# Applied Machine Learning

# Decision Trees

Grant Schoenebeck

Some material courtesy of Andrew W. Moore. See http://www.cs.cmu.edu/~awm/tutorials.html

# Outline

- Machine Learing versus Statistic Philosophy
- More Data Leakage
- Calibrating SVMs
- OneHotEncoding
- Decision Trees
- Bagging
  - *Random Forest*
- Boosting
  - *Gradient Boosted Trees*
- Multi-Class Evaluation
- (Break?)
- Paper on Detecting Review Spam

# Machine Learning Versus Statistics

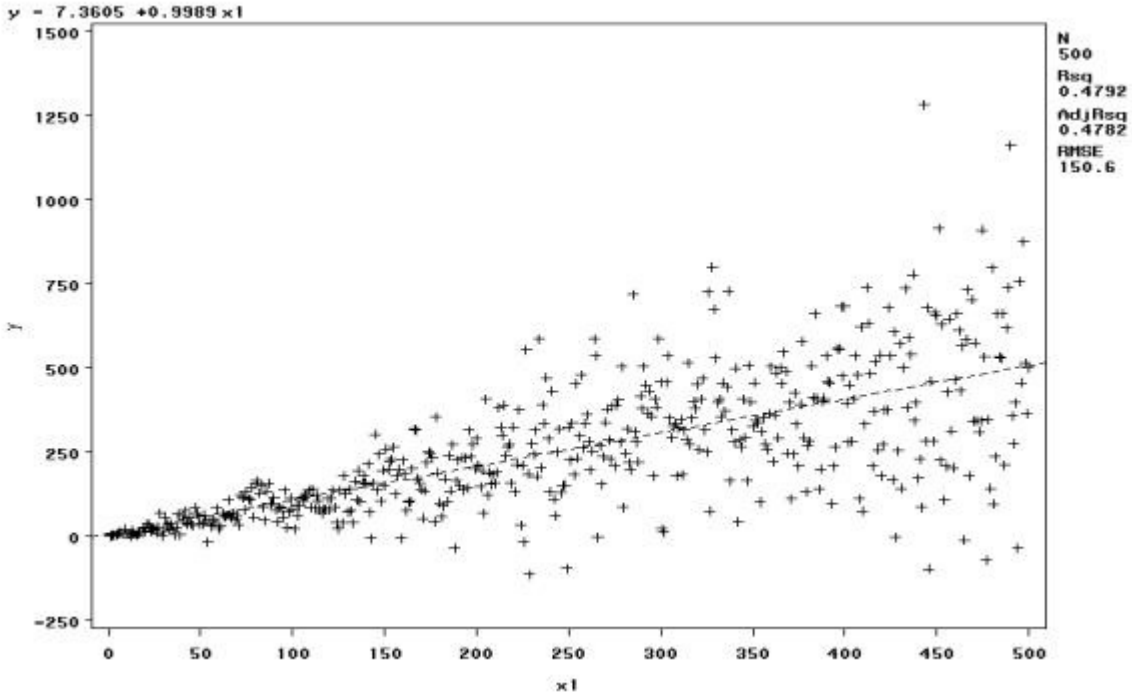
## Statistics

- Multipurpose:
  - *Predictions*
  - *Relationships Variables*
  - *Confidence Bounds*
- Requires many assumptions for validity. (Heteroskedasticity)
- New assumptions needed for each model.
- Carefully reasons about interpretations (e.g. significance of feature).

## Machine Learning

- Focuses on prediction on unseen data.
- Requires being very careful about a few things. (Data leakage, fairness).
- Complex and multifaceted models.
- Has "rule of thumb" metrics for interpretations.

# Heteroskedasticity



# Using Cross-Validation Correctly

What is wrong with the following:

1) Split Data Into Training and Validation.

2) For three different types of learners, tune hyper parameters training on Training Data; Testing on Validation Data.

3) With all the data, use cross-validation to check accuracy of each learner.

?

# Using Cross-Validation Correctly

Can stack Cross Validations:

Within each Cross-Validation Fold,

1) Use cross-validation on the (fold's) training data
2) Test on the (fold's) test data

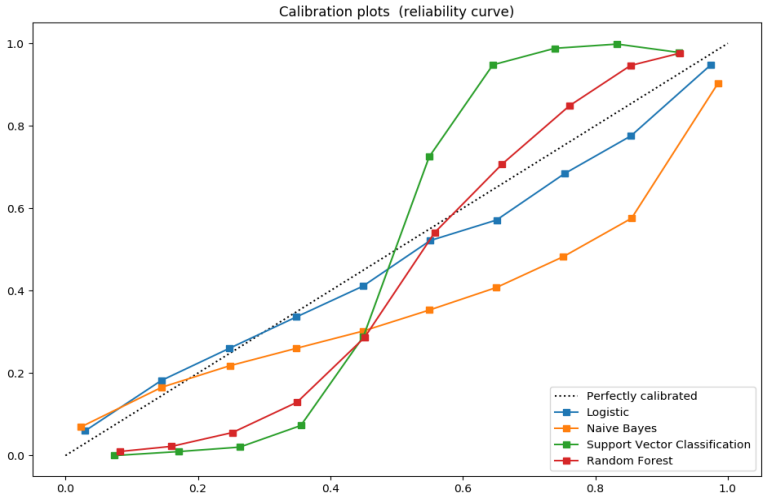
# Calibration and Calibrating

# Calibration

- **Predictor f(X) is calibrated if:**
  - *When* $E_{x_i \in W}[y_i] \approx p$ *for* $W = \{x_i : f(x_i) \approx p\}$
  - *It rains 60% of the days on which the weatherman predicts it will rain 60% of the time.*
- **Is it hard to get a calibrated predictor?**
  - *NO. Just predict the overall estimate for y.*
- **How to turn a predictor into a calibrated one:**
  - *Platt Scaling*
  - *Isotonic regression*

# Not Calibrated

$\mu_H(p)$




p

$\mu_H(p) = E[Y_i|H(X_i) = p]$ = Fraction instances with Predicted $\Pr(Y) = p$ where Y=1

# Platt Scaling

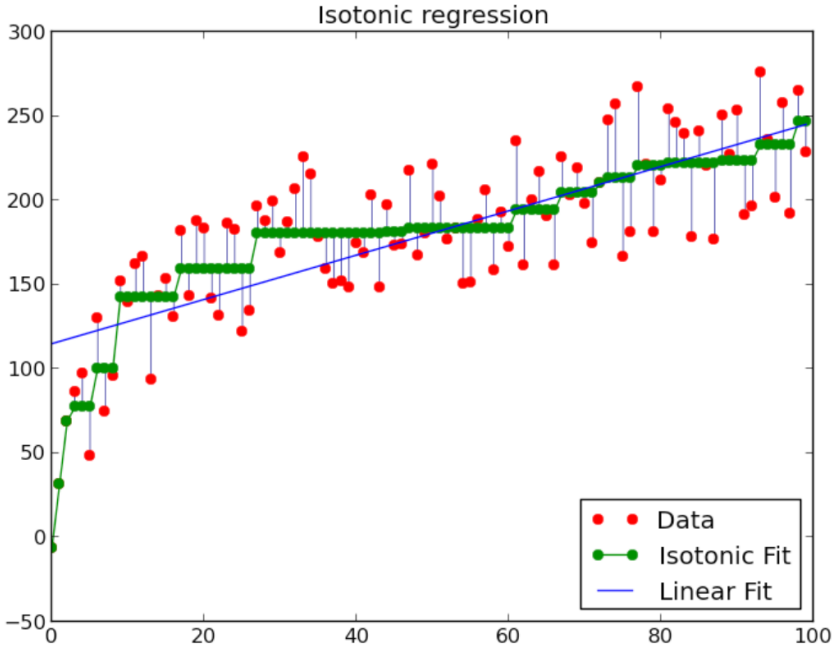
- Predict $Pr[Y_i = 1]$
- However, you have uncalibrated f(x)
- Idea: using affine transformation + logistic function

$$\mathrm{P}(y = 1|x) = \frac{1}{1 + \exp(Af(x) + B)},$$

- Tune A and B

# Isotonic Regression

- Plot prediction of soft classifier against fraction of actual Ys.
- Perform isotonic regression to get g
- Guarantees increasing function g
- Use $g \circ f(X) = g(f(X))$



# Isotonic Regression

- Plot prediction of soft classifier against fraction of actual Ys.
- Perform isotonic regression to get g
- Guarantees increasing function g
- Use $g \circ f(X) = g(f(X))$

# Question

Why do we want a calibrated classifier?

A) Improves accuracy

B) Give a more accurate sense of confidence

C) Reduces overfitting

D) All of the above

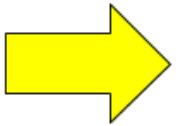
E) None of the above

# 1-hot encoding

# 1-hot encoding

- Turns categories into sequence of dummy variables.

| Color |
| --- |
| Red |
| Red |
| Yellow |
| Green |
| Yellow |



| Red | Yellow | Green |
| --- | --- | --- |
| 1 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

# 1-hot encoding

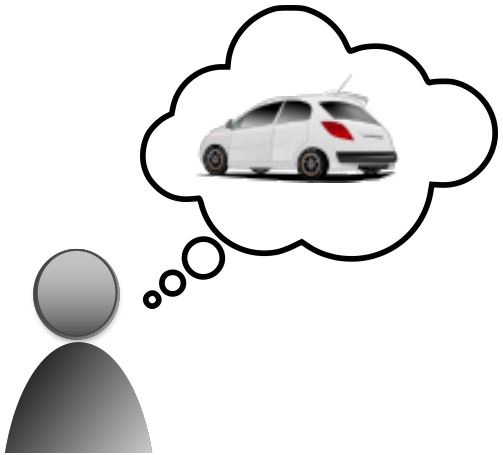
- Many classifiers cannot take categorical input:
  - *OLS/Logistic Regression*
  - *SVMs*
- Numerical data is sometimes categorical:
  - *Encodings: {1: apple, 2: orange, 3: lemon, etc}*
- Apply to training and test data together
  - *Ensures dummy variables match*
  - *Ensures all categories represented*
- Can be extended for NLP use (e.g. to encode a word)

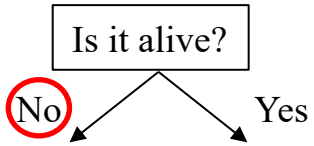
# 1-hot encoding

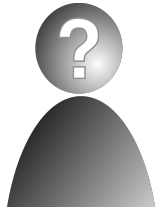
- **Can use in Pandas:**
  - *.get_dummies*
  - *Applies to all none numerical data.*
- **Can use scikit learn as well (must apply to each column):**

# Decision Trees

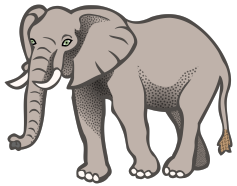
# Decision Tree Example



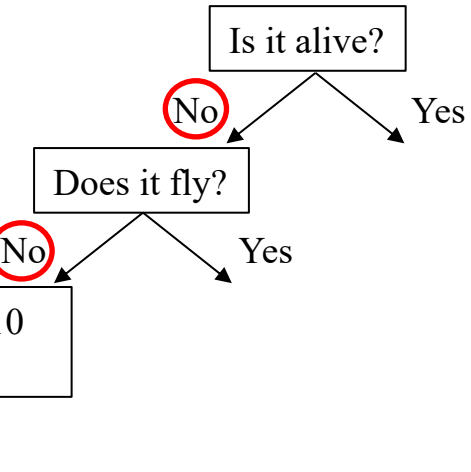


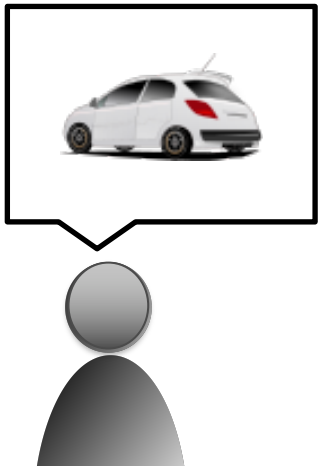



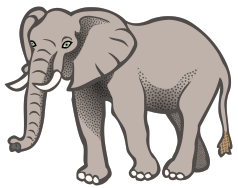
Objects with Alive == No



# Decision Tree Example

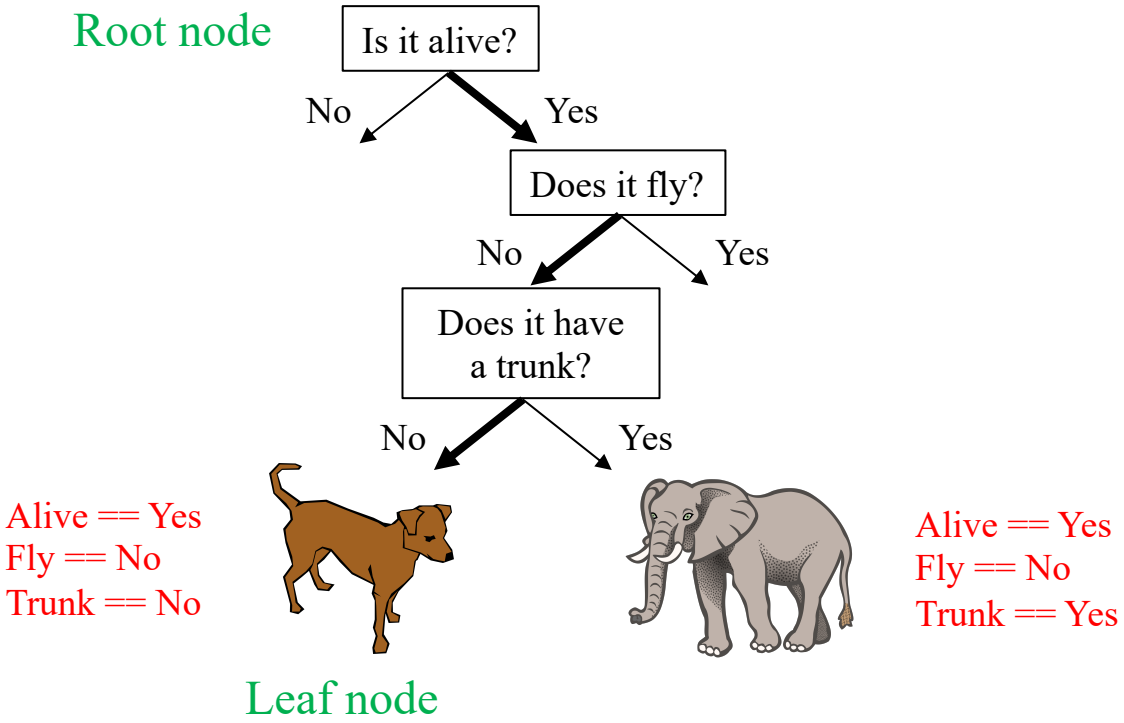




Objects with: Alive == No AND Fly == No AND Carry > 10 == No

# Decision Tree Example



# Learning Decision Trees

- **Decision tree**:
  - *A tree-structured plan for a set of attributes to test in order to predict the output.*

- **Which attribute should be tested first?**
  - *Find the one with highest "information gain"*
  - *Then recurse…*

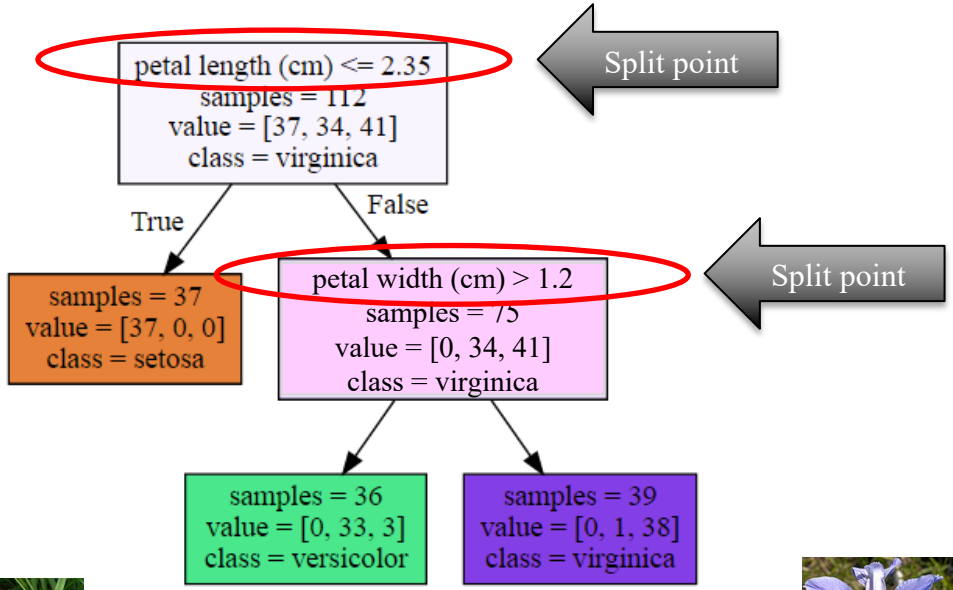
# The Iris Dataset


*Iris setosa* *Iris versicolor* *Iris virginica*

150 flowers
3 species
50 examples/species

Photo credit: Radomił Binek via https://en.wikipedia.org/wiki/Iris_flower_data_set

# Decision Tree Splits




samples at this leaf have:
**petal length <= 2.35**

*setosa*

*versicolor*

samples at this leaf have:
**petal length > 2.35**
**AND petal width <= 1.2**

samples at this leaf have:
**petal length > 2.35**
**AND petal width > 1.2**

*virginica*

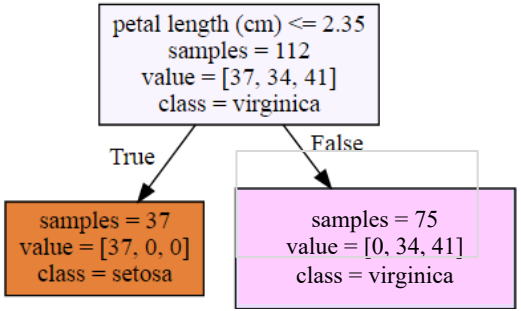
# Learning Trees

- **Scikit Learn Implements CART**
  - *(Classification and Regression Trees)*
  - *Cannot handle categorical variables*
- **Greedy binary split**
  - *Classification: entropy maximizing split*
  - *Regression: optimize mean squared error*

# Entropy Maximization

- Let $f(X_i)$ predict $Y_i$ in the "natural way"
  - *If 30 "1" instances and 20 "0" instances map to a leaf, predict 60%*
- Empirical Risk Minimization using Log loss:
  - *Loss $L(f(X_i))$ is:*
    - $L(f(X_i) = -\log(f(X_i))$ *if* $Y_i = 1$
    - $L(f(X_i) = -\log(1 - f(X_i))$ *if* $Y_i = 0$
  - *Can easily extend to multiclass regression*
    - $L(f(X_i) = \log(Pr[\text{assigned to correct class}])$
- Also called entropy minimization.
- Choose split that minimizes log loss

Pure node (all one class: perfect classification)



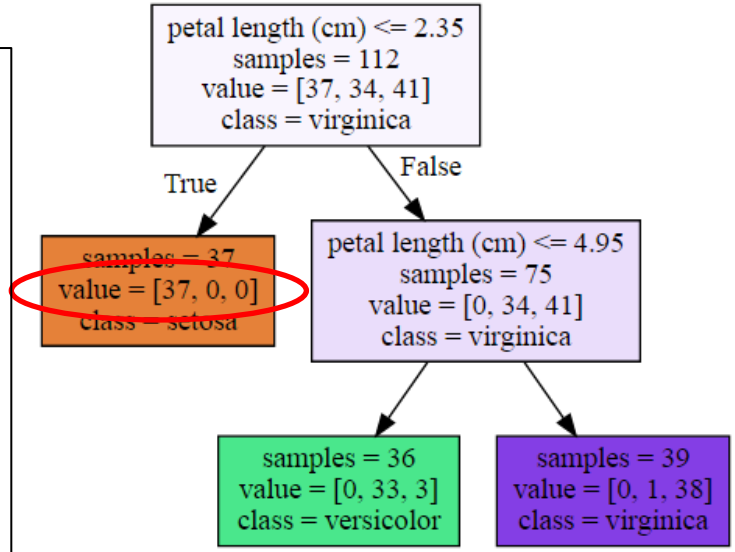

Mixed node (mixture of classes, still needs further splitting)

# Value lists

**The *value* list gives the number of samples of each class that end up at this leaf node during training.**

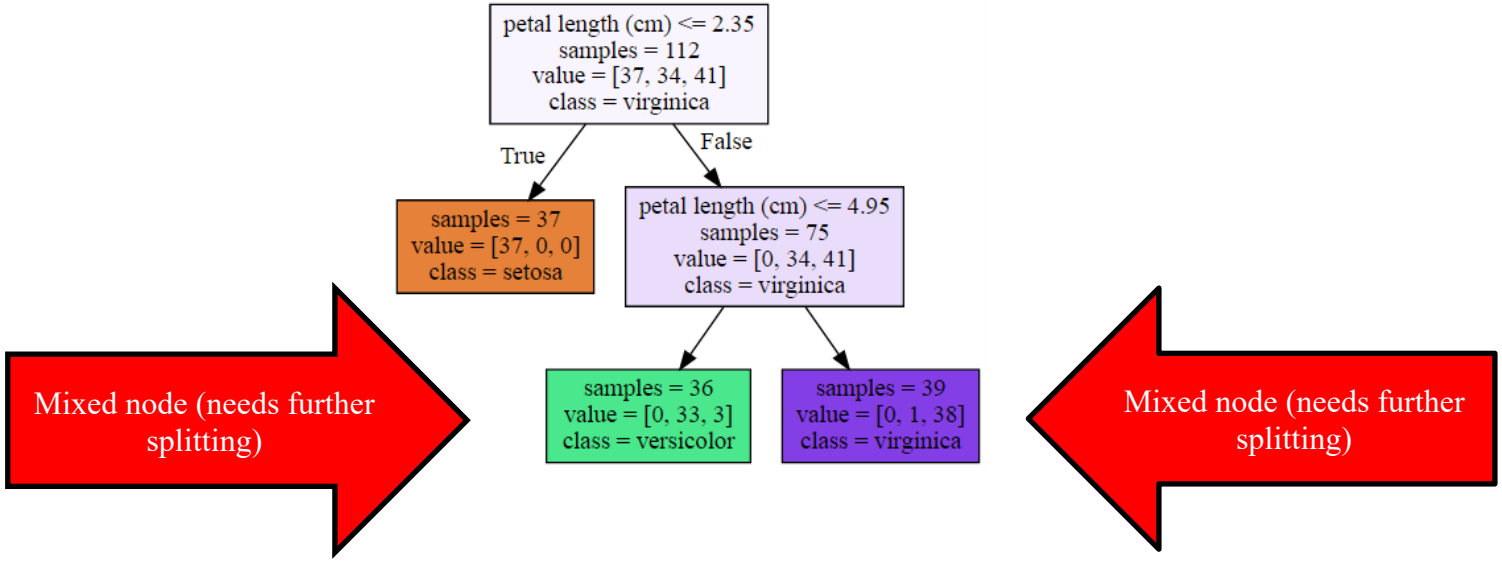
The iris dataset has 3 classes, so there are three counts.

This leaf has 37 setosa samples, zero versicolor, and zero virginica samples.



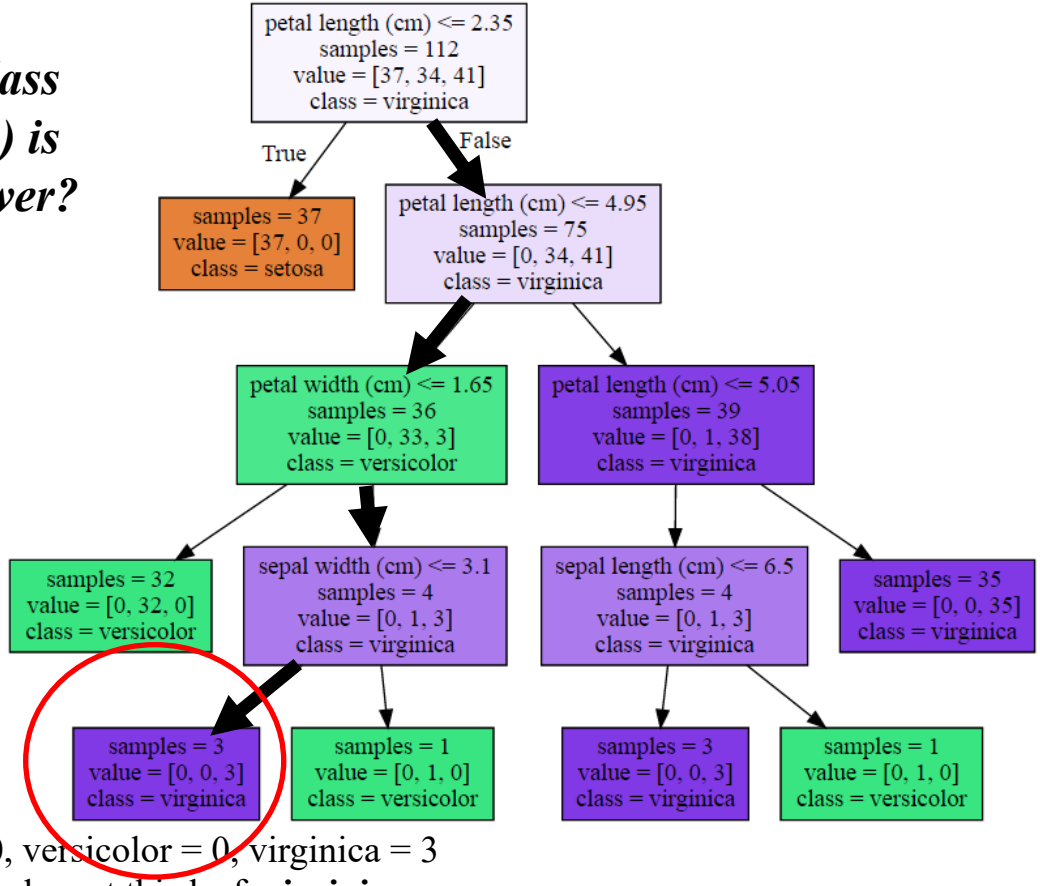

This leaf has 0 setosa, 33 versicolor, and 3 virginica samples.

This leaf has 0 setosa, 1 versicolor, and 38 virginica samples.


petal length (cm) <= 2.35
samples = 112
value = [37, 34, 41]
class = virginica
True
False
samples = 37
value = [37, 0, 0]
class = setosa
petal length (cm) <= 4.95
samples = 75
value = [0, 34, 41]
class = virginica
samples = 36
value = [0, 33, 3]
class = versicolor
samples = 39
value = [0, 1, 38]
class = virginica
Mixed node (needs further splitting)
Mixed node (needs further splitting)

***What class (species) is this flower?***

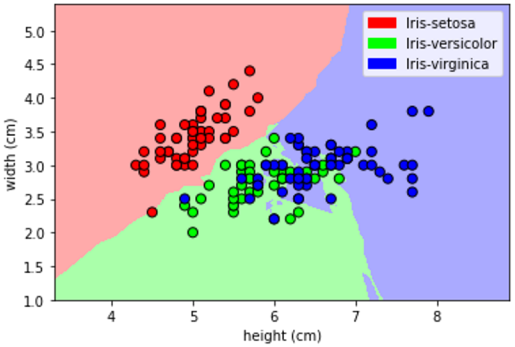
petal length: 3.0
petal width: 2.0
sepal width: 2.0
sepal length: 4.2

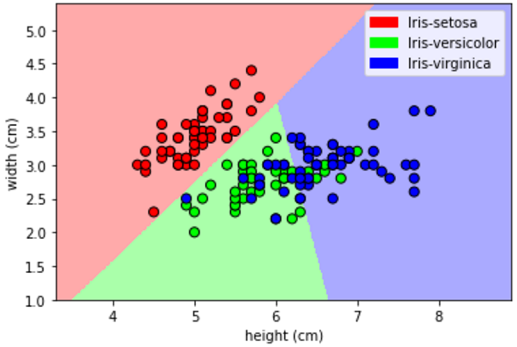


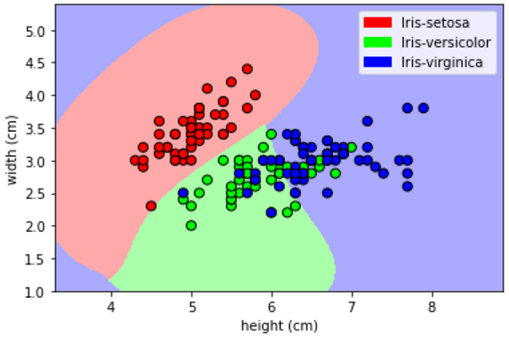

Leaf counts are: setosa = 0, versicolor = 0, virginica = 3
Predicted class is majority class at this leaf: **virginica**

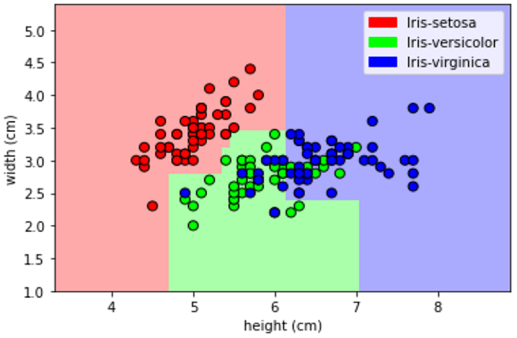
# Think, Pair, Share

- Which of the following is from a decision tree?



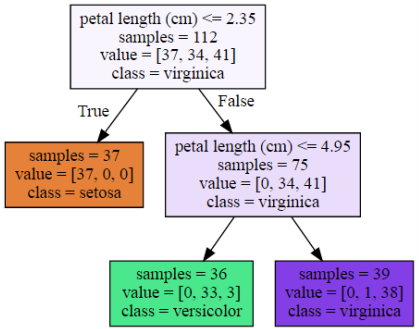



# Regression Trees

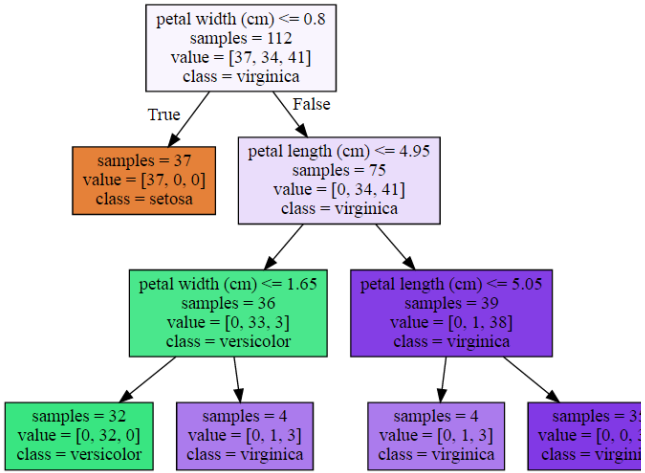
- Same as decision tree, but leaf predicts average of all corresponding items.

# Controlling the Model Complexity of Decision Trees

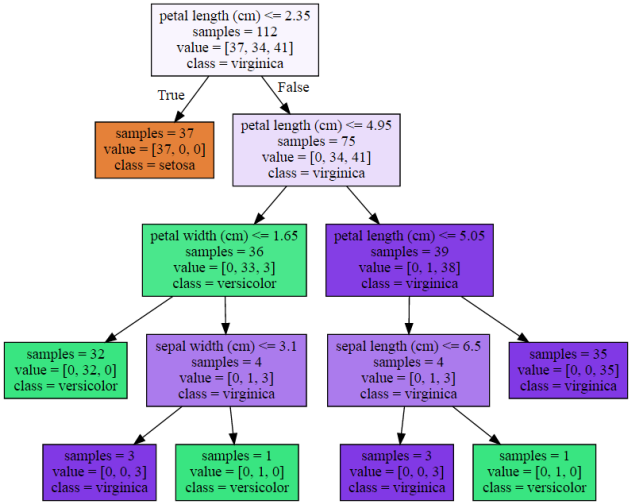
`max_depth = 2`




`max_depth = 3`

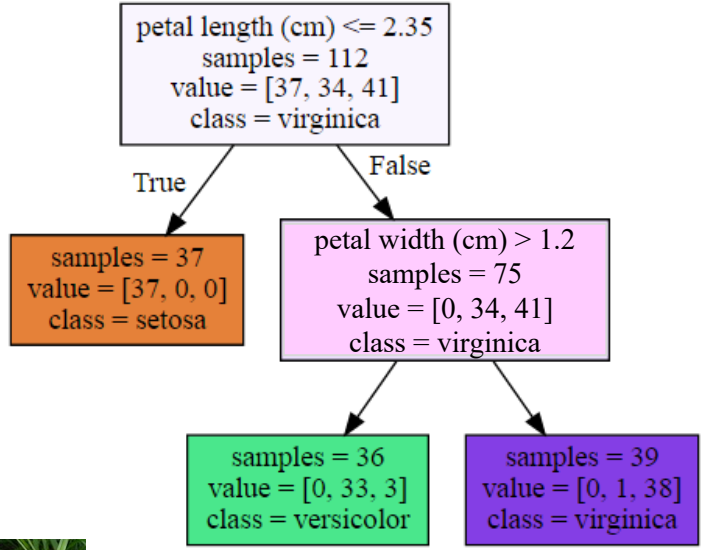



`max_depth = 4`




Other parameters: Max. # of leaf nodes: `max_leaf_nodes` Min. samples to consider splitting: `min_samples_leaf`

# What's the nature of the decision regions found by decision trees?




samples at this leaf have:
**petal length <= 2.35**

*setosa*

*versicolor*

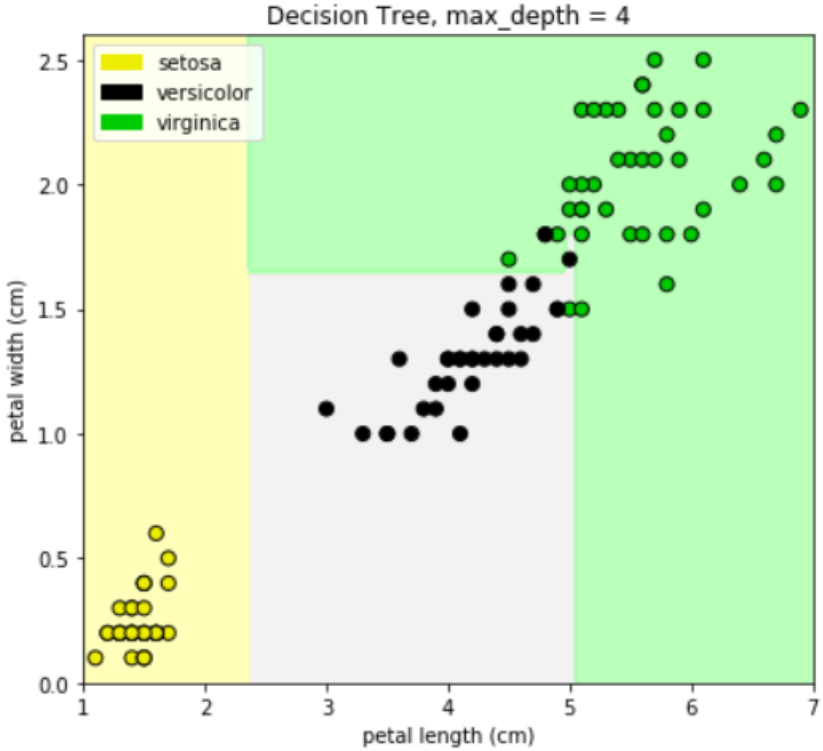
samples at this leaf have:
**petal length > 2.35**
**AND petal width <= 1.2**

samples at this leaf have:
**petal length > 2.35**
**AND petal width > 1.2**

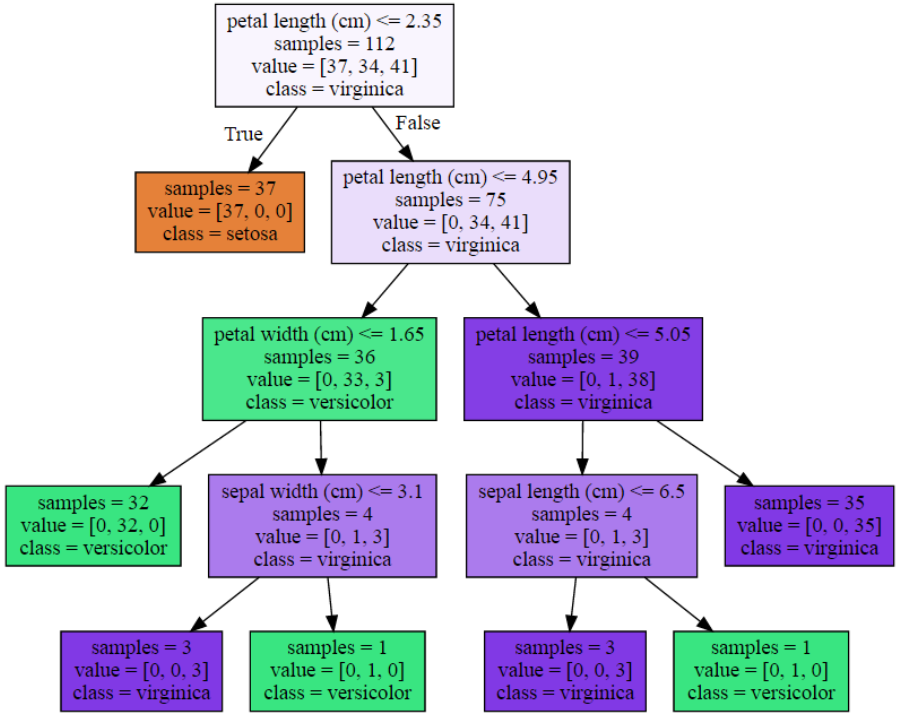
*virginica*

**Think about how each "data bin" at a leaf is defined…**

# What's the nature of the decision regions found by decision trees?



# Visualizing Decision Trees

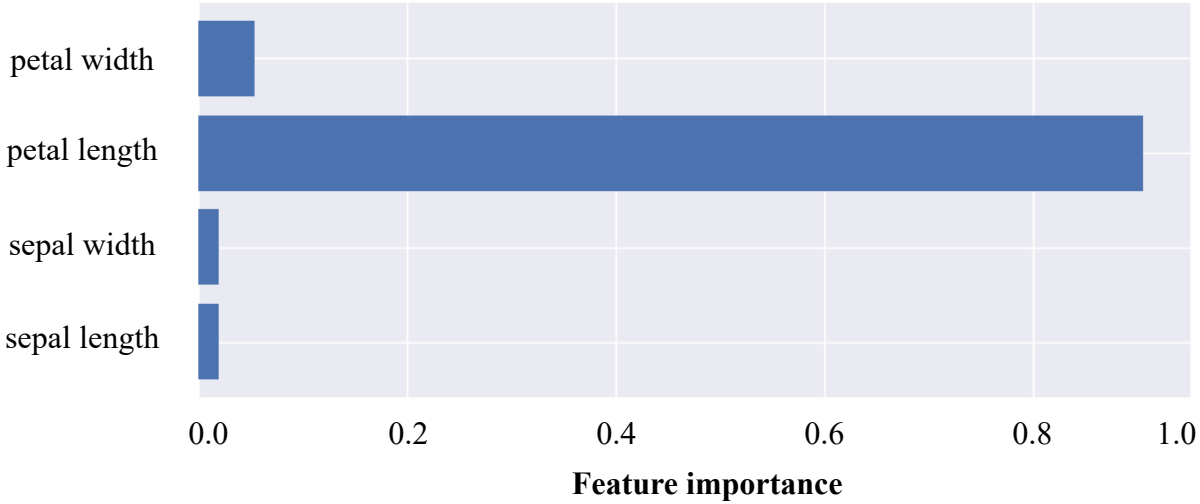


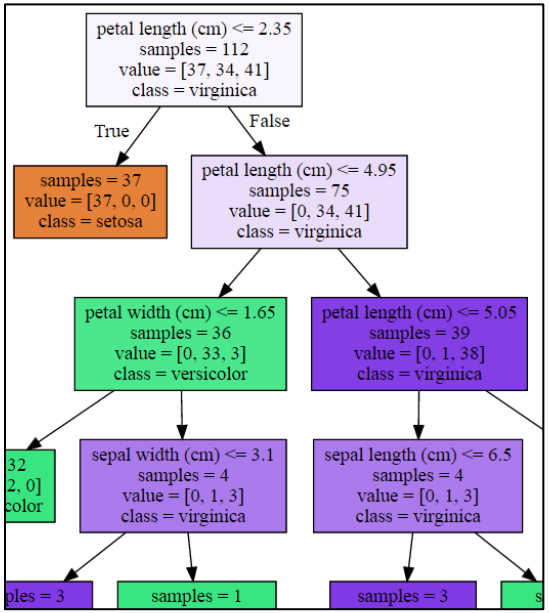

See: `plot_decision_tree()` function in `adspy_shared_utilities.py` code

# Feature Importance: How important is a feature to overall prediction accuracy?

- A number between 0 and 1 assigned to each feature.
- Feature importance of 0 → the feature was not used in prediction.
- Feature importance of 1 → the feature predicts the target perfectly.
- All feature importance are normalized to sum to 1.
- In scikit learn: ~fraction of splits on that variable
  - *a pair of items is "split" when they no longer follow the same path in tree.*
  - $\sum_{i=1}^{c} -f_i(1-f_i)$ *where* $f_i$ *is fraction of nodes in class i.*
- Often a poor metric when features are correlated

# Feature Importance Chart

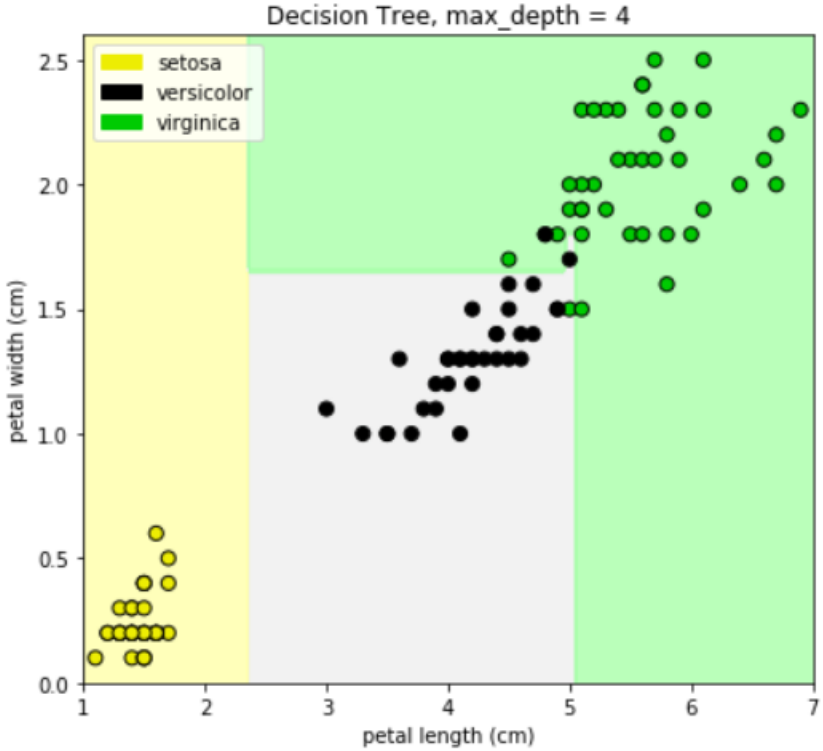



Decision tree

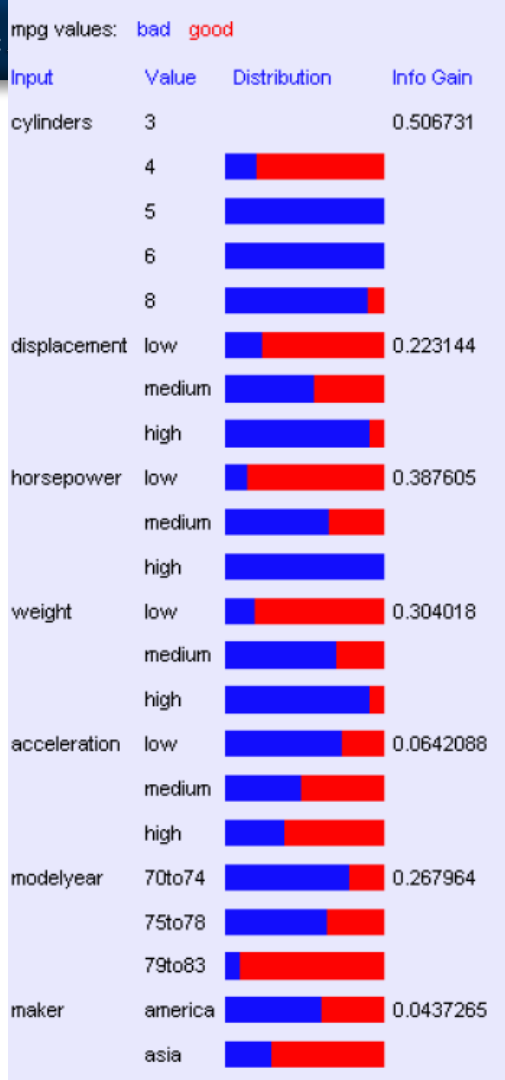
See: `plot_feature_importances()` function in `adspy_shared_utilities.py` code

# What's the nature of the decision regions found by decision trees?



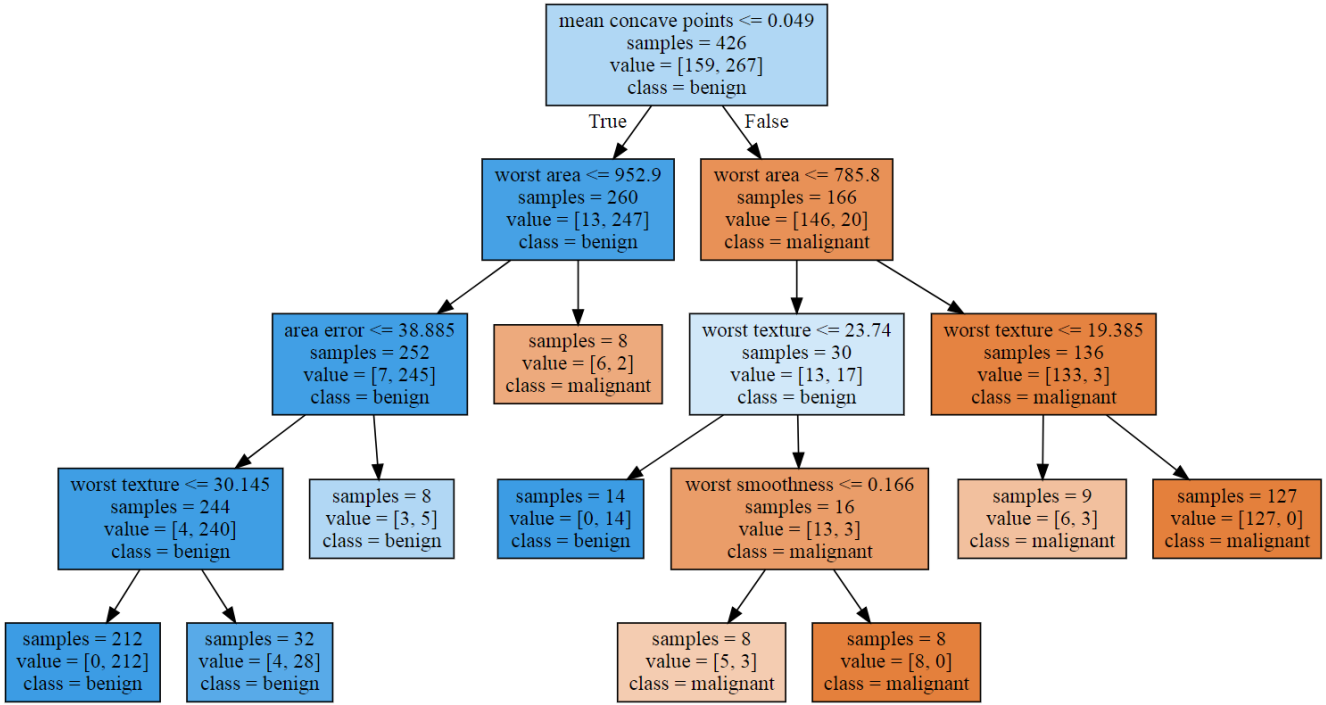
# Look at information gains

- How much does each input variable (feature) tell us about whether a car has good/bad fuel mileage (MPG)?
- Higher IG → more useful feature because it groups the data into bins with more "pure" class divisions

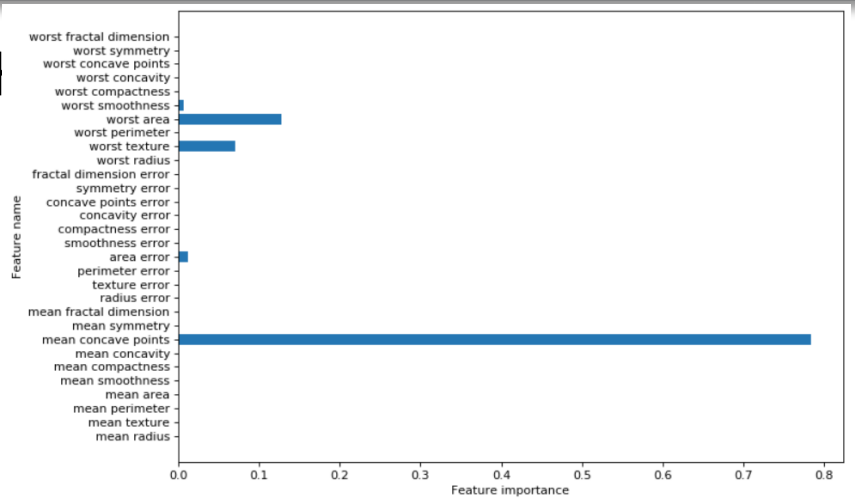


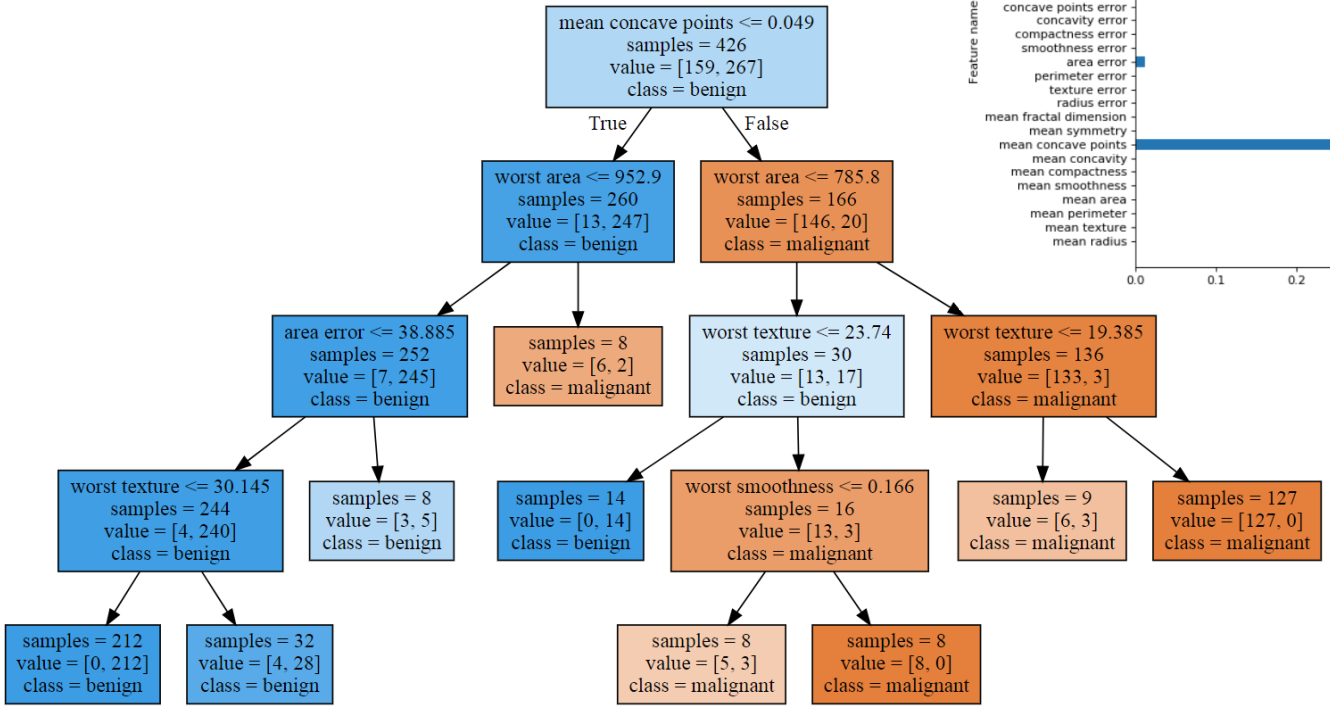

Source: Decision Tree Tutorial, Andrew Moore

# Applying decision trees to the breast cancer dataset



# Applying decision trees to tl



# Decision Trees: DecisionTreeClassifier Key Parameters

- `max_depth`: controls maximum depth (number of split points). Most common way to reduce tree complexity and overfitting.
- `min_samples_leaf`: threshold for the minimum # of data instances a leaf can have to avoid further splitting.
- `max_leaf_nodes`: limits total number of leaves in the tree.
- In practice, adjusting only one of these (e.g. `max_depth`) is enough to reduce overfitting.

# Decision Trees: Pros and Cons

**Pros:**

- Easily interpreted/ visualized
- Feature normalization/ scaling not needed.
- Seemlessly combines feature types:
  - *continuous,*
  - *categorical (scikit learn less so)*
  - *binary*

**Cons:**

- Prone to overfit
- Less great for sparse high-dimensional data (e.g. text)
- Typically use ensembles
  - *But then lose interpretability*

# Questions to Ponder

- What's an example of a classification problem where decision trees would crush linear SVMs?
- What's an example of a classification problem where a linear SVM would handily win over a decision tree?

# Bonus: Regression Trees in scikit-learn

http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html
DecisionTreeRegressor

http://www.stat.cmu.edu/~cshalizi/350-2006/lecture-10.pdf

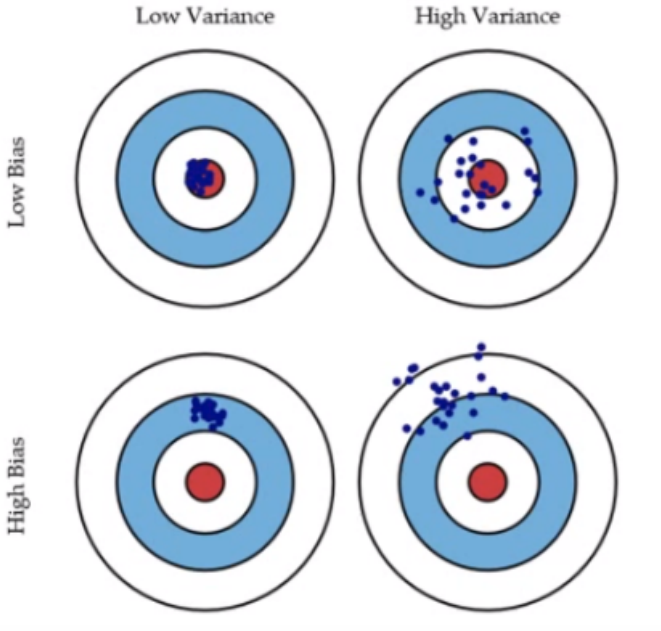
Pros and Cons of RT, and lots more on how to use:
http://www.statsoft.com/Textbook/Classification-and-Regression-Trees

# Ensembles: Bagging

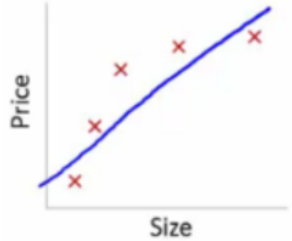
# Ensemble Methods

- combine the predictions of multiple estimators
  - *Bagging*
  - *Boosting*
- Why bagging?

# Bias vs variance  around a single "target "point

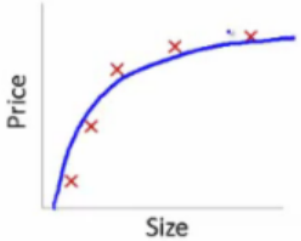


# Bias-variance tradeoffs



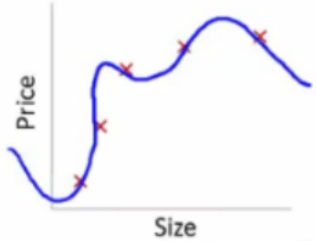

$\theta_0 + \theta_1 x$

High bias
(underfit)




$\theta_0 + \theta_1 x + \theta_2 x^2$

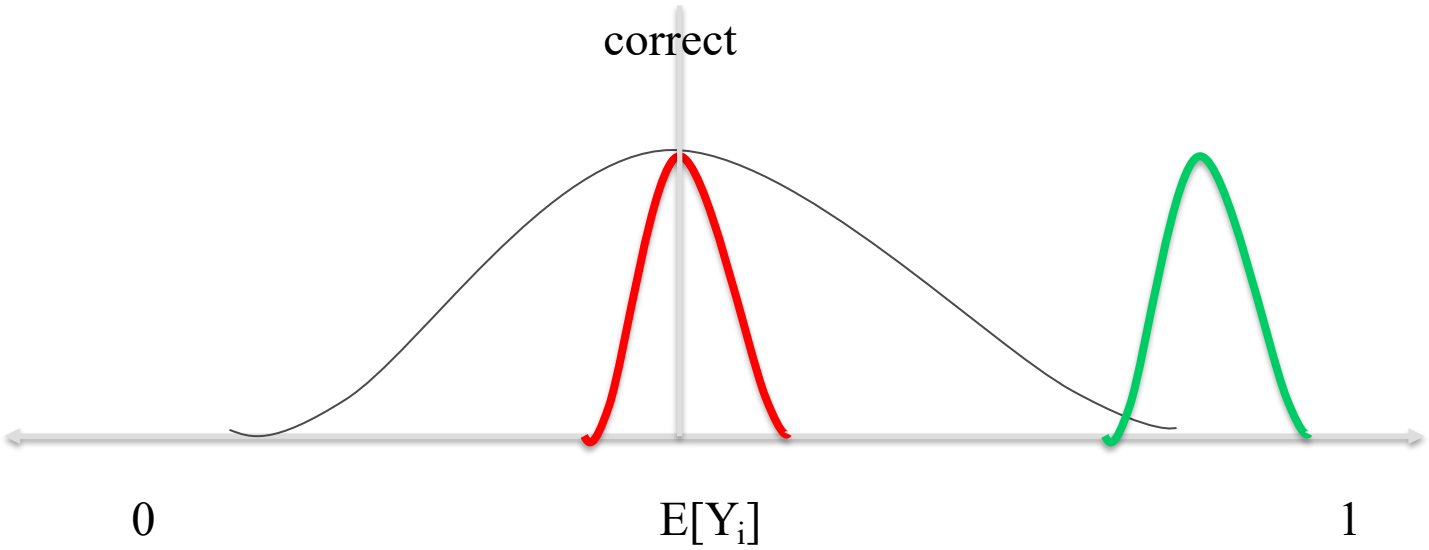
"Just right"




$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance
(overfit)

# Bias and Variance



# Bias and Variance

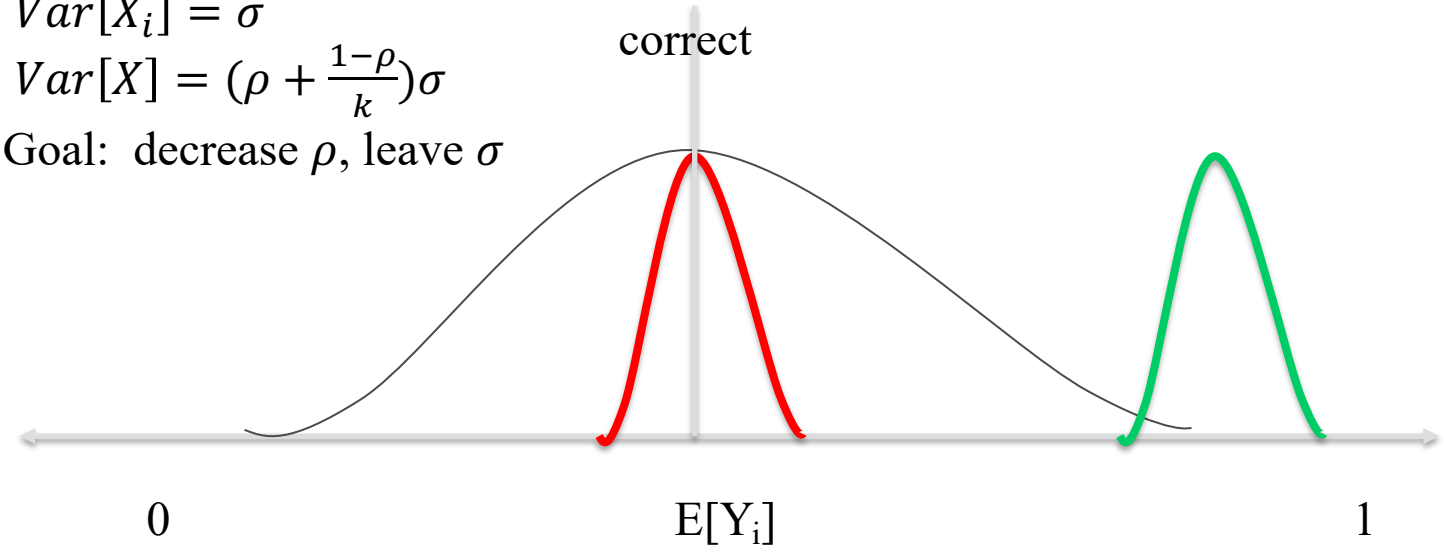
Idea: average several classifiers to reduce variance.

Let $X = \frac{1}{k}\sum_{i=1}^{k} X_i$ where $X_i$ are $\rho$ coorelated

$Var[X_i] = \sigma$

$Var[X] = (\rho + \frac{1-\rho}{k})\sigma$

Goal: decrease $\rho$, leave $\sigma$



# Bagging

- Draw k bootstrap samples
- Learn classifier on each sample
- Take average

- Bootstrap sample:
  - *Draw n items u.a.r. with replacement from original n items*

# Question

- Is there something else we could do to make our trees "more independent"?

- Randomly restrict the variables permitted to split on.

# Random Forests

- An ensemble of trees, not just one tree.
- Widely used, very good results on many problems.
- `sklearn.ensemble` module:
  - *Classification:* `RandomForestClassifier`
  - *Regression:* `RandomForestRegressor`
- One decision tree → Prone to overfitting.
- Many decision trees → More stable, better generalization
- Ensemble of trees should be diverse: introduce random variation into tree-building.

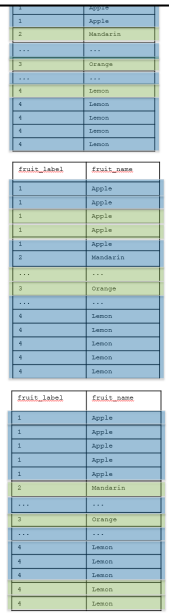
# Random Forest Process

Original dataset

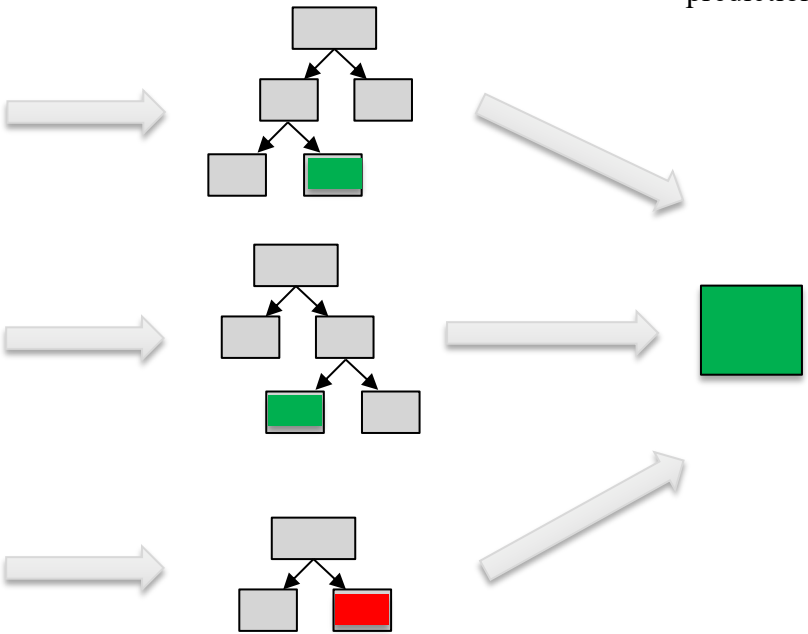
| fruit_label | fruit_name |
|---|---|
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 2 | Mandarin |
| ... | ... |
| 3 | Orange |
| ... | ... |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |

Randomized bootstrap copies

n_estimator



Randomized feature splits



Ensemble prediction

# Randomness #1: Random bootstrap samples to create each estimator's training set

Bootstrap sample 1

| fruit_label | fruit_name |
|---|---|
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 2 | Mandarin |
| ... | ... |
| 3 | Orange |
| ... | ... |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |

Bootstrap sample 2

| fruit_label | fruit_name |
|---|---|
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 2 | Mandarin |
| ... | ... |
| 3 | Orange |
| ... | ... |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |

Bootstrap sample 3

| fruit_label | fruit_name |
|---|---|
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 2 | Mandarin |
| ... | ... |
| 3 | Orange |
| ... | ... |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |

# Random Forest Process

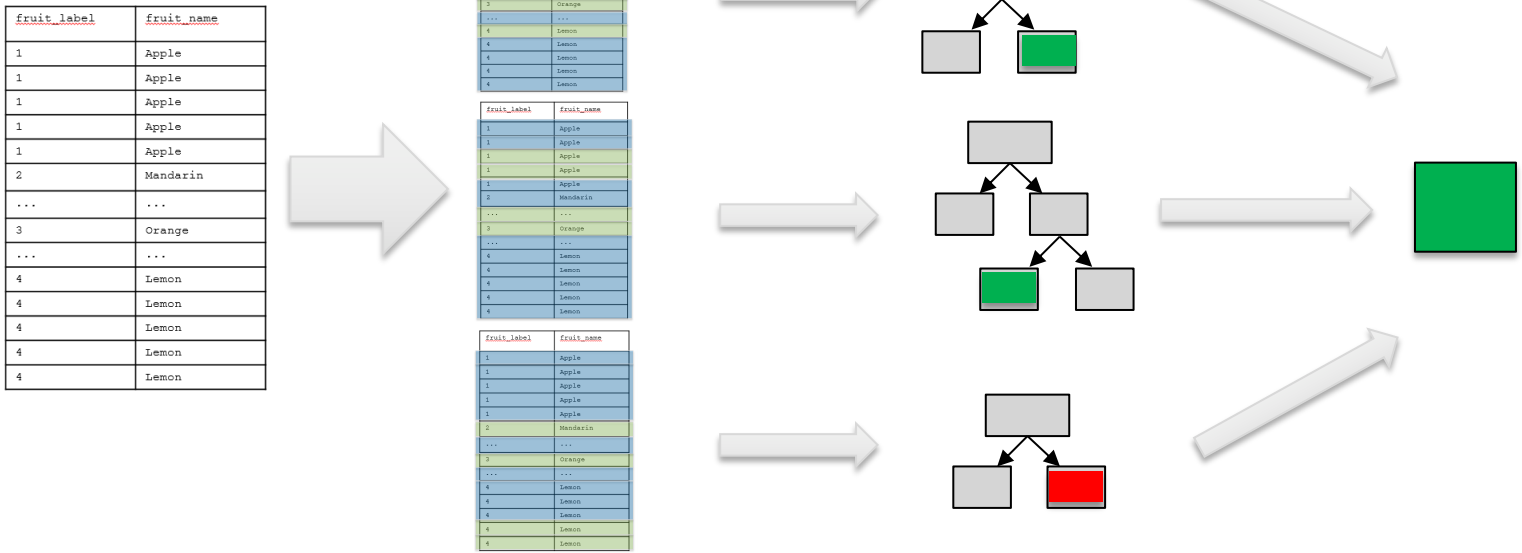
Original dataset

Randomized bootstrap copies

Randomized feature splits

max_features

Ensemble prediction



| fruit_label | fruit_name |
|---|---|
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 1 | Apple |
| 2 | Mandarin |
| ... | ... |
| 3 | Orange |
| ... | ... |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |
| 4 | Lemon |

# Random process #2: random feature-based splits when building the tree

- Only look at a random subset of possible features when creating each decision split
- Why is this important?

# Randomness #2: random feature-based splits when building the tree

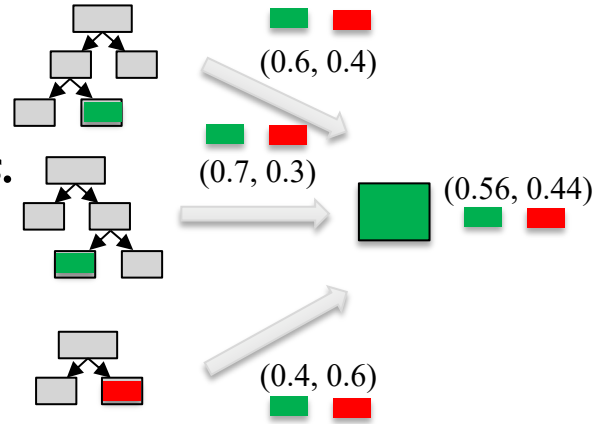
- Only look at a random subset of possible features when creating each decision split
- There are usually a few features highly correlated with each class (or the main class)
- Without random feature selection, the trees in the forest would all look quite similar… and thus would have highly correlated predictions.

# Random Forest `max_features` Parameter

- Learning is quite sensitive to `max_features`.
- Setting `max_features = 1` leads to forests with diverse, more complex trees.
- Setting `max_features = <close to number of features>` will lead to similar forests with simpler trees.

# Prediction Using Random Forests

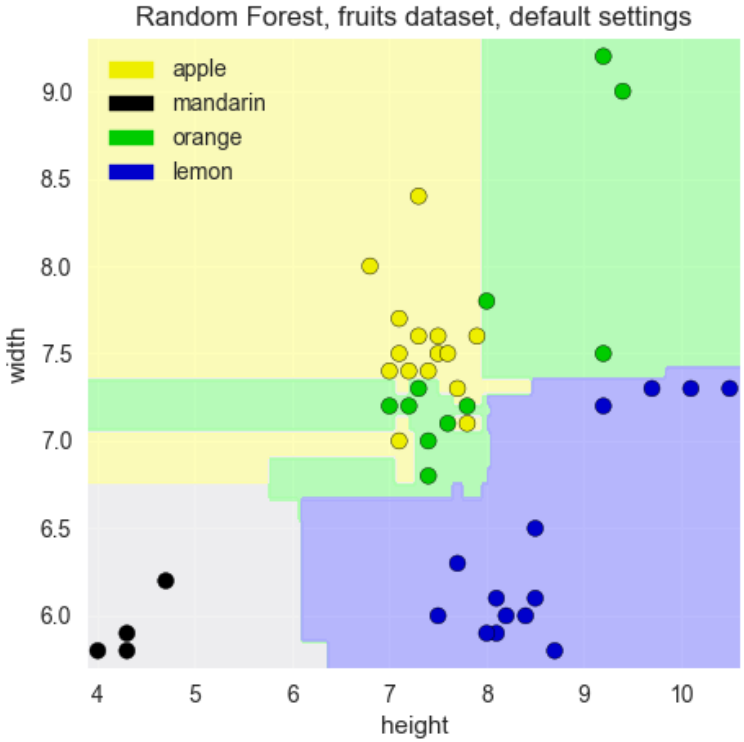
1. Make a prediction for every tree in the forest.
2. Combine individual predictions
   - *Regression: mean of individual tree predictions.*
   - *Classification:*
     - *Each tree gives probability for each class.*
     - *Probabilities averaged across trees.*
     - *Predict the class with highest probability.*



# Feature importance with random forests

- Rank the features according to how much they improve the model fit (e.g. reduce classification error) on average across trees
- Then normalize the improvement between 0 and 1
- More robust than for individual trees

# Random Forest: Fruit Dataset



# Random Forests: RandomForestClassifier Key Parameters

- `n_estimators`: number of trees to use in ensemble (default: 10).
  - *Should be larger for larger datasets to reduce overfitting (but uses more computation).*
- `max_features`: has a strong effect on performance. Influences the diversity of trees in the forest.
  - *Default works well in practice, but adjusting may lead to some further gains.*
- `max_depth`: controls the depth of each tree (default: None. Splits until all leaves are pure).
- `n_jobs`: How many cores to use in parallel during training.

- Choose a fixed setting for the random_state parameter if you need reproducible results.

# Random Forest: Pros and Cons

Pros:

- Excellent prediction performance
- No extensive parameter tuning.
- Handles a mixture of feature types.
- Easily parallelized

Cons:

- Hard to interpret.
- Not great at high dimensions

# Boosting

# Boostings

- **Very high level idea:**
  - *Learn the residuals.*
    - *Learn* $f$ *on* $(X_i, Y_i)$*;*
    - *let* $R_i = Y_i - f(X_i)$ *learn* $f'$ *on* $(X_i, R_i)$
    - *let* $R'_i = Y_i - f(R_i)$ *learn* $f''$ *on* $(X_i, R'_i)$
    - *Etc.*
    - *Return* $f + f' + f''$
  - *Is this a good idea for linear regression?'*

# Ada Boostings

---

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution $D_t$.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.
- Aim: select $h_t$ with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

- Choose $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$.
- Update, for $i = 1, \ldots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right).$$

---

# Adaboost

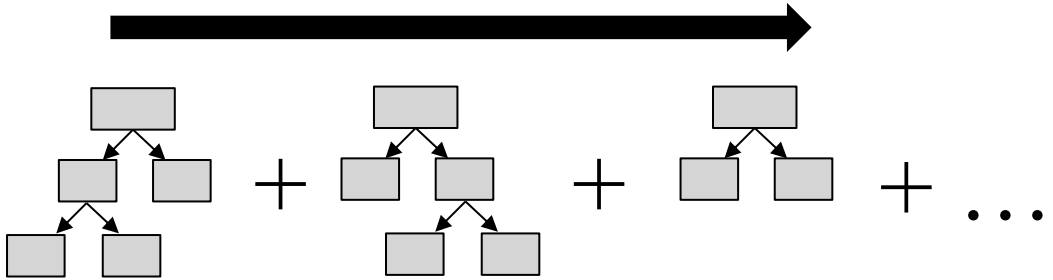
- Morally $D_t(i) \propto exp(L_{t-1}(i))$
- where $L_{t-1}(i) = \sum_{j=1}^{t-1} h_j(i) \neq Y_i$
- can be applied out of the box
- Can use gradient version: go toward new classifier a certain distance.

# Gradient Boosted Trees

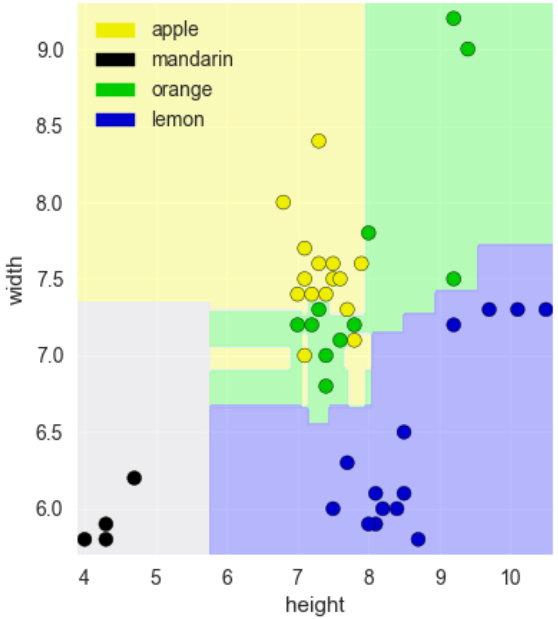
- **Algorithm:**
  - *Start with* $R_i^0 = Y_i - E[Y_i]$
  - *Learn* $f^t$ *on* $(X_i, R_i^t)$
  - *Let* $R_i^{t+1} = R_i^t - \alpha f^t(X_i)$
  - *Return* $F(X_i) = E[Y_i] + \alpha \sum_t f^t(X_i)$
- **Parameters:**
  - $\alpha$ = *stepsize*
  - *T = number of rounds*
  - *Weak learner*

# Gradient Boosted Decision Trees (GBDT)

- Training builds a series of small decision trees.
- Each tree attempts to correct errors from the previous stage.



- The learning rate controls how hard each new tree tries to correct remaining mistakes from previous round.
  - High learning rate: more complex trees
  - Low learning rate: simpler trees




GBDT decision regions on two-feature fruits dataset

# GBDT: Pros and Cons

Pros:

- Good accuracy
- Prediction is fast.
- Minimal but some tuning
- Handles a mixture of feature types.

Cons:

- Hard to interpret.
- learning rate and number estimators must be tuned
- Training requires significant computation.
- Not good for very high dimensional sparse features.

# XGBoost

- **Similar to Gradient Boosting but:**
  - *Tree weights are regularized*
  - *Other performance optimizations*

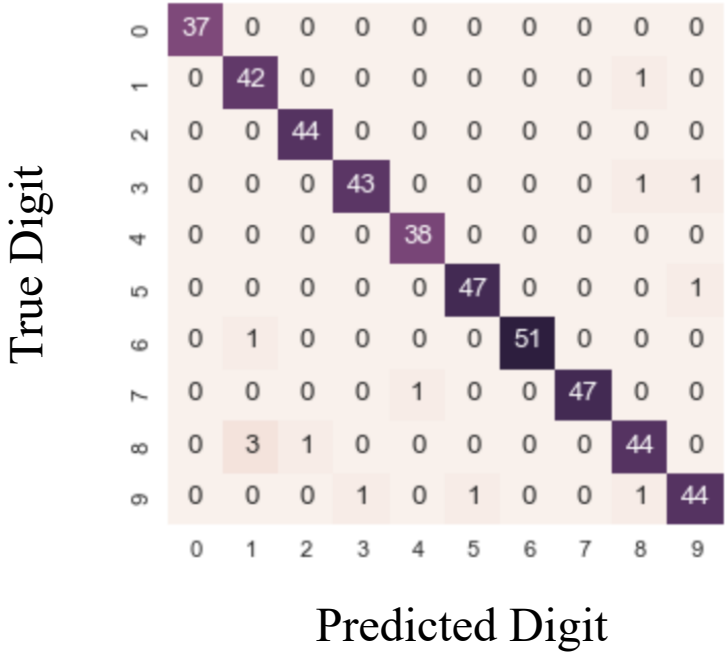
# Multi-Class Logistic Regression

- **Two models:**
  - *One versus Rest*
    - *Computes X versus not X separately*
    - *Implemented by SciKitLearn*
  - *One versus One*
    - *Computes X versus Y for all Y. Takes most likely X by summing over all outputs.*
  - *Often a matter of performance?*
    - *Which is faster?*
- **Some classifiers generalize naturally:**
  - *k-NN, decision trees*

# Multi-Class Evaluation

- Confusion matrices are especially useful
  - *Classification report*
- Multi-label classification: each instance can have multiple labels
  - *Can train on each label independently*
- Overall evaluation metrics are averages across classes
  - *How to average?*
  - *The size of classes is important*

# Multi-Class Confusion Matrix



# Micro vs Macro Average

| Class | Predicted Class | Correct? |
|---|---|---|
| orange | lemon | 0 |
| orange | lemon | 0 |
| orange | apple | 0 |
| orange | orange | 1 |
| orange | apple | 0 |
| lemon | lemon | 1 |
| lemon | apple | 0 |
| apple | apple | 1 |
| apple | apple | 1 |

**Micro-average**:

- Each instance has equal weight.
- Largest classes have most influence

1. Aggregate outcomes across all classes
2. Compute metric with aggregate outcomes

Micro-average accuracy:
4 / 9 = **0.44**

# Micro vs Macro Average

| Class | Predicted Class | Correct? |
|---|---|---|
| orange | lemon | 0 |
| orange | lemon | 0 |
| orange | apple | 0 |
| orange | orange | 1 |
| orange | apple | 0 |
| lemon | lemon | 1 |
| lemon | apple | 0 |
| apple | apple | 1 |
| apple | apple | 1 |

**Macro-average**:

- Each class has equal weight.

1. Compute metric within each class
2. Average resulting metrics across classes

| Class | Accuracy |
|---|---|
| orange | 1/5 = 0.20 |
| lemon | 1/2 = 0.50 |
| apple | 2/2 = 1.00 |

Macro-average accuracy:
(0.20 + 0.50 + 1.00) / 3 = **0.57**

# Macro-Average vs Micro-Average

- **Classes all same size?**
- **If some larger classes:**
  - *Weight toward largest classes, use micro-averaging.*
  - *Weight toward smaller classes, use macro-averaging.*
- **micro-average << macro-average examine performance on large classes.**
- **macro-average << micro-average examine performance on small classes.**

# Lab

- Can sit at tables
- Limited number of seats
- Only those at table can communicate together.
- Please seek us (Grant, Teng, Zhuofeng) out for help.
  - *Can search and locate us.*