W4995 Applied Machine Learning

Trees, Forests & Ensembles

02/18/19

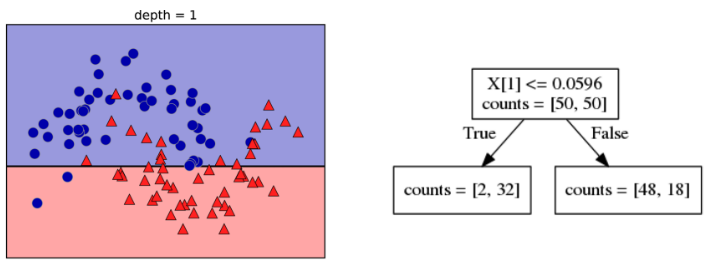
Andreas C. Müller

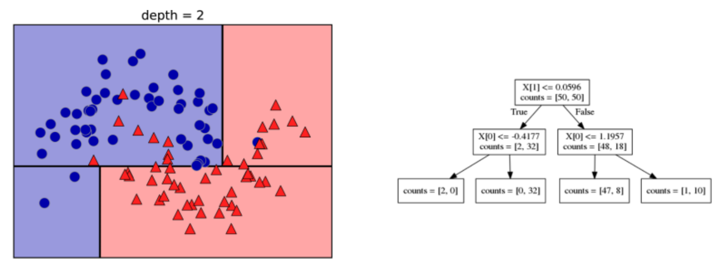
Why Trees?

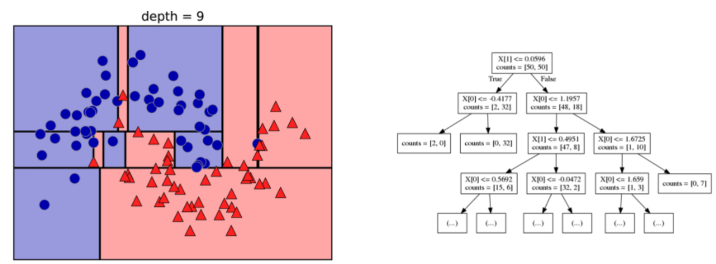
Decision Trees for Classification

Idea: series of binary questions

Building Trees







Continuous features:

- "questions" are thresholds on single features.
- Minimize impurity

Criteria (for classification)

- Gini Index:

$$H_{\text{gini}}(X_m) = \sum_{k \in \mathcal{Y}} p_{mk}(1 - p_{mk})$$

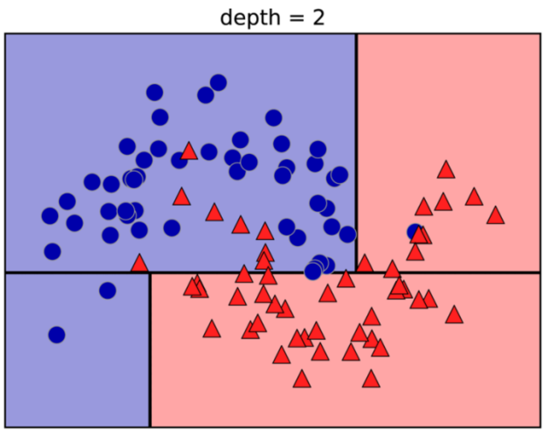
- Cross-Entropy:

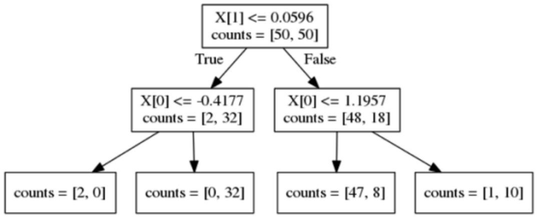
$$H_{\text{CE}}(X_m) = -\sum_{k \in \mathcal{Y}} p_{mk} \log(p_{mk})$$

X_m observations in node m
$\mathcal{Y}$ classes

$p_{m\cdot}$ distribution over classes in node m

Prediction


depth = 2


X[1] <= 0.0596
counts = [50, 50]
True
False
X[0] <= -0.4177
counts = [2, 32]
X[0] <= 1.1957
counts = [48, 18]
counts = [2, 0]
counts = [0, 32]
counts = [47, 8]
counts = [1, 10]

Regression trees

$$\text{Prediction: } \bar{y}_m = \frac{1}{N_m} \sum_{i \in N_m} y_i$$

Mean Squared Error:

$$H(X_m) = \frac{1}{N_m} \sum_{i \in N_m} (y_i - \bar{y}_m)^2$$

Mean Absolute Error:

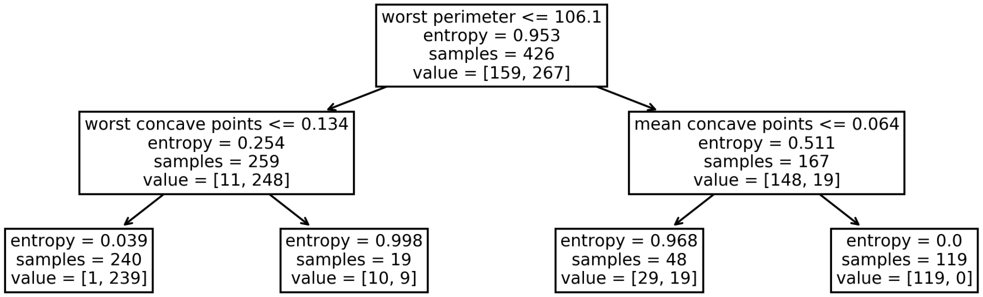
$$H(X_m) = \frac{1}{N} \sum |y_i - \bar{y}_m|$$

Visualizing trees with sklearn

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data,cancer.target,
                                                    stratify=cancer.target,
                                                    random_state=0)


from sklearn.tree import DecisionTreeClassifier, export_graphviz
tree = DecisionTreeClassifier(max_depth=2)
tree.fit(X_train, y_train)
```

Visualizing trees with sklearn

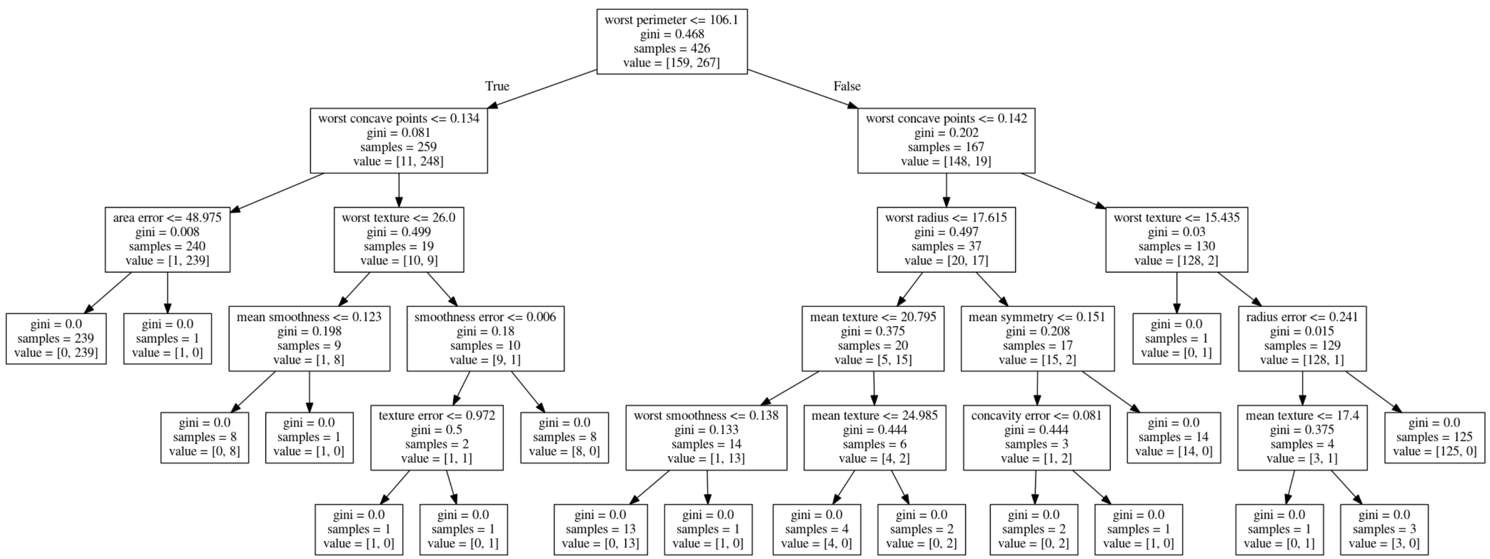
```
from sklearn.tree import plot_tree
tree_dot = plot_tree(tree, feature_names=cancer.feature_names)
```



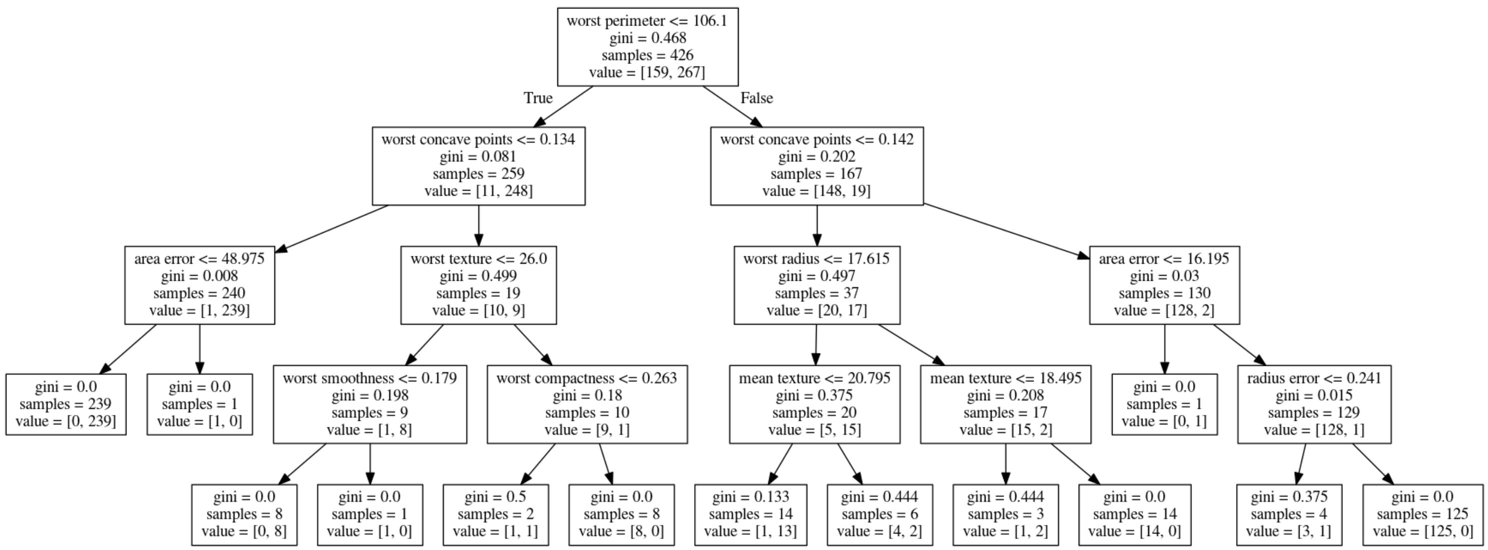
Parameter Tuning

- Pre-pruning and post-pruning (not in sklearn yet)
- Limit tree size (pick one, maybe two):
  - max_depth
  - max_leaf_nodes
  - min_samples_split
  - min_impurity_decrease

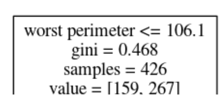
No pruning


worst perimeter <= 106.1
gini = 0.468
samples = 426
value = [159, 267]
True
False
worst concave points <= 0.134
gini = 0.081
samples = 259
value = [11, 248]
worst concave points <= 0.142
gini = 0.202
samples = 167
value = [148, 19]
area error <= 48.975
gini = 0.008
samples = 240
value = [1, 239]
worst texture <= 26.0
gini = 0.499
samples = 19
value = [10, 9]
worst radius <= 17.615
gini = 0.497
samples = 37
value = [20, 17]
worst texture <= 15.435
gini = 0.03
samples = 130
value = [128, 2]
gini = 0.0
samples = 239
value = [0, 239]
gini = 0.0
samples = 1
value = [1, 0]
mean smoothness <= 0.123
gini = 0.198
samples = 9
value = [1, 8]
smoothness error <= 0.006
gini = 0.18
samples = 10
value = [9, 1]
mean texture <= 20.795
gini = 0.375
samples = 20
value = [5, 15]
mean symmetry <= 0.151
gini = 0.208
samples = 17
value = [15, 2]
gini = 0.0
samples = 1
value = [0, 1]
radius error <= 0.241
gini = 0.015
samples = 129
value = [128, 1]
gini = 0.0
samples = 8
value = [0, 8]
gini = 0.0
samples = 1
value = [1, 0]
texture error <= 0.972
gini = 0.5
samples = 2
value = [1, 1]
gini = 0.0
samples = 8
value = [8, 0]
worst smoothness <= 0.138
gini = 0.133
samples = 14
value = [1, 13]
mean texture <= 24.985
gini = 0.444
samples = 6
value = [4, 2]
concavity error <= 0.081
gini = 0.444
samples = 3
value = [1, 2]
gini = 0.0
samples = 14
value = [14, 0]
mean texture <= 17.4
gini = 0.375
samples = 4
value = [3, 1]
gini = 0.0
samples = 125
value = [125, 0]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 13
value = [0, 13]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 4
value = [4, 0]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 3
value = [3, 0]

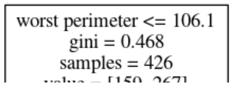
max_depth = 4


worst perimeter <= 106.1
gini = 0.468
samples = 426
value = [159, 267]
True
False
worst concave points <= 0.134
gini = 0.081
samples = 259
value = [11, 248]
worst concave points <= 0.142
gini = 0.202
samples = 167
value = [148, 19]
area error <= 48.975
gini = 0.008
samples = 240
value = [1, 239]
worst texture <= 26.0
gini = 0.499
samples = 19
value = [10, 9]
worst radius <= 17.615
gini = 0.497
samples = 37
value = [20, 17]
area error <= 16.195
gini = 0.03
samples = 130
value = [128, 2]
gini = 0.0
samples = 239
value = [0, 239]
gini = 0.0
samples = 1
value = [1, 0]
worst smoothness <= 0.179
gini = 0.198
samples = 9
value = [1, 8]
worst compactness <= 0.263
gini = 0.18
samples = 10
value = [9, 1]
mean texture <= 20.795
gini = 0.375
samples = 20
value = [5, 15]
mean texture <= 18.495
gini = 0.208
samples = 17
value = [15, 2]
gini = 0.0
samples = 1
value = [0, 1]
radius error <= 0.241
gini = 0.015
samples = 129
value = [128, 1]
gini = 0.0
samples = 8
value = [0, 8]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.5
samples = 2
value = [1, 1]
gini = 0.0
samples = 8
value = [8, 0]
gini = 0.133
samples = 14
value = [1, 13]
gini = 0.444
samples = 6
value = [4, 2]
gini = 0.444
samples = 3
value = [1, 2]
gini = 0.0
samples = 14
value = [14, 0]
gini = 0.375
samples = 4
value = [3, 1]
gini = 0.0
samples = 125
value = [125, 0]

max_leaf_nodes = 8



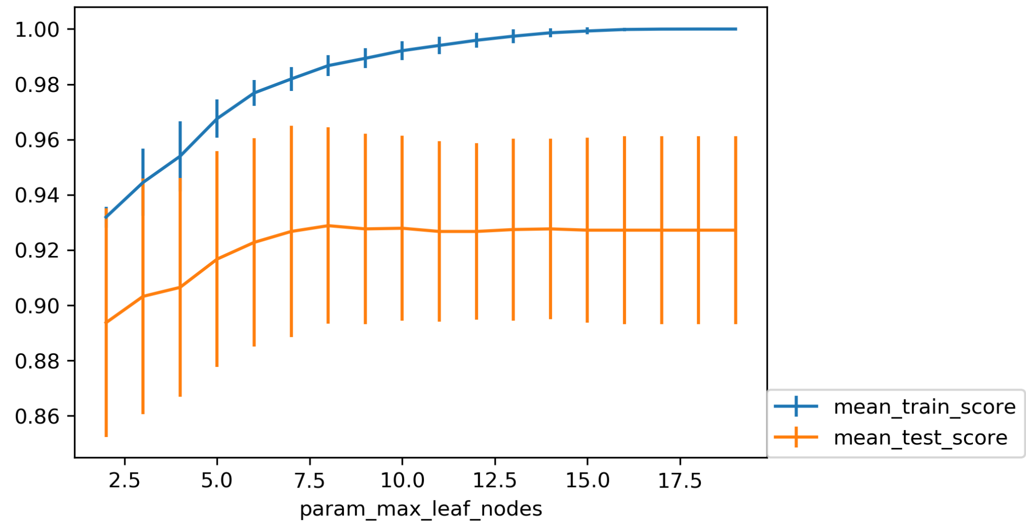
min_samples_split = 50



```
from sklearn.model_selection import GridSearchCV
param_grid = {'max_depth':range(1, 7)}
grid = GridSearchCV(DecisionTreeClassifier(random_state=0),
                    param_grid=param_grid, cv=10)
grid.fit(X_train, y_train)
```

16 / 43

```
from sklearn.model_selection import GridSearchCV
param_grid = {'max_leaf_nodes':range(2, 20)}
grid = GridSearchCV(DecisionTreeClassifier(random_state=0),
                    param_grid=param_grid, cv=10)
grid.fit(X_train, y_train)
```

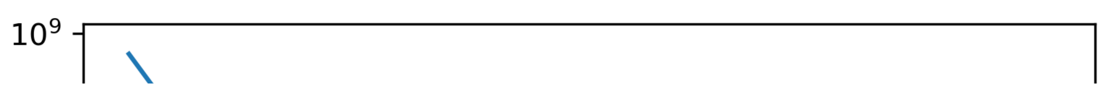


17 / 43

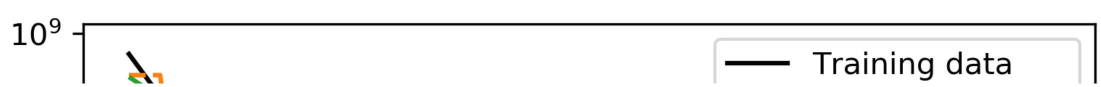
Relation to Nearest Neighbors

- Predict average of neighbors – either by k, by epsilon ball or by leaf.
- Trees are much faster to predict.
- Both can't extrapolate

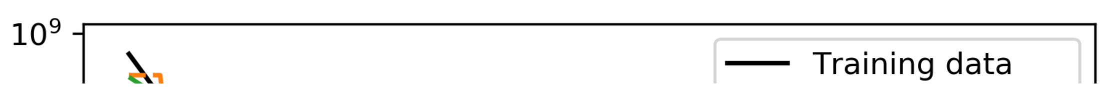
Extrapolation



Extrapolation

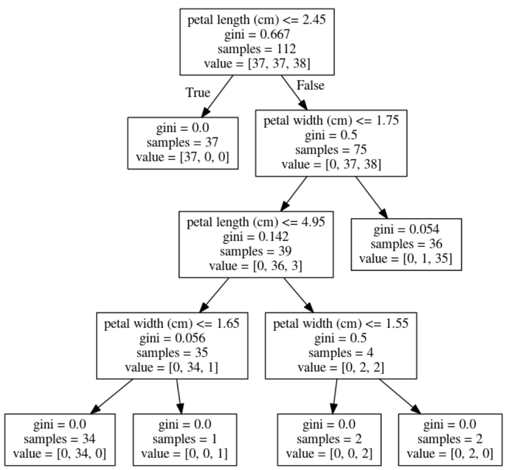


Extrapolation

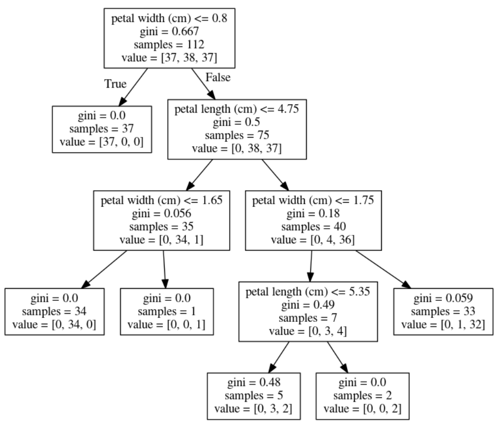


Instability

```
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, stratify=iris.target, random_state=0)
tree = DecisionTreeClassifier(max_leaf_nodes=6)
tree.fit(X_train, y_train)
```

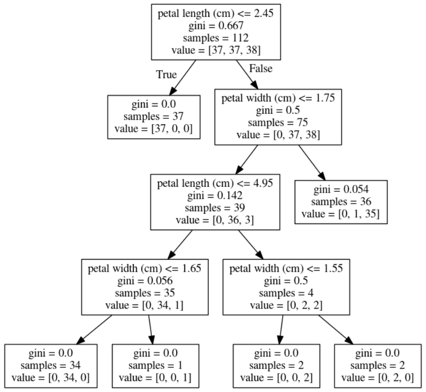



```
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, stratify=iris.target, random_state=1)
tree = DecisionTreeClassifier(max_leaf_nodes=6)
tree.fit(X_train, y_train)
```

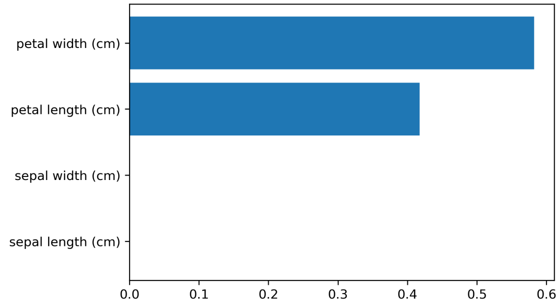


Feature importance

```
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, stratify=iris.target, random_state=0)
tree = DecisionTreeClassifier(max_leaf_nodes=6)
tree.fit(X_train,y_train)
```




```
tree.feature_importances_
array([0.0, 0.0, 0.414, 0.586])
```



Categorical Data

- Can split on categorical data directly
- Intuitive way to split: split in two subsets
- 2 ^ n_values many possibilities
- Possible to do in linear time exactly for gini index and binary classification.
- Heuristics done in practice for multi-class.
- Not in sklearn :(

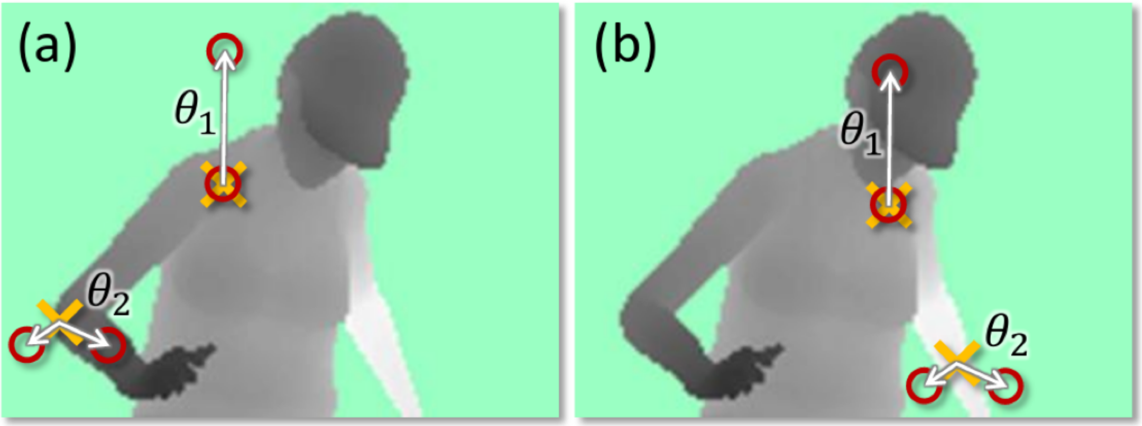
Predicting probabilities

- Fraction of class in leaf.
- Without pruning: Always 100% certain!
- Even with pruning might be too certain.

Conditional Inference Trees

- Select “best” split with correcting for multiple-hypothesis testing.
- More “fair” to categorical variables.
- Only in R so far (party)

Different splitting methods



(taken from Shotton et. al. Real-Time Human Pose Recognition ..)

Ensemble Models

Poor man’s ensembles

- Build different models
- Average the result
- Owen Zhang (long time kaggle 1st): build XGBoosting models with different random seeds.
- More models are better – if they are not correlated.
- Also works with neural networks
- You can average any models as long as they provide calibrated (“good”) probabilities.
- Scikit-learn: VotingClassifier hard and soft voting

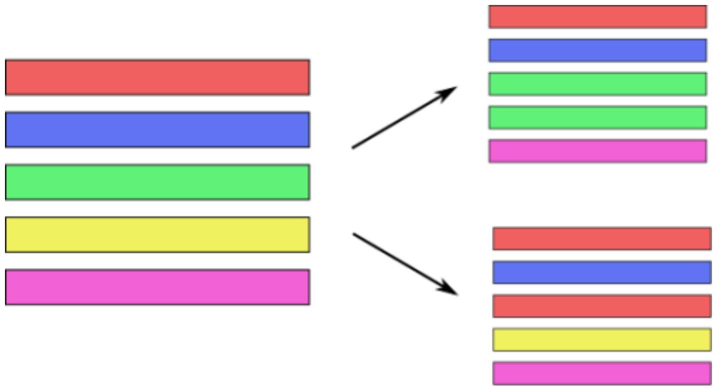
VotingClassifier

```
voting = VotingClassifier(
    [('logreg', LogisticRegression(C=100)),
     ('tree', DecisionTreeClassifier(max_depth=3, random_state=0))],
    voting='soft')
voting.fit(X_train, y_train)
lr, tree = voting.estimators_
voting.score(X_test, y_test), lr.score(X_test, y_test), tree.score(X_test, y_test)
```

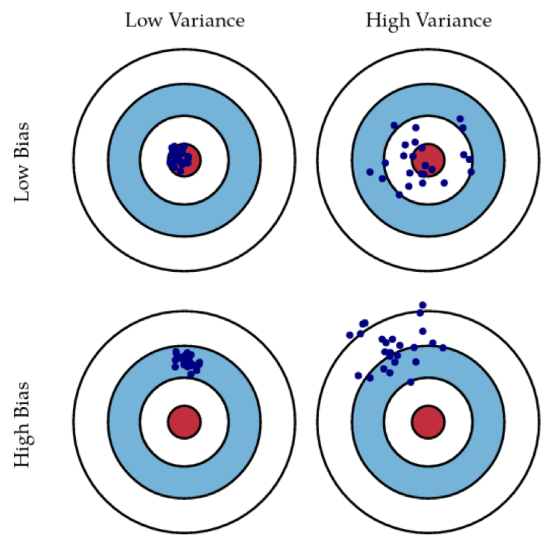
```
0.88 0.84 0.80
```

Bagging (Bootstrap AGGregation)

- Generic way to build “slightly different” models
- BaggingClassifier, BaggingRegressor



Bias and Variance

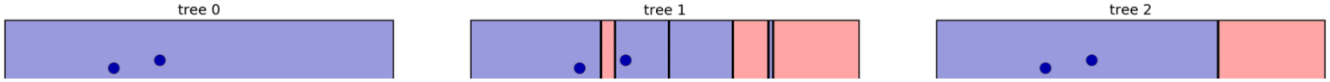


http://scott.fortmann-roe.com/docs/BiasVariance.html

Bias and Variance in Ensembles

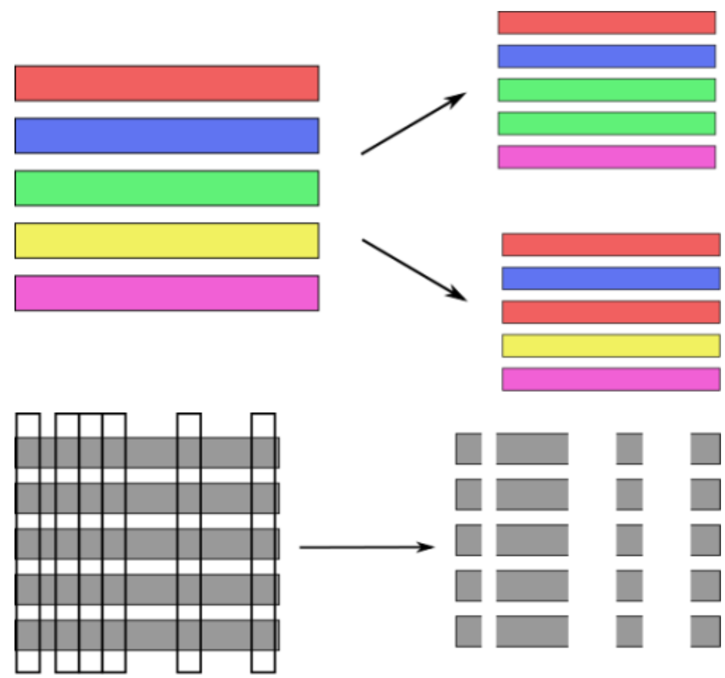
- Breiman showed that generalization depends on strength of the individual classifiers and (inversely) on their correlation
- Uncorrelating them might help, even at the expense of strength

Random Forests



Randomize in two ways

- For each tree:
  - Pick bootstrap sample of data
- For each split:
  - Pick random sample of features
- More trees are always better



Tuning Random Forests

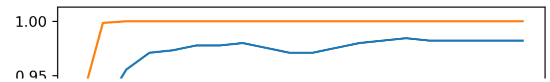
- Main parameter: max_features
  - around sqrt(n_features) for classification
  - Around n_features for regression
- n_estimators > 100
- Prepruning might help, definitely helps with model size!
- max_depth, max_leaf_nodes, min_samples_split again

Extremely Randomized Trees

- More randomness!
- Randomly draw threshold for each feature!
- Doesn't use bootstrap
- Faster because no sorting / searching
- Can have smoother boundaries

Warm-Starts

```
train_scores = []
test_scores = []

rf = RandomForestClassifier(warm_start=True)
estimator_range = range(1, 100, 5)
for n_estimators in estimator_range:
    rf.n_estimators = n_estimators
    rf.fit(X_train, y_train)
    train_scores.append(rf.score(X_train, y_train))
    test_scores.append(rf.score(X_test, y_test))
```



Out-of-bag estimates

- Each tree only uses ~66% of data
- Can evaluate it on the rest!
- Make predictions for out-of-bag, average, score.
- Each prediction is an average over different subset of trees

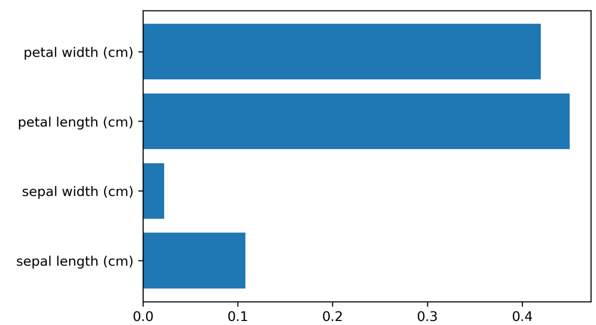
```
train_scores = []
test_scores = []
oob_scores = []

feature_range = range(1, 64, 5)
for max_features in feature_range:
    rf = RandomForestClassifier(max_features=max_features, oob_score=True,
                                n_estimators=200, random_state=0)
    rf.fit(X_train, y_train)
    train_scores.append(rf.score(X_train, y_train))
    test_scores.append(rf.score(X_test, y_test))

    oob_scores.append(rf.oob_score_)
```


1.00

Variable Importance

```
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, stratify=iris.target, random_state=1)
rf = RandomForestClassifier(n_estimators=100).fit(X_train, y_train)
rf.feature_importances_
plt.barh(range(4), rf.feature_importances_)
plt.yticks(range(4), iris.feature_names);
```

```
array([ 0.126, 0.033, 0.445, 0.396])
```



Questions ?