch test set too often

- Drawbacks?
  - Computationally expensive (linear complexity increase with factor k)
  - Consider dependent/grouped data: need to be careful about fold creation
    - E.g., time series analysis: do not produce folds at the wrong cut-offs
      - May employ TimeSeriesSplit for this
    - Also an issue in classification: need to make sure that fold sizes are large enough in terms of absolute number of class members

CrossValidation in SKLearn

- cross_val_score function in metrics.model_selection

- Procedure:
  - Import cross_val_score from metrics.model_selection
  - Create classifier/regressor object **clf**
  - Then, instead of using **clf.fit** use:

```
cross_val_score(clf, train[features], train[target], cv=k, scoring=score_function)
```

  - Returns an array of length k, containing the validation error for each of the k folds
  - You can average the scores of each fold to determine the CV error

CrossValidation in SKLearn

- Scoring can be any metric

- Why neg_* in Regression?
  - SKLearn internal implementation
    - Optimization to maximum
    - Scores that need to be minimized are thus negated
  - Simply treat it as the absolute value, the closer to zero, the better

Scoring	Function
Classification	
'accuracy'	metrics.accuracy_score
'average_precision'	metrics.average_precision_score
'f1'	metrics.f1_score
'f1_micro'	metrics.f1_score
'f1_macro'	metrics.f1_score
'f1_weighted'	metrics.f1_score
'f1_samples'	metrics.f1_score
'neg_log_loss'	metrics.log_loss
'precision' etc.	metrics.precision_score
'recall' etc.	metrics.recall_score
'roc_auc'	metrics.roc_auc_score
Clustering	
'adjusted_mutual_info_score'	metrics.adjusted_mutual_info_score
'adjusted_rand_score'	metrics.adjusted_rand_score
'completeness_score'	metrics.completeness_score
'fowlkes_mallows_score'	metrics.fowlkes_mallows_score
'homogeneity_score'	metrics.homogeneity_score
'mutual_info_score'	metrics.mutual_info_score
'normalized_mutual_info_score'	metrics.normalized_mutual_info_score
'v_measure_score'	metrics.v_measure_score
Regression	
'explained_variance'	metrics.explained_variance_score
'neg_mean_absolute_error'	metrics.mean_absolute_error
'neg_mean_squared_error'	metrics.mean_squared_error
'neg_mean_squared_log_error'	metrics.mean_squared_log_error
'neg_median_absolute_error'	metrics.median_absolute_error
'r2'	metrics.r2_score

CrossValidation in SKLearn

In [22]:
```
white_val_cv_score = cross_val_score(rf_white, train_white[features_white], train_white['quality'],
                                     scoring='neg_mean_absolute_error', cv=5)
white_all_cv_score = cross_val_score(rf_white_all, train_white_all[features_white], train_white_all['quality'],
                                     scoring='neg_mean_absolute_error', cv=5)

print('white Val CV', white_val_cv_score)
print('white All CV', white_all_cv_score)
print('Improvement: ', -white_val_cv_score + white_all_cv_score)
```

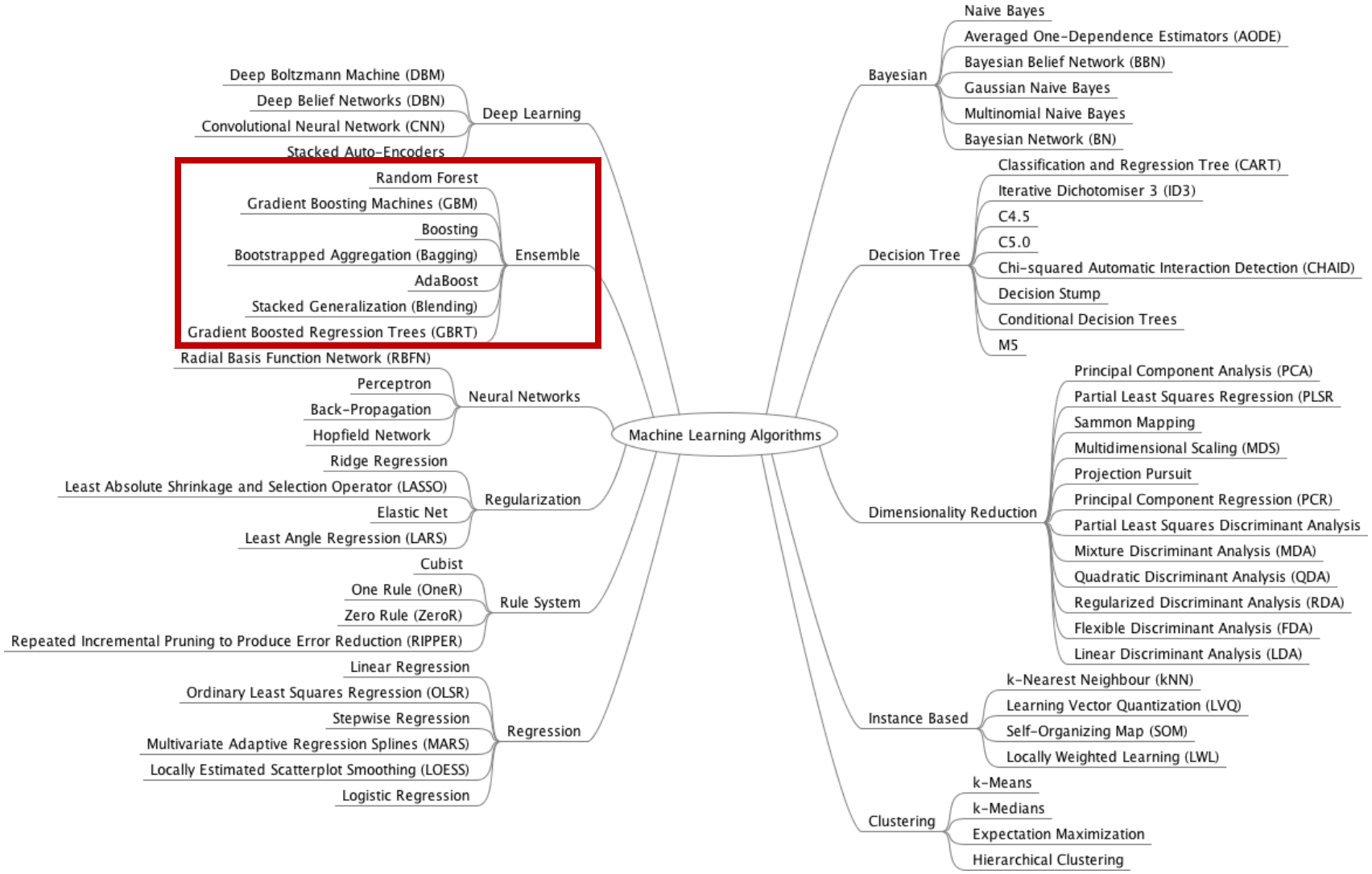
```
C:\Users\David\Anaconda2\lib\site-packages\sklearn\model_selection\_split.py:581: Warning: The least populated class in y has o
nly 3 members, which is too few. The minimum number of groups for any class cannot be less than n_splits=5.
  % (min_groups, self.n_splits)), Warning)
C:\Users\David\Anaconda2\lib\site-packages\sklearn\model_selection\_split.py:581: Warning: The least populated class in y has o
nly 4 members, which is too few. The minimum number of groups for any class cannot be less than n_splits=5.
  % (min_groups, self.n_splits)), Warning)
```

```
('white Val CV', array([-0.41204437, -0.42539683, -0.39745628, -0.41373802, -0.38338658]))
('white All CV', array([-0.39163498, -0.38529785, -0.38656527, -0.38040712, -0.38491049]))
('Improvement: ', array([ 0.02040939,  0.04009898,  0.01089101,  0.03333089, -0.0015239 ]))
```

CrossValidation in SKLearn

- A few comments:
  - For unbalanced data: use StratifiedKFold
    - What does it ensure?
  - For grouped data: use GroupKFold
    - Ensures that data belonging to the same group is not both in train and test set
    - Helps to avoid overfitting
  - Sometimes it may be a good idea to shuffle data before CV
    - E.g., if you have an ordered train set (regarding the output)
    - Why?

ML Algorithms



Ensemble Learning

"The idea of ensemble methodology is to build a predictive model by integrating multiple models. It is well-known that ensemble methods can be used for improving prediction performance." [1]

[1] Rokach, L. (2010). "Ensemble-based classifiers". *Artificial Intelligence Review*. **33** (1-2): 1–39

Ensemble Performance

- Ensemble methods have been hugely successful in DS competitions
  - Dominant methods often get superseded after some time
  - Still, dominated methods are useful (e.g., ensemble of ensembles)

- In practice, simple ensembles or even single models may be preferred
  - Application may require fast training and fast predictions (e.g., streamed data)
  - Overly complex ensembles not suitable for a wide range of tasks
    - E.g., several hours or even days training time is too much in many cases

- Trade-off between edge over competitors and performance

Ensemble Learning – Naïve Approach

- Scenario1 : predict number of bike rides at hour X

- Naïve approach:
  - Build multiple models
  - Use the same features
  - Use different algorithms

- Example 1:
  - Model 1: Linear regression; Model 2: kNN regression
  - Ensemble: Average results of both to predict bike rides
    - Good idea?

Ensemble Learning – Naïve Approach

- Scenario 2: binary classification

- Naïve approach:
  - Build multiple models
  - Use the same features
  - Use different algorithms

This solution is often not called ensemble, but multiple classifier system!

- Example 2:
  - Model 1: Decision Tree classifier; Model 2: Logistic Regression classifier; Model 3: kNN classifier
  - Ensemble: Take majority vote
  - Intuition: Get better results on uncertain samples, thereby improve performance

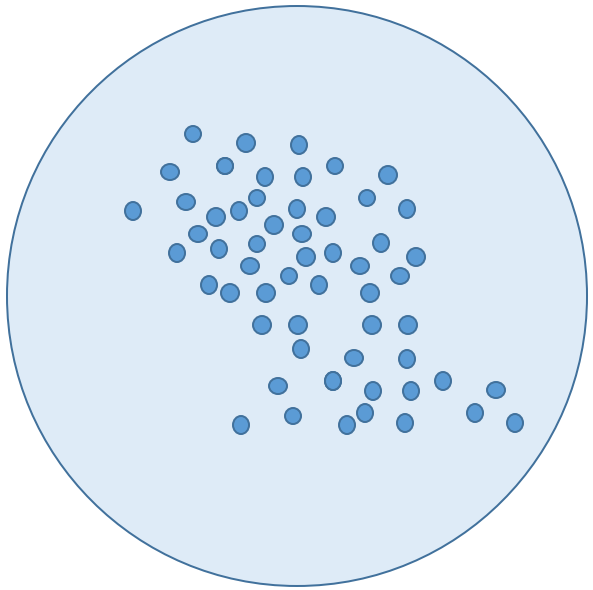
Ensemble Learning - Algorithms

- Key rationale: many weak learners are better than one strong learner
  - All learners are of the same class (e.g., decision trees)
  - Learners should be diverse (e.g., choose different data samples)
    - Can help to avoid overfitting
  - Diminishing returns: at some point, adding more learners does not yield further improvement

- The remaining lectures cover three styles of ensemble learning:
  - Bagging (in particular Random Forest)
  - Boosting (in particular Gradient Boosting)
  - Stacking/Blending

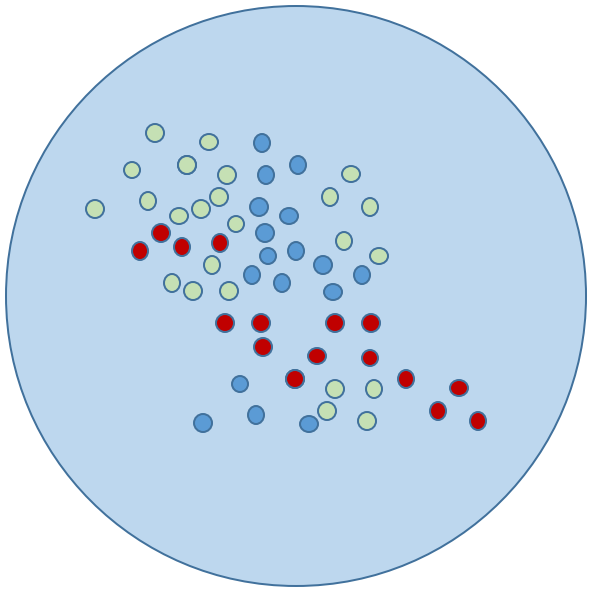
Ensembles: Bagging vs Boosting

1 learner



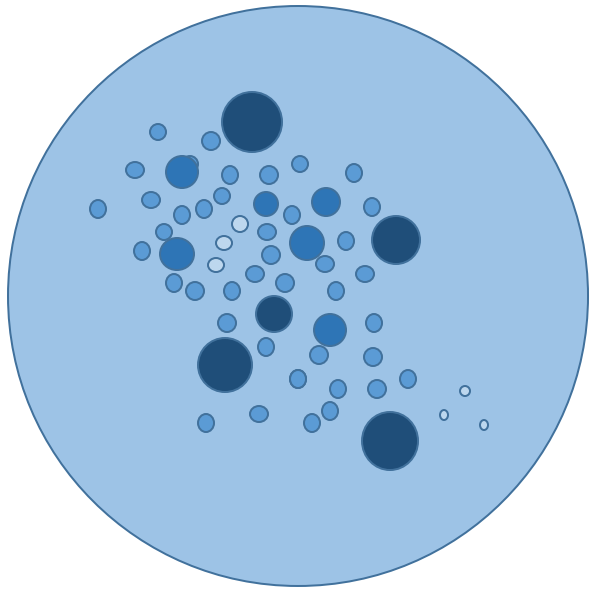
Single Model

n learners



Bagging

n learners

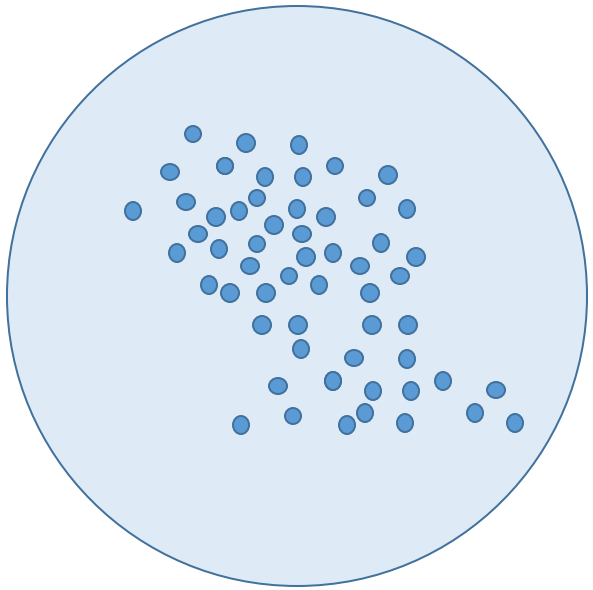


Boosting

Bagging re-samples data for each learner,
boosting *iteratively* weighs data

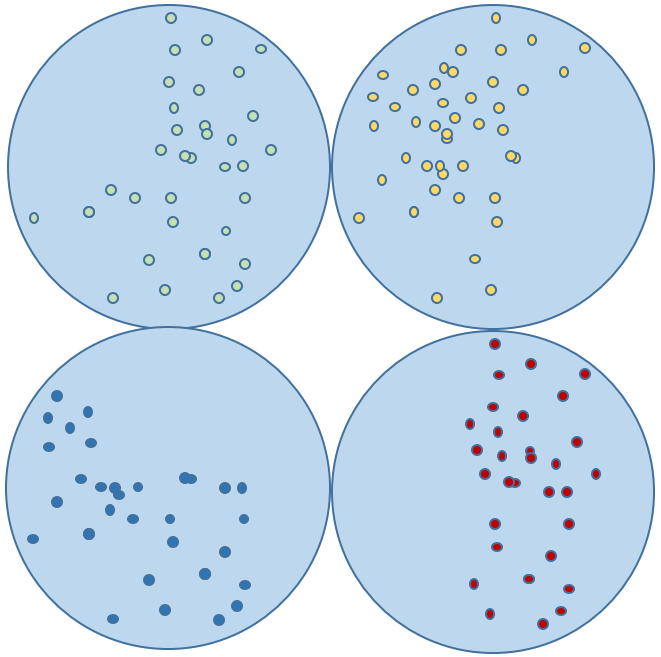
Ensembles: Bagging vs Boosting

1 learner



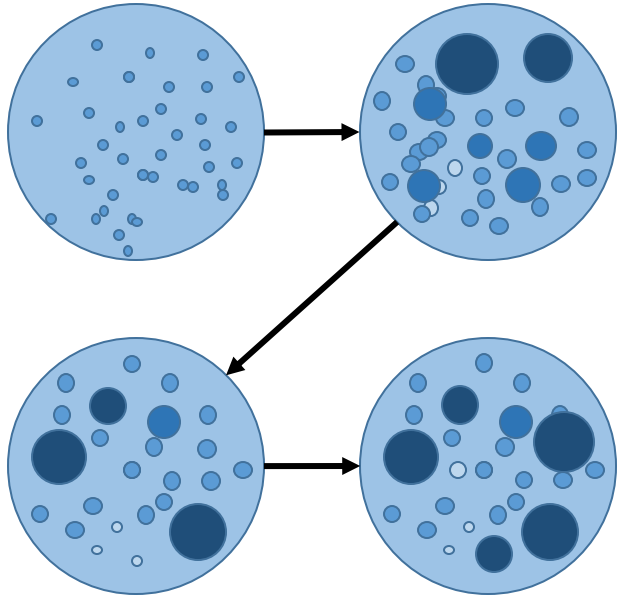
Single Model

n learners



Bagging

n learners



Boosting

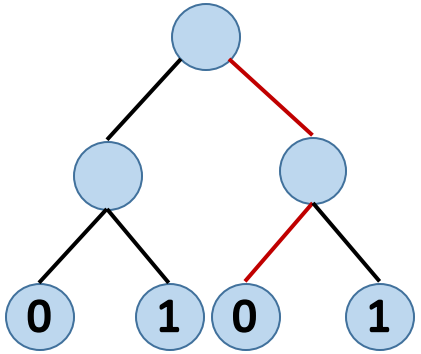
Bagging re-samples data for each learner,
boosting *iteratively* weighs data

Bagging vs Boosting

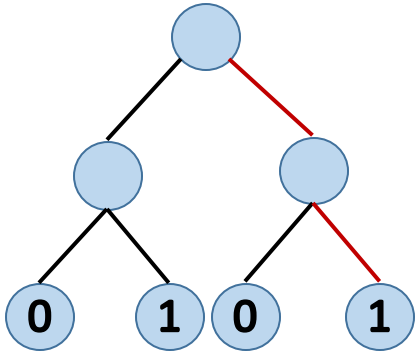
- Core differences:
  - Bagging is a parallel operation, boosting is iterative
    - Bagging can be parallelized easily
    - What about boosting? Bagging or Boosting faster?
  - Boosting may introduce additional overfitting, bagging reduces it
    - Boosting can overfit as it focuses on the set of samples it underperformed in previous iteration (at the same time, reduces bias)
    - Bagging re-samples randomly, avoiding overfitting (but will not reduce bias)
  - Bagging takes simple average, boosting takes weighted average of learners
  - Performance depends on data and task
- Best to know both!

Bagging: RandomForest

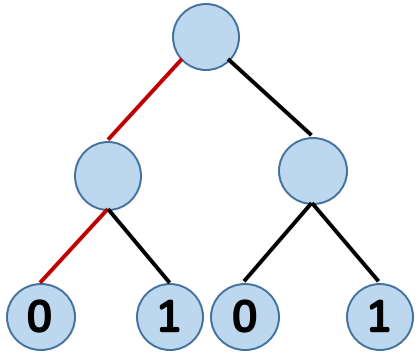
- A **RandomForest** grows ***n*** **DecisionTrees** and takes their majority vote



Prediction: 0



Prediction: 1

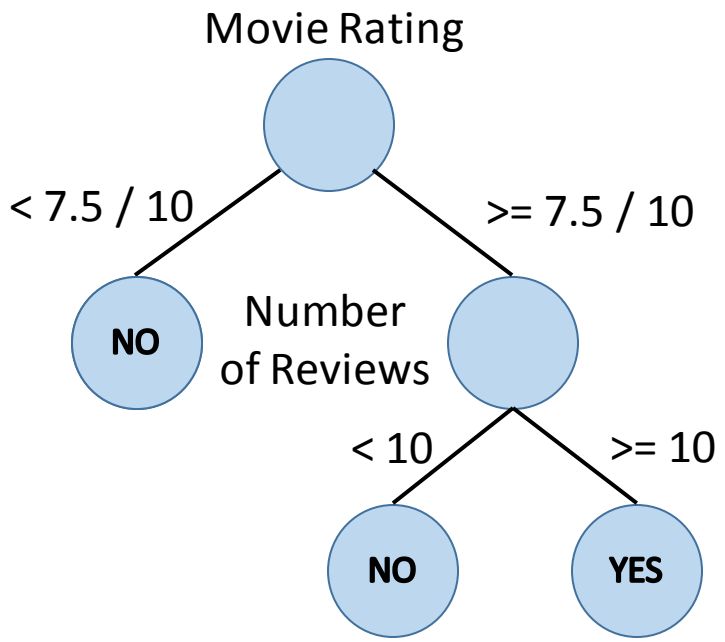


Prediction: 0

This (very basic) RandomForest predicts 0

Recap: Decision / Regression Trees

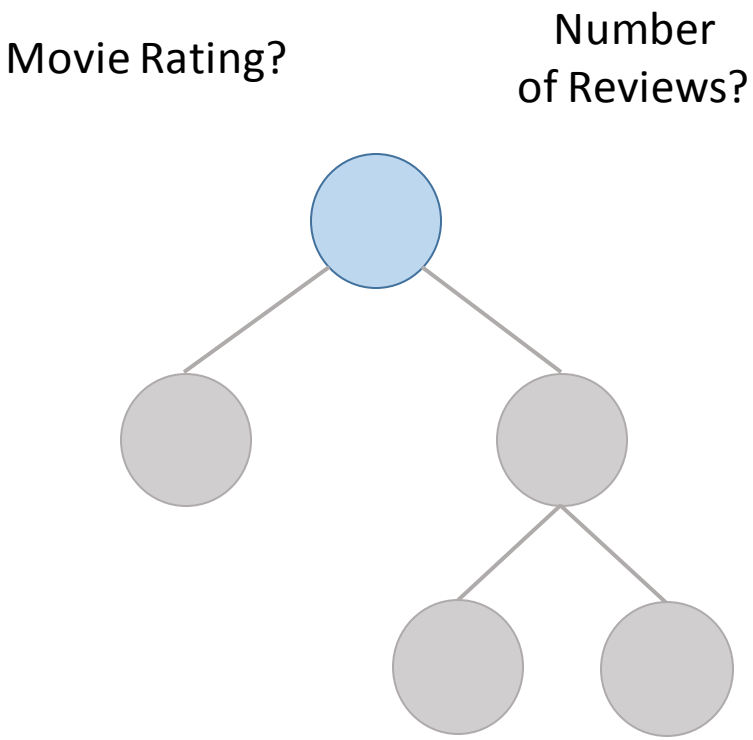
- The leafs of a **DecisionTree** show the estimated class/value of a datapoint
- Arrive at the leaf by evaluating decision nodes top-down from root node



Movie Rating	# of Reviews	Recommend
8	100	Yes
9	1000	Yes
7	200	No
5	10	No
10	5	No
8	250	Yes
3	600	No
5	150	No
10	10000	Yes

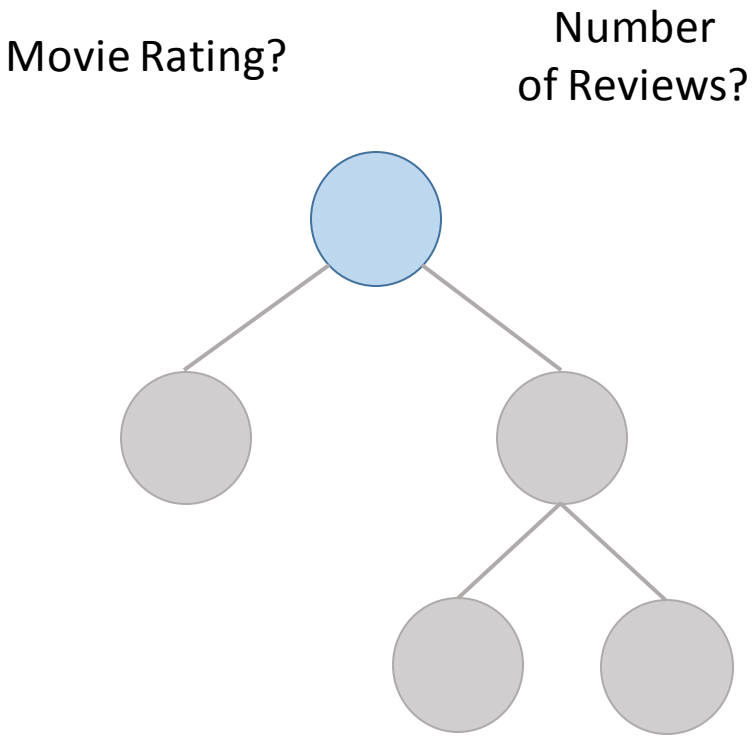
How are DecisionTrees learned?

- Finding best tree is an NP-hard problem, hence need approximation
- **What do you know about DecisionTree construction?**



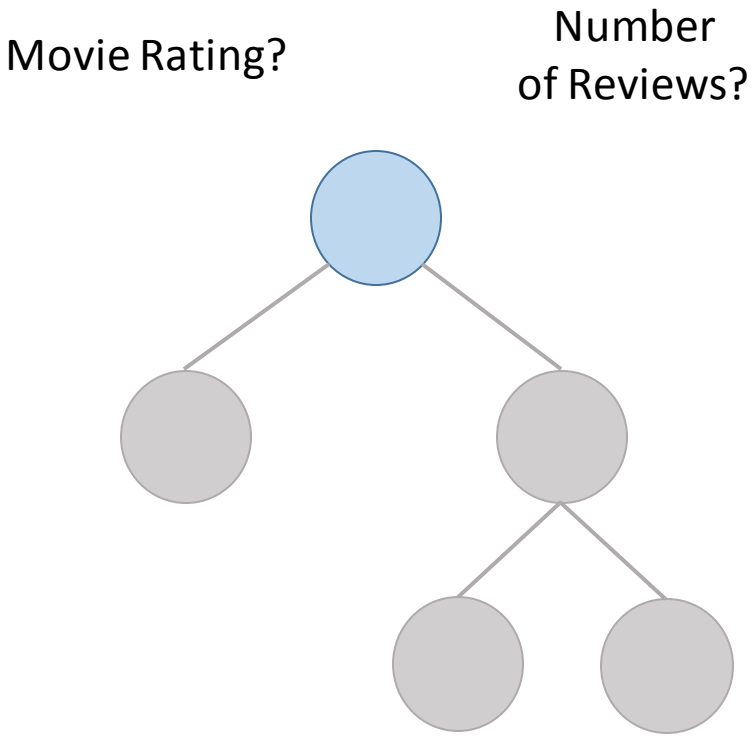
How are DecisionTrees learned?

- Recall: Nodes are determined based on feature importance
  - **Do you remember which metrics are used?**



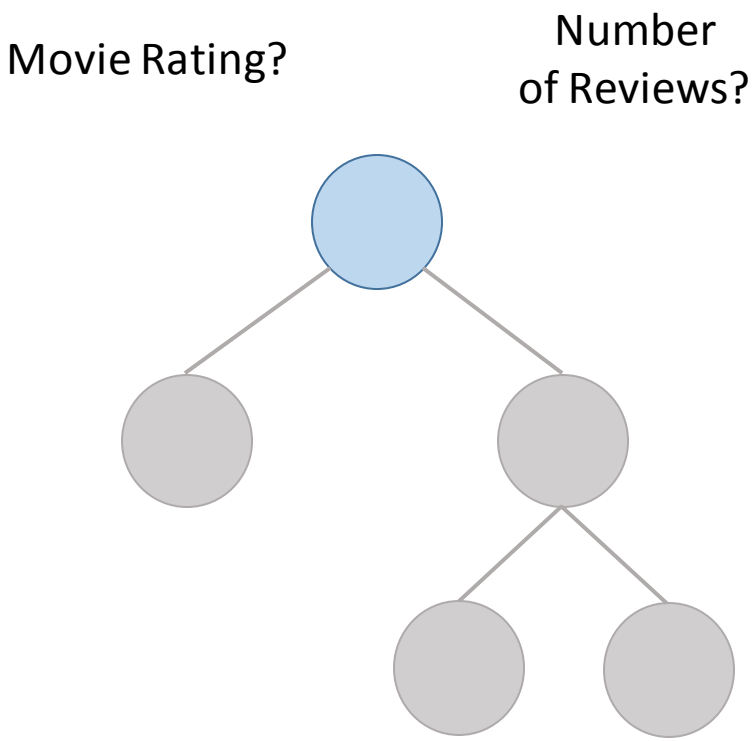
How are DecisionTrees learned?

- Most prominent metric: information gain (IG) based on entropy
  - **Select feature with highest IG as root, then recurse**



How are DecisionTrees learned?

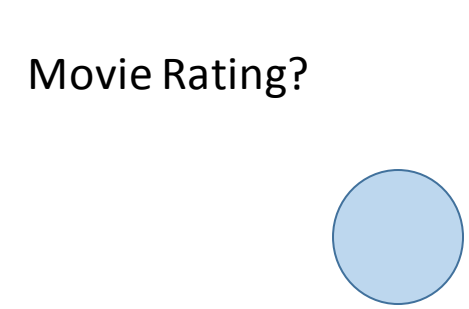
- Most prominent metric: information gain (IG) based on entropy
  - **Select feature with highest IG as root, then recurse**



Movie Rating	# of Reviews	Recommend
8	100	Yes
9	1000	Yes
7	200	No
5	10	No
10	5	No
8	250	Yes
3	600	No
5	150	No
10	10000	Yes

How are DecisionTrees learned?

- Most prominent metric: information gain (IG) based on Shannon entropy (H)
  - Entropy delivers information about
  - **Select feature with highest IG as root, then recurse**



H(Y = Movie Rating > 7.5) = ~0.722
H(Y = Number Reviews > 10) = ~0.971

IG(Movie Rating > 7.5) = H(Y) – H(Y|X_1) = 1 – 0.45 = **~0.55**
IG(Number Reviews > 10) = H(Y) – H(Y|X_1) = 1 – 0.69 = ~0.31

Movie Rating	# of Reviews	Recommend
8	100	Yes
9	1000	Yes
7	200	No
5	10	No
10	5	No
8	250	Yes
3	600	No
10	10000	Yes

Recall: Entropy: H(Y) = $-\sum_{i=1}^{n} p_i \log p_i$
Uncertainty of Data, here: purity of data

How are DecisionTrees learned?

- Most prominent metric: information gain (IG) based on Shannon entropy (H)
  - Entropy delivers information about
  - **Select feature with highest IG as root, then recurse**

H(Y = Movie Rating > 7.5) = ~0.722
H(Y = Number Reviews > 10) = ~0.971

IG(Movie Rating > 7.5) = H(Y) – H(Y|X_1) = 1 – 0.45 = **~0.55**
IG(Number Reviews > 10) = H(Y) – H(Y|X_1) = 1 – 0.69 = ~0.31

Movie Rating	# of Reviews	Recommend
8	100	Yes
9	1000	Yes
7	200	No
5	10	No
10	5	No
8	250	Yes
3	600	No
10	10000	Yes

Quick Example: Entropy and IG

- What is the Entropy H and Information Gain IG when splitting the following data on Movie Rating > 7.5?

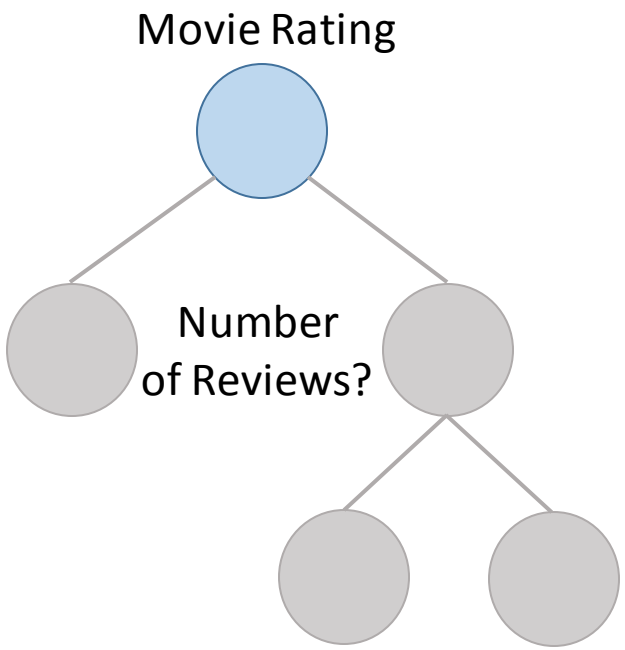
Movie Rating	# of Reviews	Recommend
8	100	Yes
9	1000	Yes
7	200	No
5	10	No

- Entropy: 0 (note that log(0) is usually undefined, but here treated as 0)
- Information Gain?
  - 1
- Can we also split elsewise with the same result?

Recall: Entropy: H(Y) = $-\sum_{i=1}^{n} p_i \log p_i$
Uncertainty of Data, here: purity of data

How are DecisionTrees learned?

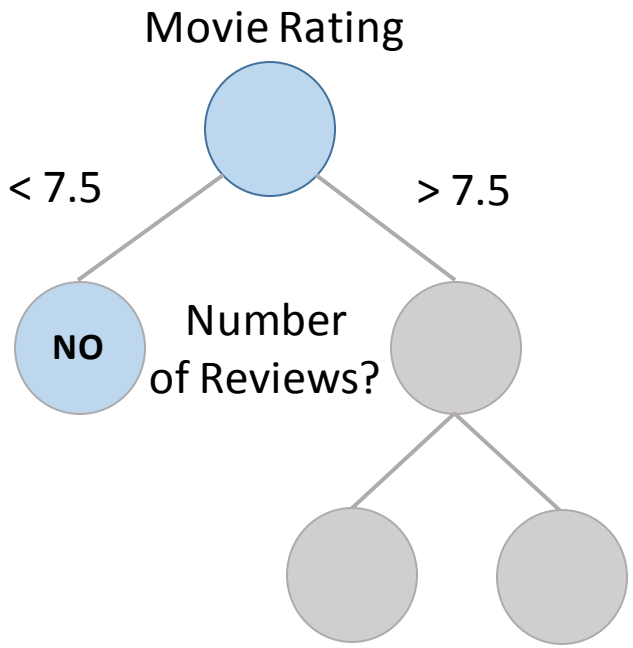
- Recursive splitting



Movie Rating	# of Reviews	Recommend
8	100	Yes
9	1000	Yes
7	200	No
5	10	No
10	5	No
8	250	Yes
3	600	No
10	10000	Yes

How are DecisionTrees learned?

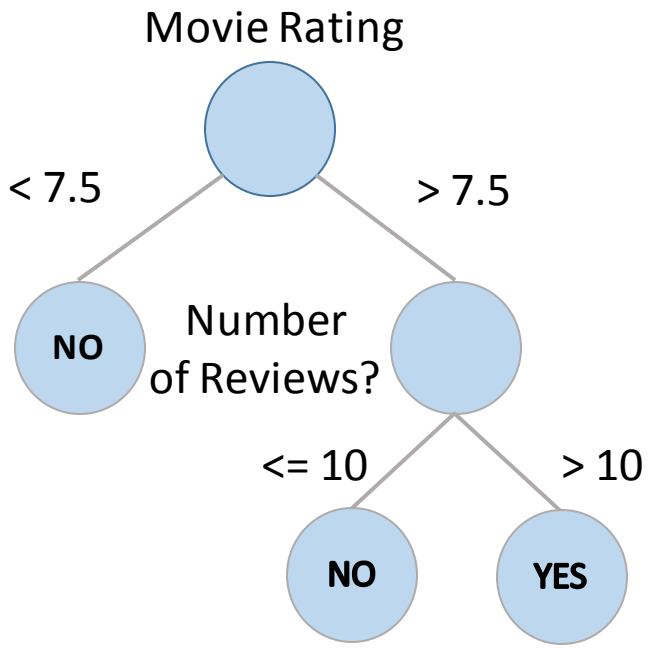
- Recursive splitting
  - **Why don‘t we split in the left sub-tree?**



Movie Rating	# of Reviews	Recommend
8	100	Yes
9	1000	Yes
7	200	No
5	10	No
10	5	No
8	250	Yes
3	600	No
10	10000	Yes

How are DecisionTrees learned?

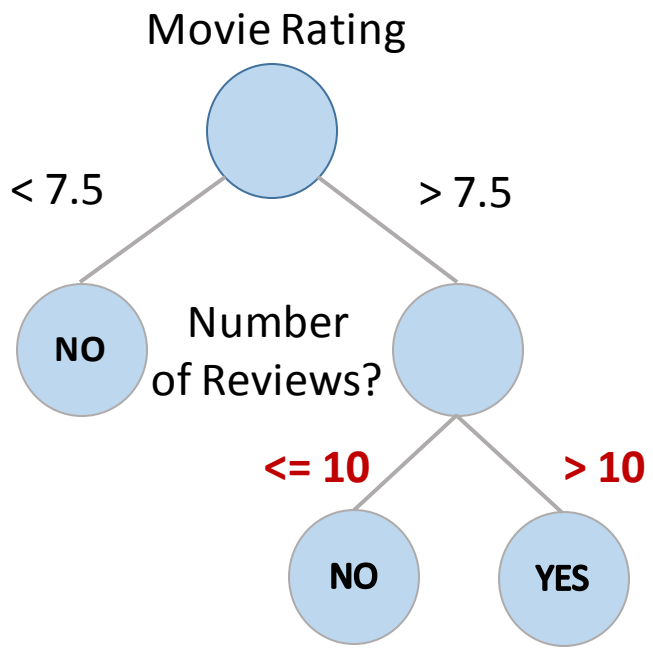
- Recursive splitting
  - **Why don‘t we split in the left sub-tree?**



Movie Rating	# of Reviews	Recommend
8	100	Yes
9	1000	Yes
7	200	No
5	10	No
10	5	No
8	250	Yes
3	600	No
10	10000	Yes

How are DecisionTrees learned?

- Infinite number of possible split values
  - **How to determine split values?**



Movie Rating	# of Reviews	Recommend
8	100	Yes
9	1000	Yes
7	200	No
5	10	No
10	5	No
8	250	Yes
3	600	No
10	10000	Yes

How are DecisionTrees learned?

- Infinite number of possible split values
  - **How to determine split values?**

- **One branch each numeric value?**
- Typical approach:
  - Test IG(Y | X:t), where X:t denotes testing a threshold t for information gain
  - How to limit t?
    - Pick only one value in between two datapoints
    - E.g., movie rating has only 6 different values
    - Sort values
    - Only test IG when label changes

Movie Rating	# of Reviews	Recommend
8	100	Yes
9	1000	Yes
7	200	No
5	10	No
10	5	No
8	250	Yes
3	600	No
10	10000	Yes

DecisionTrees for Regression

- For continous target value:
  - Train regression model based on single feature
  - Select feature with lowest error (sum of squares) as root
  - Recurse

- Predictions: mean of values in leaf
  - Leaf-size 1 = most accurate predictions?

Advantages of DecisionTrees

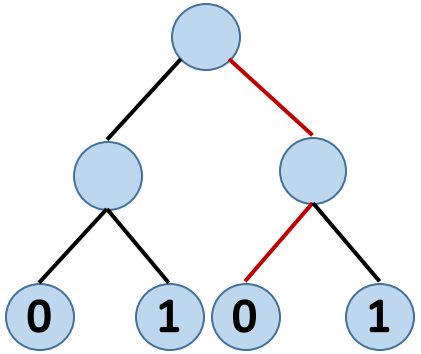
- Applicable to a wide range of problems (classification + regression)
- Humanly readable and interpretable
- Can handle categorical variables
- No formal assumptions on variable distributions
- Simple, easily implementable approach
- Low computational cost

Issues with Decision Trees?

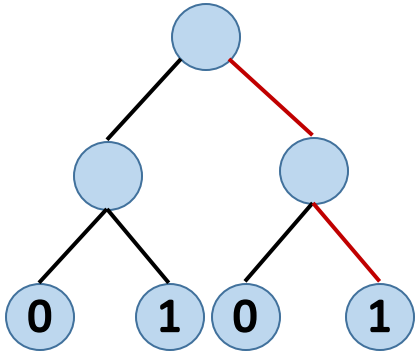
- Overfitting
  - Pruning,
  - Maximum depth,
  - Min data points in leafs,
  - …
- In general, trees are highly sensitive to input given to them
  - High Variance, Low Bias

Bagging

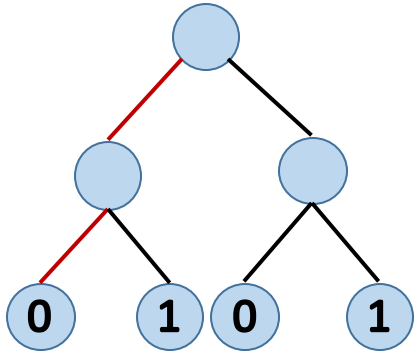
- A Bagging approach grows ***n*** **DecisionTrees** and takes their majority vote



Prediction: 0



Prediction: 1

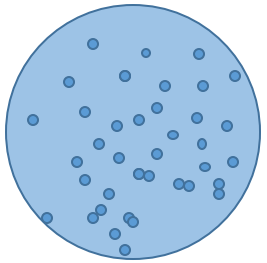


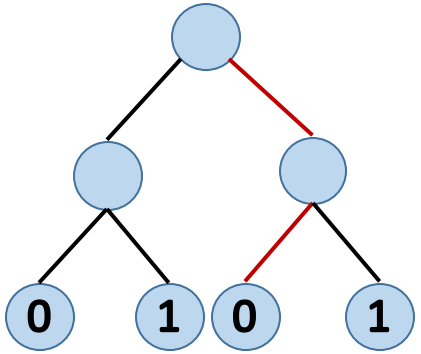
Prediction: 0

This (very basic) Bagging ensemble predicts 0

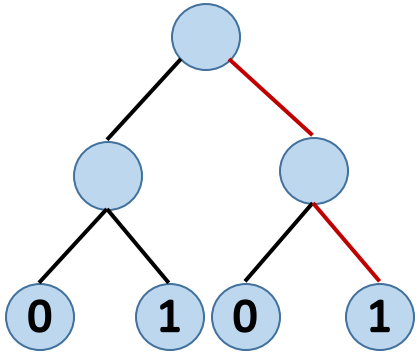
Bagging

- Bagging employs **bootstrap sampling:**
  - Random sampling with replacement!

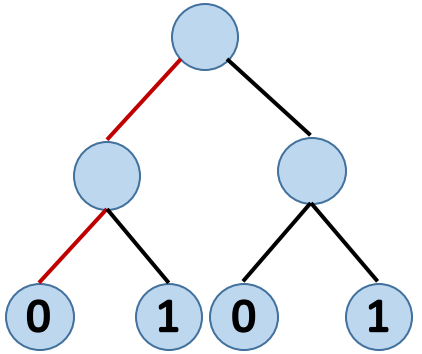




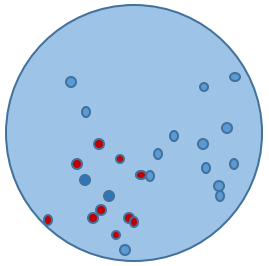
Prediction: 0

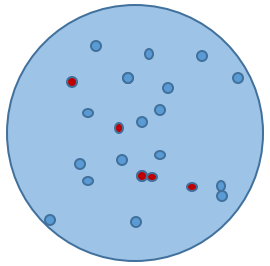


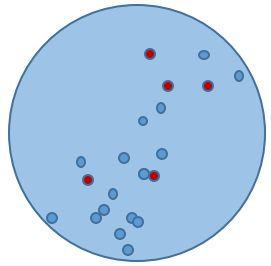
Prediction: 1



Prediction: 0

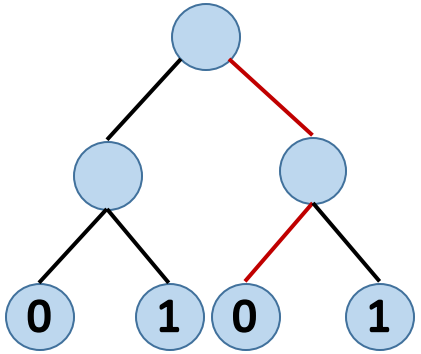




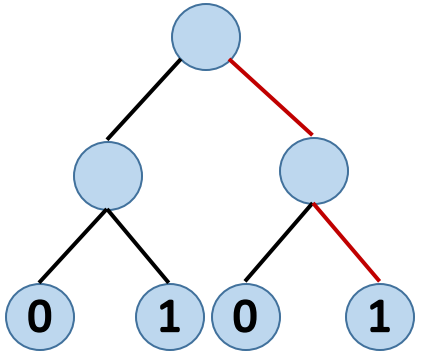


Bagging

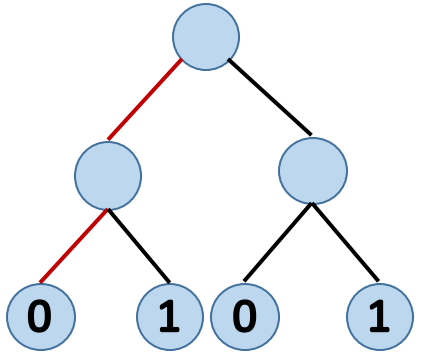
- Bagging typically outperforms a single tree because of random sampling
  - Decrease the variance, while not (significantly) increasing bias (bias-variance tradeoff)



Prediction: 0



Prediction: 1

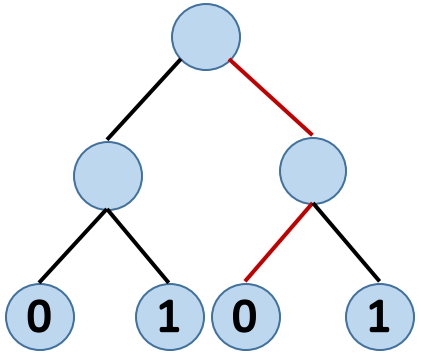


Prediction: 0

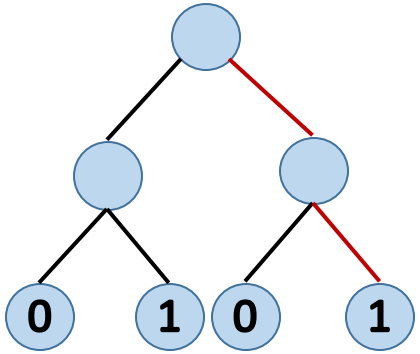
This (very basic) Bagging ensemble predicts 0

Bagging

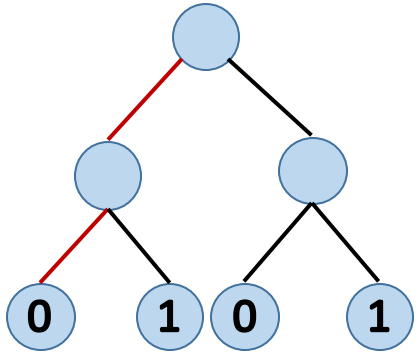
- Issue with Bagging: trees are typically (highly) correlated
  - In other words: they are very similar to each other



Prediction: 0



Prediction: 1

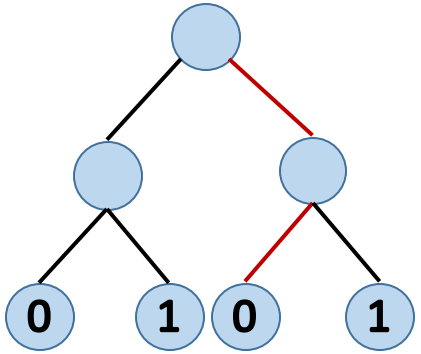


Prediction: 0

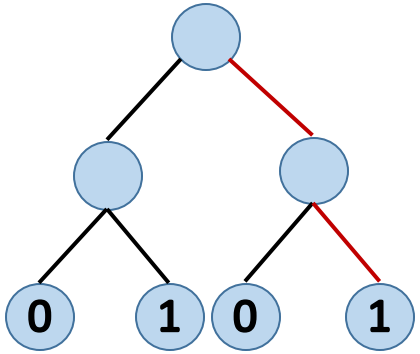
This (very basic) Bagging ensemble predicts 0

Bagging: RandomForest

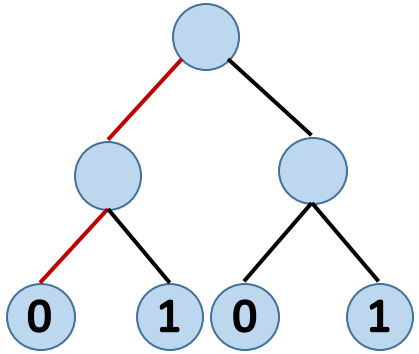
- A **RandomForest** extends the Bagging idea
- **Key point: Decorrelation of trees**



Prediction: 0



Prediction: 1



Prediction: 0

This (very basic) RandomForest predicts 0

RandomForest: Weak Learners

- How can we decorrelate trees?

```
In [93]: hour_data.head()
```

Out[93]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

RandomForest: Weak Learners

- Data is a n-by-m matrix: can sample in either dimension

```
In [93]: hour_data.head()
```

Out[93]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

RandomForest: Weak Learners

- Which dimension would be preferred?
  - **RandomForest: BOTH!**
  - Take sample of rows (bootstrap sample) AND
  - **Subset of columns for every tree (also called „random subspace method“)**
  - Sampling and subset selection?

RandomForest: Algorithm [1]

- N = number of data points in data set, M = number of features

Define size of forest num_trees
for tree in num_trees:
    select **n < N** data points **with replacement**
    fully* train a **DecisionTree**:
        at each splitting node:
            consider **m < M random features**
            select feature with highest IG/lowest RSS…

Predict majority class (classification) or mean value (regression)

*no pruning, and to max_depth

[1] Breiman L., Random forests. In Machine Learning, pp. 5-32, 2001

RandomForest: Out-Of-Bag Error

- How to estimate performance?
- Option 1: Cross-validation
- Option 2: **Out-Of-Bag Error** (inherent RF feature):

- We do not use all training data for all trees (n < N)
- How can we use that for validation?
  - Each data point n* is not used in a number of trees num_trees* (1/e)
  - Let num_trees* predict class/value for n*
  - Take majority vote/mean value from num_trees* predictions
  - Calculate Out-Of-Bag Error (OOB) for these predictions.

RandomForest: Feature Importances

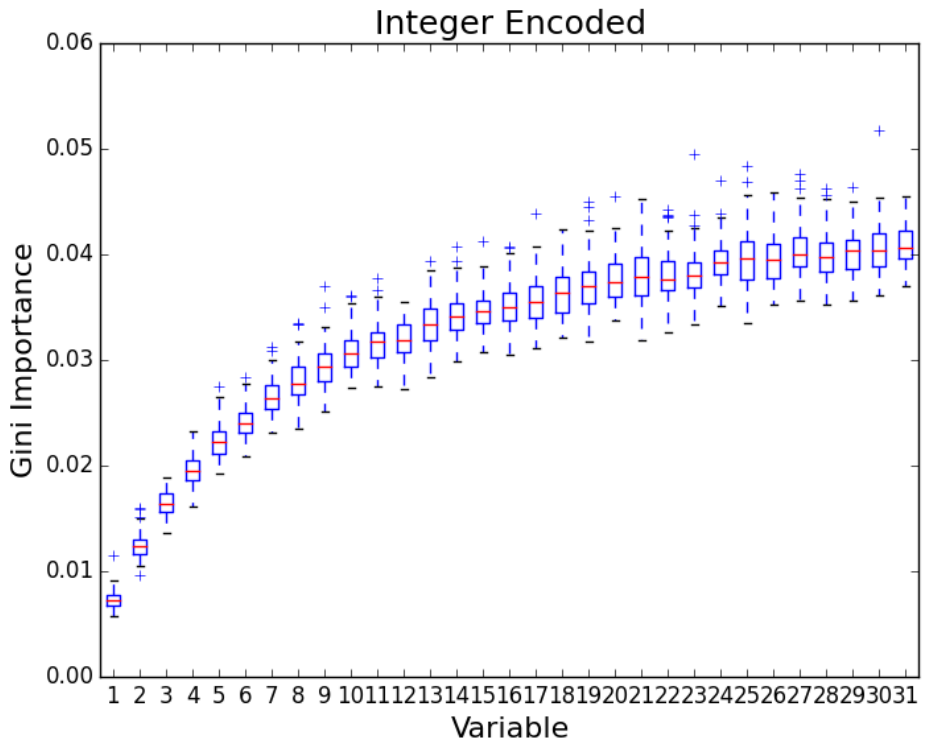
- What do you think constitutes an important feature in a RF?

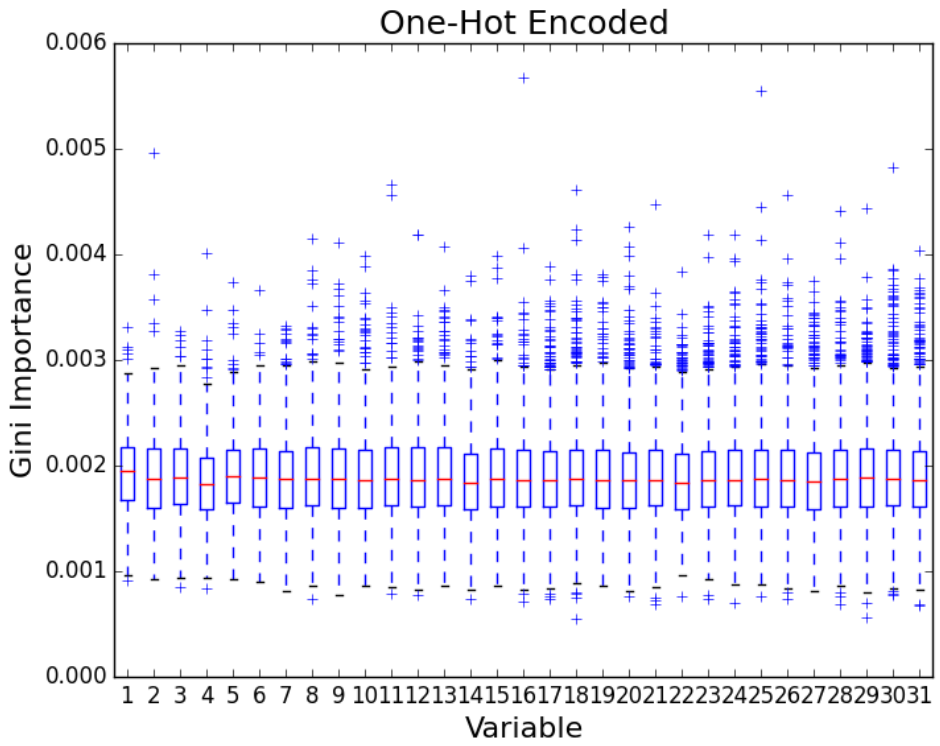
- Two measures:
  - Mean information gain (averaged over all trees in the forest)
  - Decrease in accuracy when permuting feature
    - Idea: if a feature is important to the prediction, then randomly permuting its values will decrease accuracy; if not, feature is not important

- One occasion where R and Python differ:
  - SciKit Learn does not offer permuted importance, equivalent R package does

RandomForest: Feature Importances

- Important: feature importances have shown to be
  - …biased towards high order categorical variables [1]





http://rnowling.github.io/machine/learning/2015/08/10/random-forest-bias.html

[1] Strobl et al: „Bias in random forest variable importance measures: Illustrations, sources and a solution“, BMC Bioinformatics 2007

RandomForest: Feature Importances

- Important: feature importances have shown to be
  - …not expressive for correlated variables [1]:

- If you have 2 correlated features:
  - As soon as one of them is picked as a split criterion, importance of other can decrease
  - Why?
    - Information is already gained from first variable/feature

[1] http://blog.datadive.net/selecting-good-features-part-iii-random-forests/

RandomForest: Some Questions!

- Is limiting max_depth of the trees the same as selecting equivalently small m?
- Can I have the same feature appear in different distances from the root?
- Are random forests still humanly readable?
- Do random forests overfit? Why?
- Is OOB validation the same as cross validataion?
- Why is feature importance more reliable in RF than in, e.g., LR?

RandomForest: Parameter Tuning!

- Ideas which parameters we should tune?
  - Obvious: number of trees
    - Increase in complexity by adding new trees?
  - Depth of each tree
  - Number of features used for each tree
  - Data points / samples used for each tree
  - Splitting criterion

RF Tuning in SKlearn

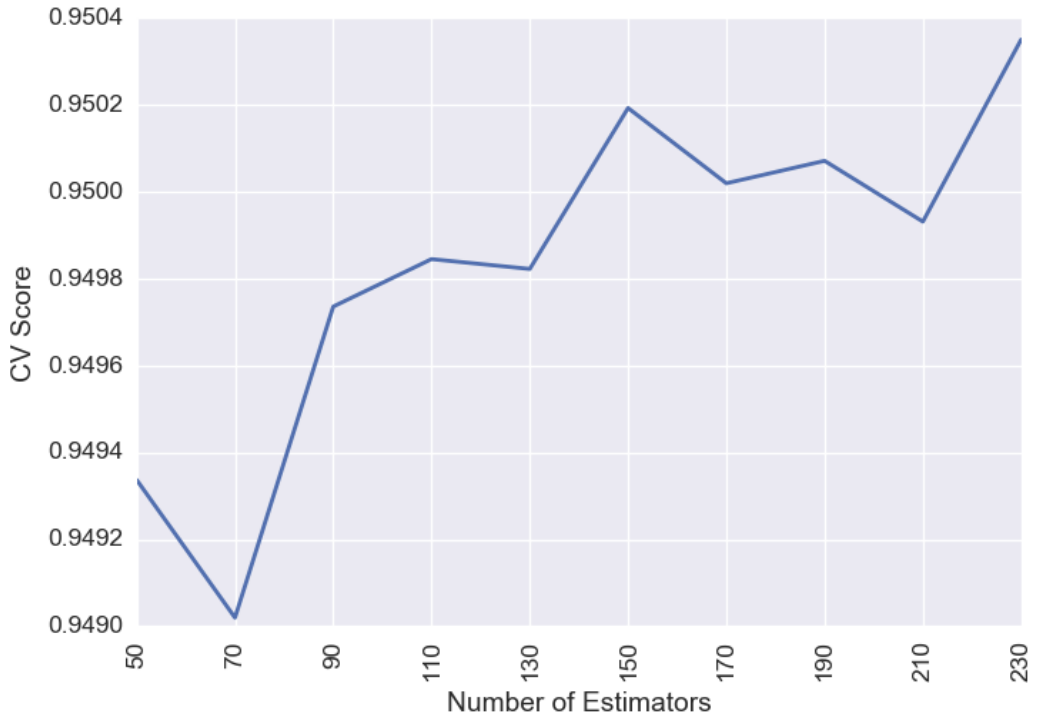
- Number of trees:
  - n_estimators (default = 10)
  - How to evaluate best setting?
  - First: how much data and many features does our dataset have?
    - The more features, the more data, the more trees we can and should use
    - Why?
  - Second: after initial guess, evaluate different values

RF Tuning in SKlearn

- Number of tr
  - n_estimatc
  - How to eva
  - First: how r
    - The more
    - Why?
  - Second: aft

In [120]:
```
from sklearn.model_selection import cross_val_score
cv_score = []
for i in np.arange(50, 250, 20):
    rfr = RandomForestRegressor(n_estimators=i)
    cv_score.append(np.mean(cross_val_score(rfr, train, target_train_registered, cv=5)))
```

In [121]:
```
plt.plot(cv_score)
plt.ylabel('CV Score')
plt.xlabel('Number of Estimators');
plt.xticks(np.arange(0,10,1), np.arange(50, 250, 20), rotation='vertical');
```

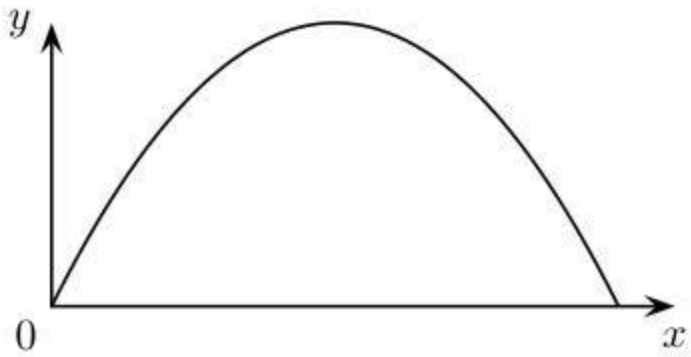


t have?

hould use

RF Tuning in SKlearn

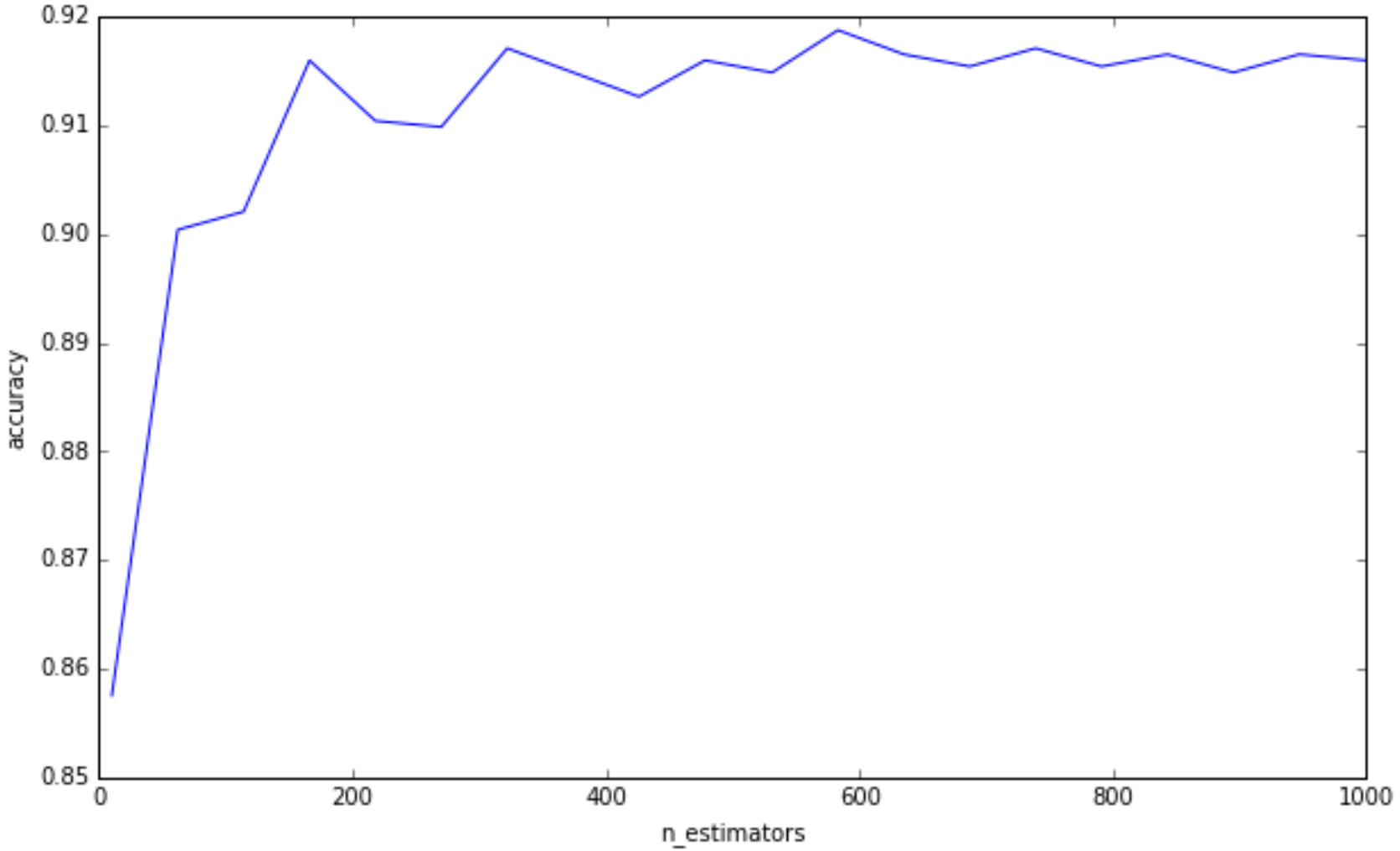
- Number of trees:
  - n_estimators (default = 10)
  - How to evaluate best setting?
  - First: how much data r dataset have?
    - The more features, t can and should use
    - Why?



  - Second: after initial guess, evaluate different values

RF Tuning in SKlearn

- Numb
  - n_e
  - Hov
  - Firs
    -
    -
  - Sec



RF Tuning in SKlearn

- Maximum depth of tree:
  - max_depth (default=None)
  - What does limiting max_depth help with?
  - Other ways to avoid trees being built until overfitting?
  - min_samples_split (default=2)
    - Controls the minimum number of samples in a leaf to split further
    - With bigger data, use larger min_samples_split (e.g., 20, 30, 40, 50)
    - Helps avoiding capturing noise in the data

RF Tuning in SKlearn

- Number of features used in each tree:
  - max_features (default=auto)
  - Any comment on: auto = sqrt(m) for classification, n_features for regression?
  - Alternatives: log2, None
  - More alternatives: Integer values or fractions

  - Typically: use auto or define a fraction
    - Optimal fraction depends on data, have to test multiple different values
    - Large number of correlated features typically dictates lower fraction used
      - Remember: we want to decorrelate trees!

  - Note: increasing max_features also increases training time!
    - With big data, have to find a tradeoff between optimality and runtime

RF Tuning in SKlearn

- Number of data points used for each tree:
  - Straightforward in BaggingClassifier: max_samples
  - In RandomForest: bootstrap=True|False (default=True)
  - Uses bootstrap sampling with sample size=n (any questions on this?)
  - We cannot directly tune sample size!
  - You can show mathematically that each data point will roughly appear 1 – 1/e trees, and not appear in 1/e trees (these are used for OOB error)
  - Note: R RF package offers sampSize as a tuneable parameter

RF Tuning in SKlearn

- Splitting criterion
  - Criterion=gini|entropy (default=gini), note: entropy = information gain
  - Criterion=mse|mae (default=mse)
  - Which criterion is used for which type of task?

- Gini vs entropy?

$$Gini: Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

$$Entropy: H(E) = -\sum_{j=1}^{c} p_j \log p_j$$

  - Both are pretty similar
  - Both have been shown to give the same result in 98% of the cases [1]
  - Computationally?

[1] Raileanu et al: “Theoretical comparison between the Gini Index and Information Gain criteria”, AMA 2004

RF Tuning in SKlearn

- Gini vs Entropy should be (one of) the last parameter(s) to explore
- Usually does not give any significant difference
- My own experience: with large datasets, it sometimes does matter, gini sometimes works a little bit better

RF Tuning in SKlearn

- Many more parameters: try out yourself!

min_samples_leaf : int, float, optional (default=1)

The minimum number of samples required to be at a leaf node:

- If int, then consider min_samples_leaf as the minimum number.
- If float, then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

Changed in version 0.18: Added float values for percentages.

min_weight_fraction_leaf : float, optional (default=0.)

The minimum weighted fraction of the sum total of weights (of all the input sample required to be at a leaf node. Samples have equal weight when sample_weight is provided.

max_leaf_nodes : int or None, optional (default=None)

Grow trees with `max_leaf_nodes` in best-first fashion. Best nodes are defined as reduction in impurity. If None then unlimited number of leaf nodes.

min_impurity_split : float,

Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.

Deprecated since version 0.19: `min_impurity_split` has been deprecated in favor of `min_impurity_decrease` in 0.19 and will be removed in 0.21. Use `min_impurity_decrease` instead.

min_impurity_decrease : float, optional (default=0.)

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

The weighted impurity decrease equation is the following:

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

where `N` is the total number of samples, `N_t` is the number of samples at the current node, `N_t_L` is the number of samples in the left child, and `N_t_R` is the number of samples in the right child.

`N`, `N_t`, `N_t_R` and `N_t_L` all refer to the weighted sum, if `sample_weight` is passed.

New in version 0.19.

Parameter Tuning in SKlearn

HELP! That‘s a lot of parameters to test!

Automated Parameter Tuning in SKlearn

- Luckily, SciKit has a way of automating this for you.
  - **GridSearchCV**

- Takes dictionary of parameters and values as input and computes the best parameter combinations by looping through all possible combinations

- Find optimal value for all parameters at once, from a large range of values each? Why? Why not?

Automated Parameter Tuning in SKlearn

In [30]:
```
from sklearn.model_selection import GridSearchCV
params = {'n_estimators': [140,145,150], 'max_features': [0.3,0.5,0.7,0.9,1]}
grid = GridSearchCV(rf_white_all, params, cv=3, scoring='neg_mean_absolute_error',verbose=1)
grid.fit(train_white[features_white], train_white['quality'])
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
```

```
[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:  1.2min finished
```

Out[30]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
            criterion='gini', max_depth=None, max_features=None,
            max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=145, n_jobs=1,
            oob_score=False, random_state=1, verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_estimators': [140, 145, 150], 'max_features': [0.3, 0.5, 0.7, 0.9, 1]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='neg_mean_absolute_error', verbose=1)
```

In [31]:
```
grid.best_estimator_
```

Out[31]:
```
RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
            criterion='gini', max_depth=None, max_features=1,
            max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=145, n_jobs=1,
            oob_score=False, random_state=1, verbose=0, warm_start=False)
```

Automated Parameter Tuning in SKlearn

In [32]:
```
from sklearn.model_selection import GridSearchCV
params = {'n_estimators': [140,145,150], 'max_features': [0.3,0.5,0.7,0.9,1], 'min_samples_split': [2, 5, 10, 20, 50]}
grid = GridSearchCV(rf_white_all, params, cv=3, scoring='neg_mean_absolute_error',verbose=1)
grid.fit(train_white[features_white], train_white['quality'])
```

```
Fitting 3 folds for each of 75 candidates, totalling 225 fits
```

```
[Parallel(n_jobs=1)]: Done 225 out of 225 | elapsed:  5.4min finished
```

Out[32]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
            criterion='gini', max_depth=None, max_features=None,
            max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=145, n_jobs=1,
            oob_score=False, random_state=1, verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_estimators': [140, 145, 150], 'max_features': [0.3, 0.5, 0.7, 0.9, 1], 'min_samples_split': [2, 5, 10, 2
0, 50]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='neg_mean_absolute_error', verbose=1)
```

In [33]:
```
grid.best_estimator_
```

Out[33]:
```
RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
            criterion='gini', max_depth=None, max_features=0.3,
            max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, n_estimators=145, n_jobs=1,
            oob_score=False, random_state=1, verbose=0, warm_start=False)
```

Automated Parameter Tuning in SKlearn

In [34]:
```
from sklearn.model_selection import GridSearchCV
params = {'n_estimators': [140,145,150], 'max_features': [0.3,0.5,0.7,0.9,1], 'min_samples_split': [2, 5, 10, 20, 50]}
grid = GridSearchCV(rf_white_all, params, cv=5, scoring='neg_mean_absolute_error',verbose=1)
grid.fit(train_white[features_white], train_white['quality'])
```

```
Fitting 5 folds for each of 75 candidates, totalling 375 fits
```

```
[Parallel(n_jobs=1)]: Done 375 out of 375 | elapsed: 10.7min finished
```

Out[34]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
            criterion='gini', max_depth=None, max_features=None,
            max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=145, n_jobs=1,
            oob_score=False, random_state=1, verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_estimators': [140, 145, 150], 'max_features': [0.3, 0.5, 0.7, 0.9, 1], 'min_samples_split': [2, 5, 10, 2
0, 50]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='neg_mean_absolute_error', verbose=1)
```

In [35]:
```
grid.best_estimator_
```

Out[35]:
```
RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
            criterion='gini', max_depth=None, max_features=0.3,
            max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=145, n_jobs=1,
            oob_score=False, random_state=1, verbose=0, warm_start=False)
```

Automated Parameter Tuning in SKlearn

In [42]:
```
from sklearn.model_selection import GridSearchCV
params = {'n_estimators': [140,145,150], 'max_features': [0.3,0.5,0.7, 0.9, 1.0]}
grid = GridSearchCV(rf_white_all, params, cv=3, scoring='neg_mean_absolute_error',verbose=1, n_jobs=4)
grid.fit(train_white[features_white], train_white['quality'])
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
```

```
[Parallel(n_jobs=4)]: Done  45 out of  45 | elapsed:   25.5s finished
```

Out[42]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
            criterion='gini', max_depth=None, max_features=None,
            max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=145, n_jobs=1,
            oob_score=False, random_state=1, verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=4,
       param_grid={'n_estimators': [140, 145, 150], 'max_features': [0.3, 0.5, 0.7, 0.9, 1.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='neg_mean_absolute_error', verbose=1)
```