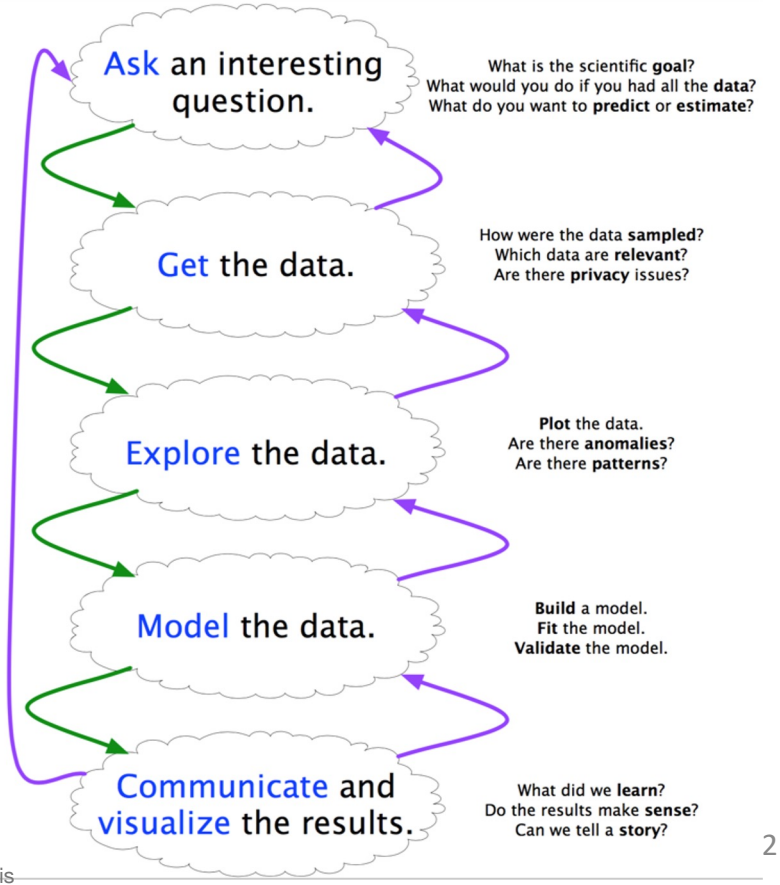
Introduction to Data Science and Analytics (DSC510)

Machine Learning

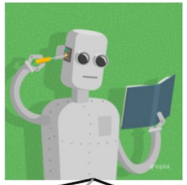
Why ML as part of data science?

- ML can facilitate most steps of the data analysis cycle



Taxonomy of Machine Learning

Taxonomy of Machine Learning



Labeled Data

Reward

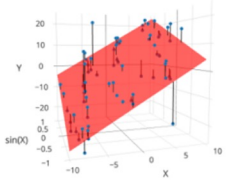
Unlabeled Data

Supervised Learning

Quantitative Response

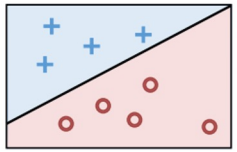
Categorical Response

Regression



OLS; linear regression w/regularization

Classification

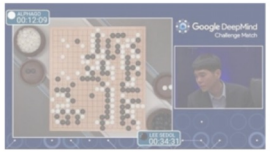


k-nearest neighbors

Logistic Regression

Decision Trees

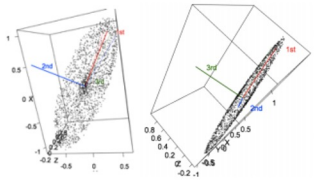
Reinforcement Learning (not covered)



Alpha Go

Unsupervised Learning

Dimensionality Reduction



PCA

Clustering

Machine learning

- **Supervised:** We are given input/output pairs (X, y) (a.k.a. “samples”) which we relate with a function y = f(X). We would like to “learn” f, and evaluate it on new data. Types:
  - **Classification:** y is discrete (class labels)
  - **Regression:** y is continuous, e.g., linear/logistic regression
- **Unsupervised:** Given only samples X of the data, we compute a function f such that y = f(X) is a “simpler” representation.
  - Discrete y: **“clustering”**
  - Continuous y: **“dimensionality reduction”**

Machine learning: examples

- **Supervised (*today*):**
  - Is this image a cat, dog, car, house?
  - How would this user rate that restaurant?
  - Is this email spam?
  - Is this blob a supernova?
- **Unsupervised (*later*):**
  - Cluster handwritten digit data into 10 classes
  - What are the top 20 topics in Twitter right now?
  - Find the best 2D visualization of 1000-dimensional data

Machine learning: techniques

- **Supervised Learning:**
  - k-NN (k nearest neighbors)
  - Naïve Bayes
  - Linear + logistic regression
  - Support vector machines
  - Random forests
  - Supervised neural networks
  - etc.
- **Unsupervised Learning:**
  - Clustering
  - Dimensionality reduction: topic modeling, matrix factorization (PCA, SVD, word2vec)
  - Hidden Markov models (HMM)
  - etc.

Criteria

Predictive performance (accuracy, AUC/ROC, precision, recall, F1-score, etc.)

Speed and scalability

- Time to build the model
- Time to use the model
- In memory vs. on disk processing
- Communication cost

Robustness

- Handling noise, outliers, missing values

Interpretability

- Understanding the model and its decisions (black box vs. white box)

Compactness of the model

- Mobile and embedded devices

Classification

training set

classification algorithm

classifier (model)

NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

categorical attributes

numerical attribute

class label

IF rank = “professor”
AND years > 6
THEN tenured = “yes”

Intro to supervised learning: k nearest neighbors (k-NN)

k nearest neighbors (k-NN)

Given a query item:

Find k closest matches

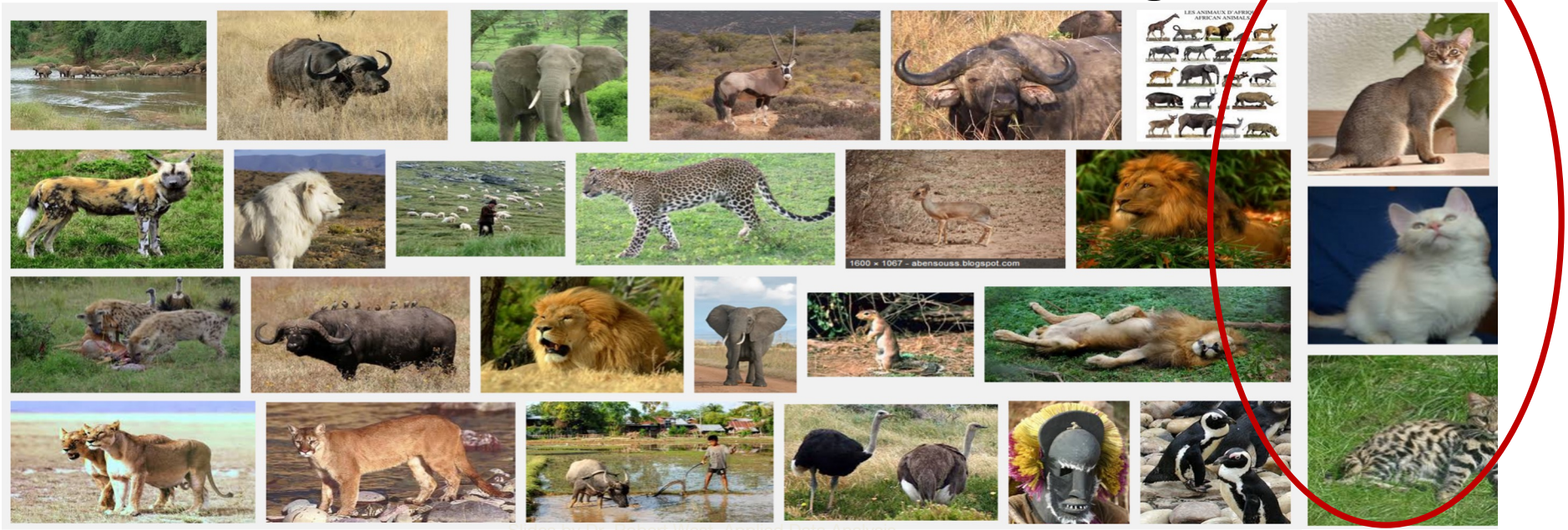
in a labeled dataset ↓

k nearest neighbors (k-NN)

Given a query item:

Find k closest matches

Return the most frequent label among the k



k nearest neighbors (k-NN)

k = 3 votes for “cat”

k-NN issues

The data is the model

- No training needed.
- Accuracy generally improves with more data.
- Matching is simple and fairly fast if data fits in memory.
- Usually need data in memory, but can be run off disk.

Minimal configuration:

- Only parameter is k (number of neighbors)
- But two other choices are important:
  - Weighting of neighbors (e.g. inverse distance)
  - Similarity metric

k-NN flavors

Classification:

- Model is y = f(X), y is from a discrete set (labels).
- Given X, compute y = majority vote of the k nearest neighbors.
- Can also use a weighted vote* of the neighbors.

Regression:

- Model is y = f(X), y is a real value.
- Given X, compute y = average value of the k nearest neighbors.
- Can also use a weighted average* of the neighbors.

* Weight function is usually the inverse distance.

k-NN distance measures

- **Euclidean Distance:** Simplest, fast to compute

$$d(x, y) = \|x - y\|$$

- **Cosine Distance:** Good for documents, images, etc.

$$d(x, y) = 1 - \frac{x \cdot y}{\|x\|\|y\|}$$

- **Jaccard Distance:** For set data:

$$d(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

- **Hamming Distance:** For string data:

$$d(x, y) = \sum_{i=1}^{n} (x_i \neq y_i)$$

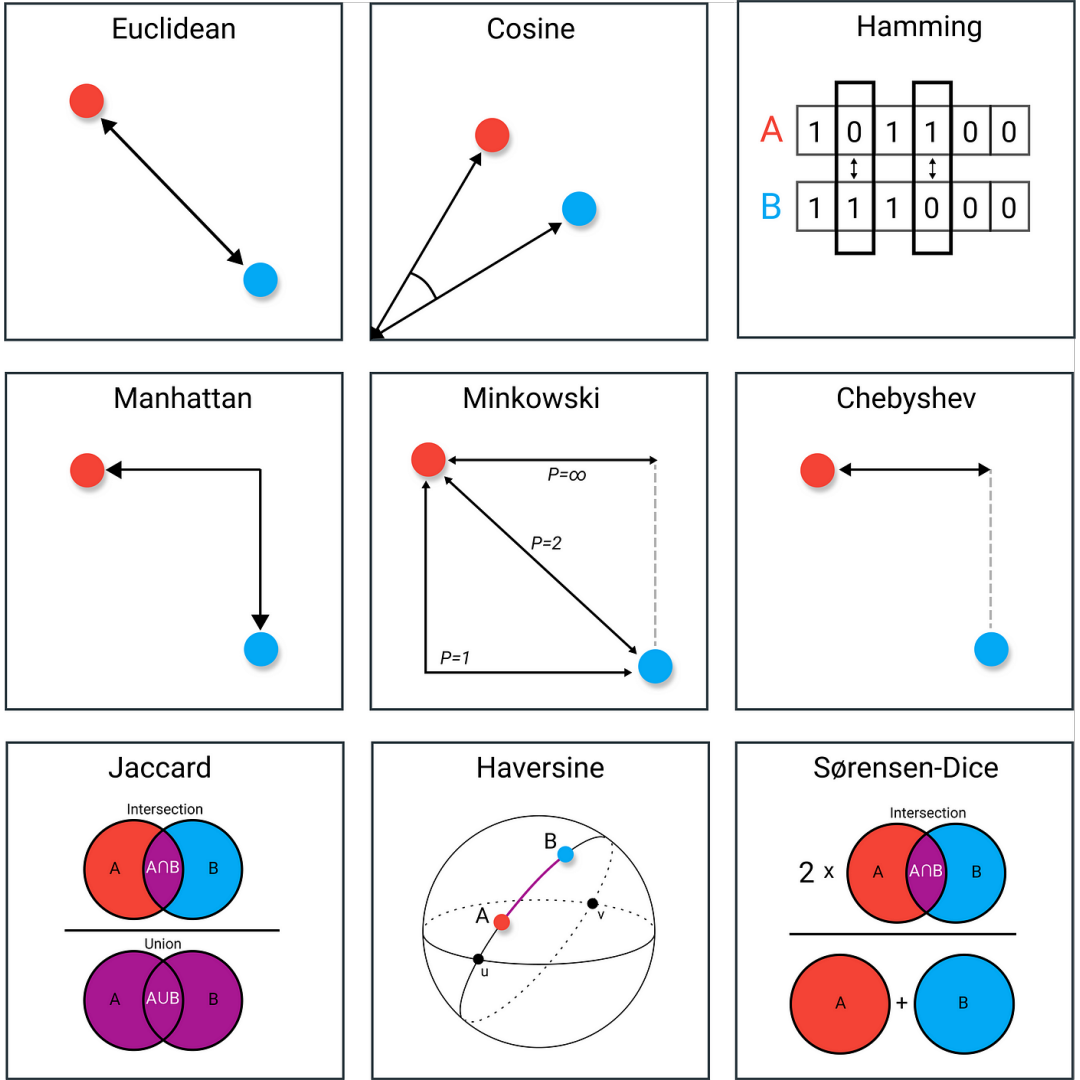
k-NN distance measures

- **Manhattan Distance:** Coordinate-wise distance

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

- **Edit Distance:** for strings, especially genetic data.

- **Mahalanobis Distance:** Normalized by the sample covariance matrix – unaffected by coordinate transformations.


Euclidean
Cosine
Hamming
A 1 0 1 1 0 0
B 1 1 1 0 0 0
Manhattan
Minkowski
P=∞
P=2
P=1
Chebyshev
Jaccard
Intersection
A
A∩B
B
Union
A
A∪B
B
Haversine
B
A
v
u
Sørensen-Dice
Intersection
2 x
A
A∩B
B
A
+
B

Predicting from samples

- Most datasets are **samples** from an **infinite population**.
- We are most interested in **models of the population**, but we have access only to a **sample** of it.

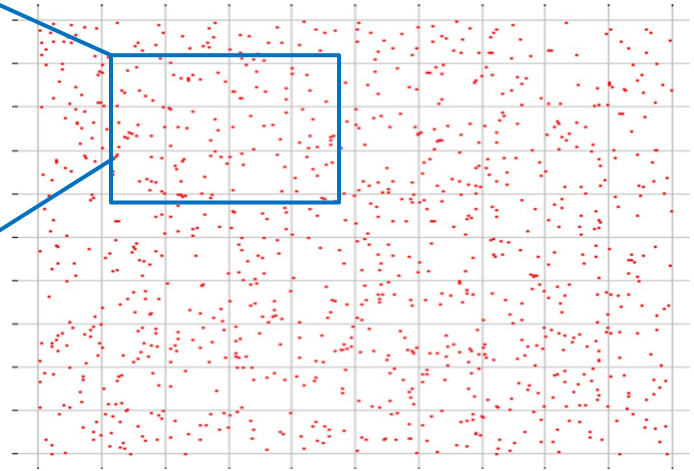
For datasets consisting of (X, y)

- features X, label y

For the true model f:

- $y = f(X)$

We train on a training sample D
and we denote the model as $f_D(X)$



Bias and variance

Our data-generated model $f_D(X)$ is a **statistical estimate** of the true function $f(X)$.

Because of this, its subject to bias and variance:

Bias: if we train models $f_D(X)$ on many training sets D, bias is the expected difference between their predictions and the true y's.

i.e. $$Bias = \mathrm{E}[f_D(X) - y]$$

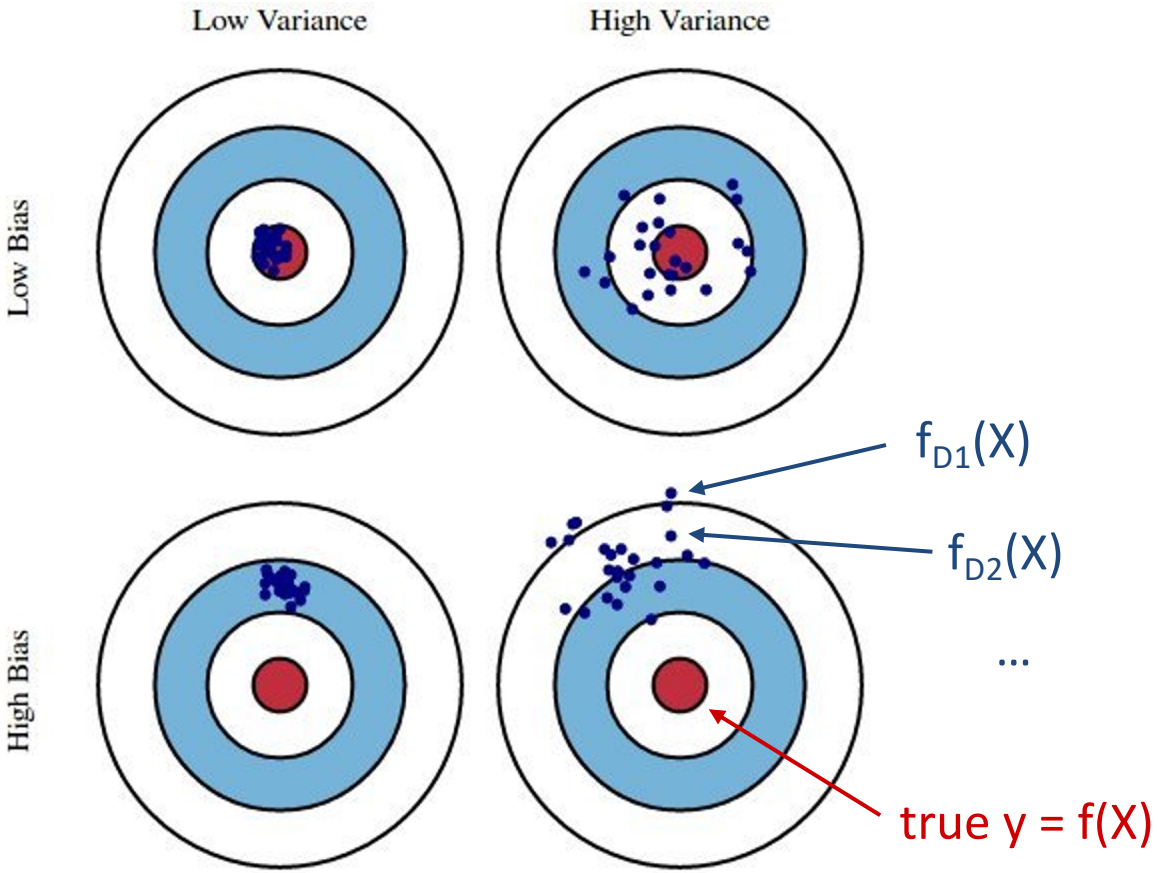
E[] is taken over points X and datasets D

Variance: if we train models $f_D(X)$ on many training sets D, variance is the variance of the estimates:

$$Variance = \mathrm{E}\left[\left(f_D(X) - \bar{f}(X)\right)^2\right]$$

Where $\bar{f}(X) = \mathrm{E}[f_D(X)]$ is the average prediction on X.

Consider a fixed X



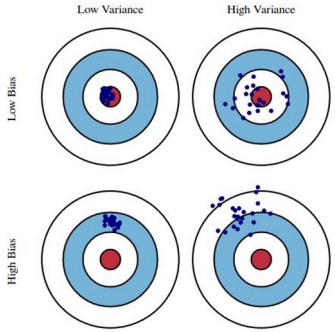
"Full" bias/variance: average this picture over all X

Bias/variance tradeoff

There is usually a bias-variance tradeoff caused by model complexity.

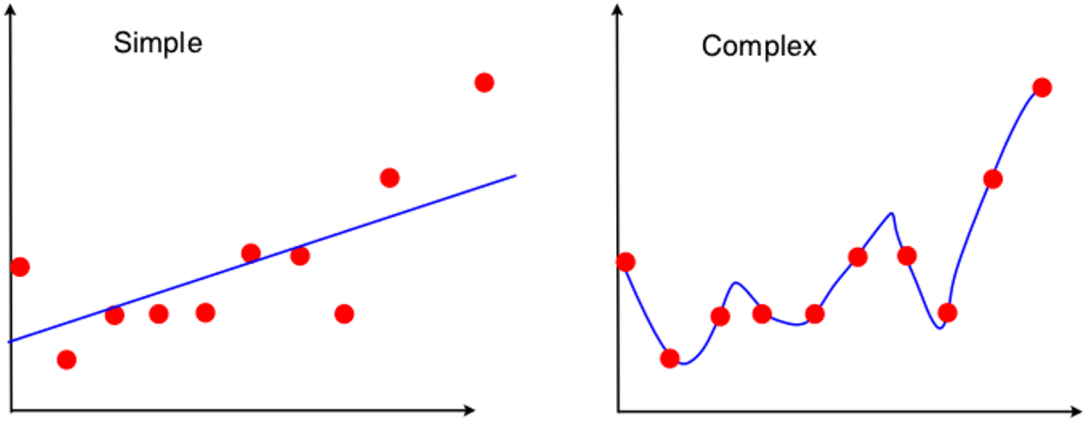
Complex models (many parameters) usually have lower bias, but higher variance.

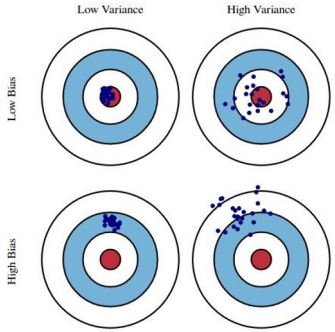
Simple models (few parameters) have higher bias, but lower variance.



Bias/variance tradeoff

e.g. a linear model can only fit a straight line. A high-degree polynomial can fit a complex curve. But the polynomial can fit the individual sample, rather than the population. Its shape can vary from sample to sample, so it has high variance.





Bias/variance tradeoff

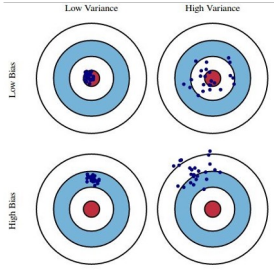
The total expected error is

$$Bias^2 + Variance$$

Because of the bias-variance trade-off, we want to **balance** these two contributions.

If $Variance$ strongly dominates, it means there is too much variation between models. This is called **over-fitting**.

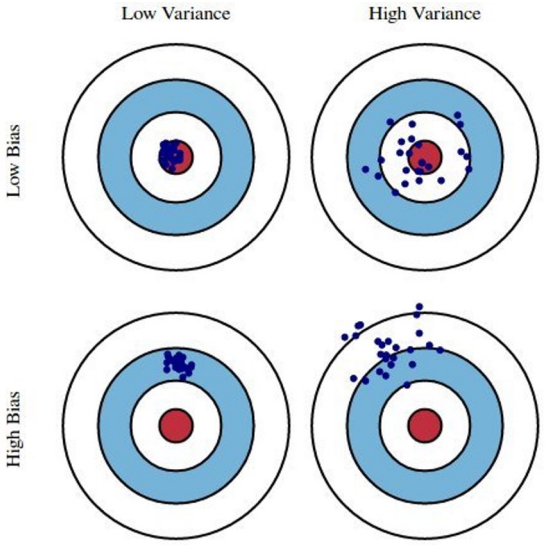
If $Bias$ strongly dominates, then the models are not fitting the data well enough. This is called **under-fitting**.



Choosing k for k nearest neighbors

We have a bias/variance tradeoff:

- Small k → ?
- Large k → ?

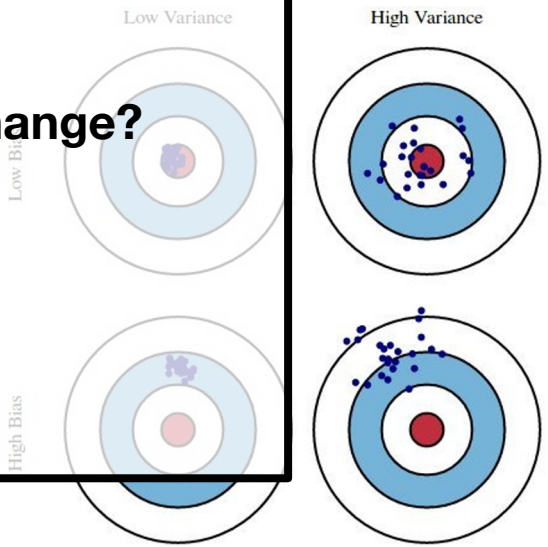


Choosing k for k nearest neighbors

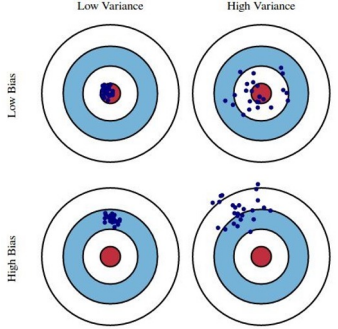
We have a bias/variance tradeoff:

- Small k → ?
- Large k → ?

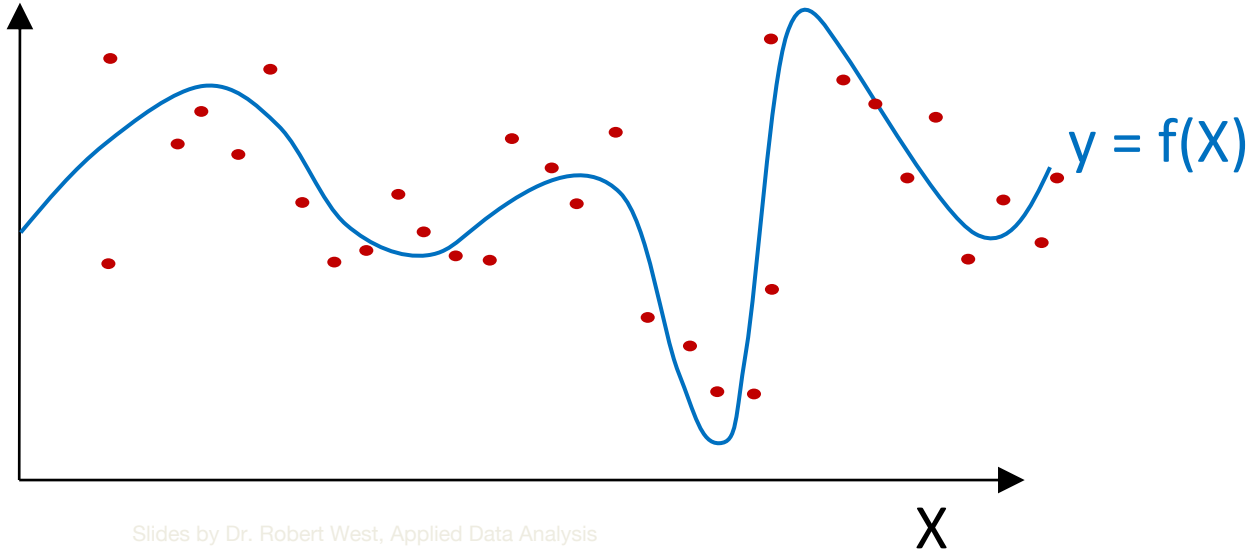
Think for 1 minute:
When *k* increases,
how do bias and variance change?



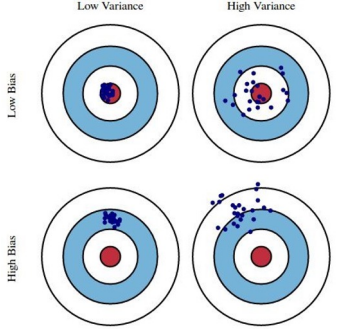
Choosing k



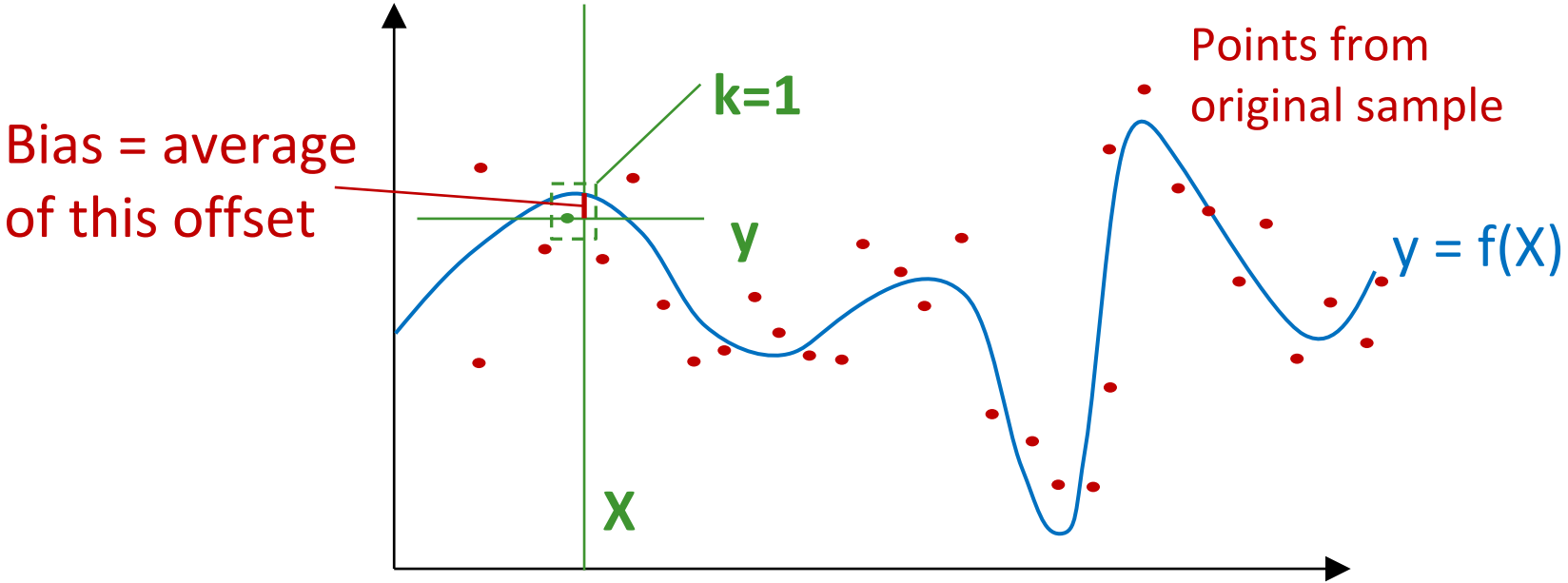
- Small k → low bias, high variance
- Large k → high bias, low variance
- Assume the real data follows the blue curve, with some mean-zero additive noise. Red points are a data sample.



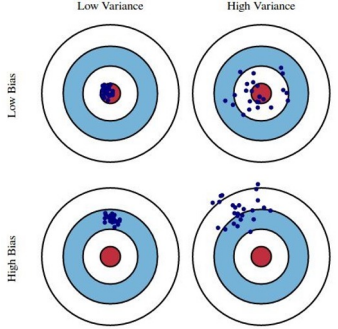
Choosing k



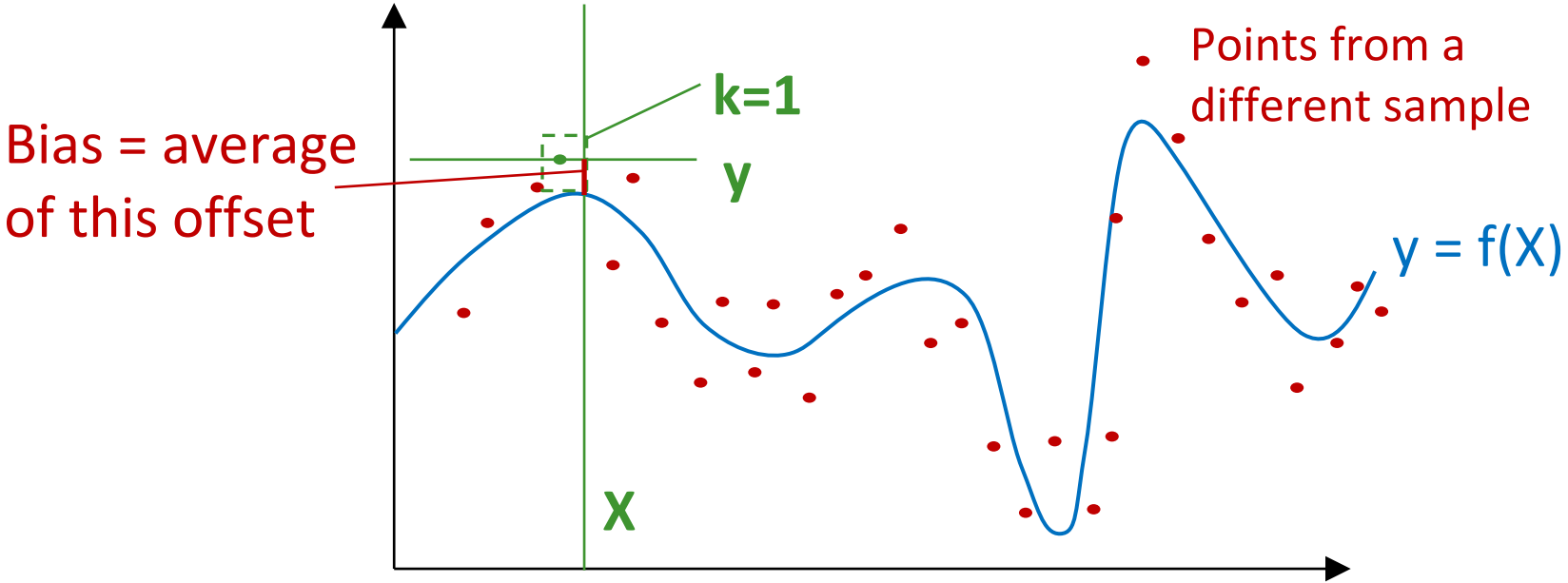
- **Small k** → low bias, high variance
- Large k → high bias, low variance



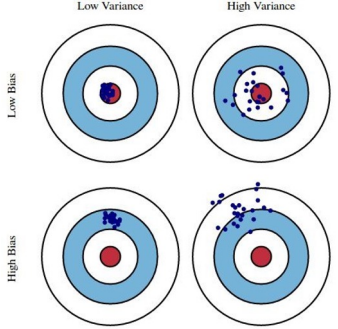
Choosing k



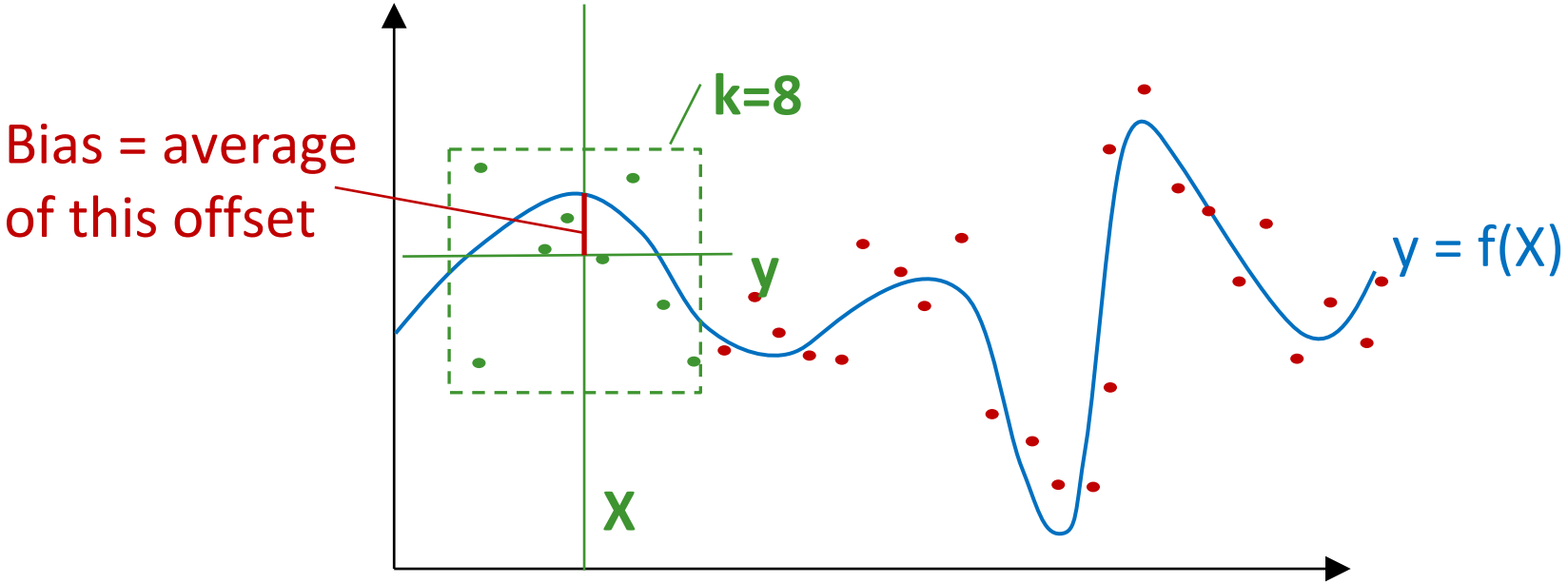
- **Small k** → low bias, high variance
- Large k → high bias, low variance



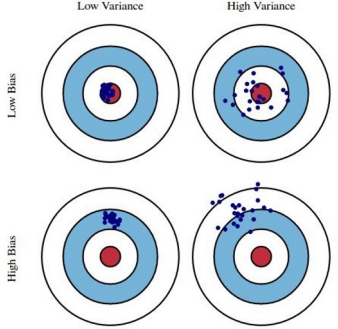
Choosing k



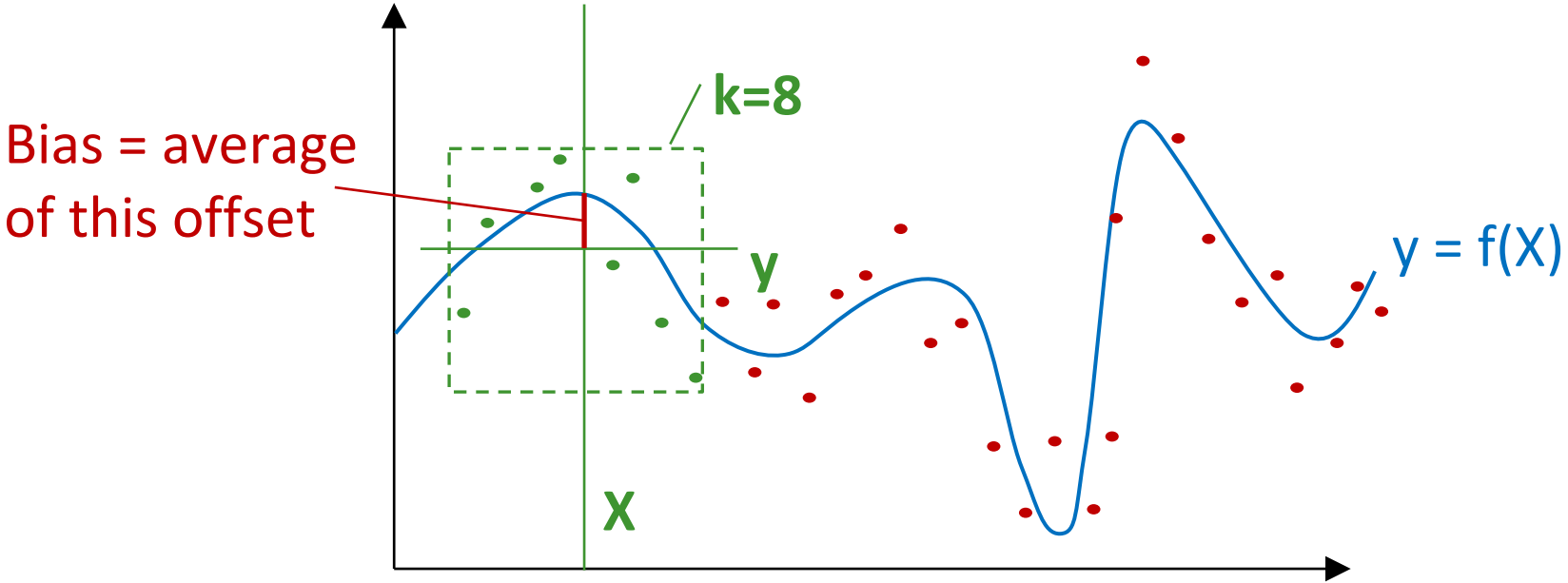
- Small k → low bias, high variance
- **Large k** → high bias, low variance



Choosing k

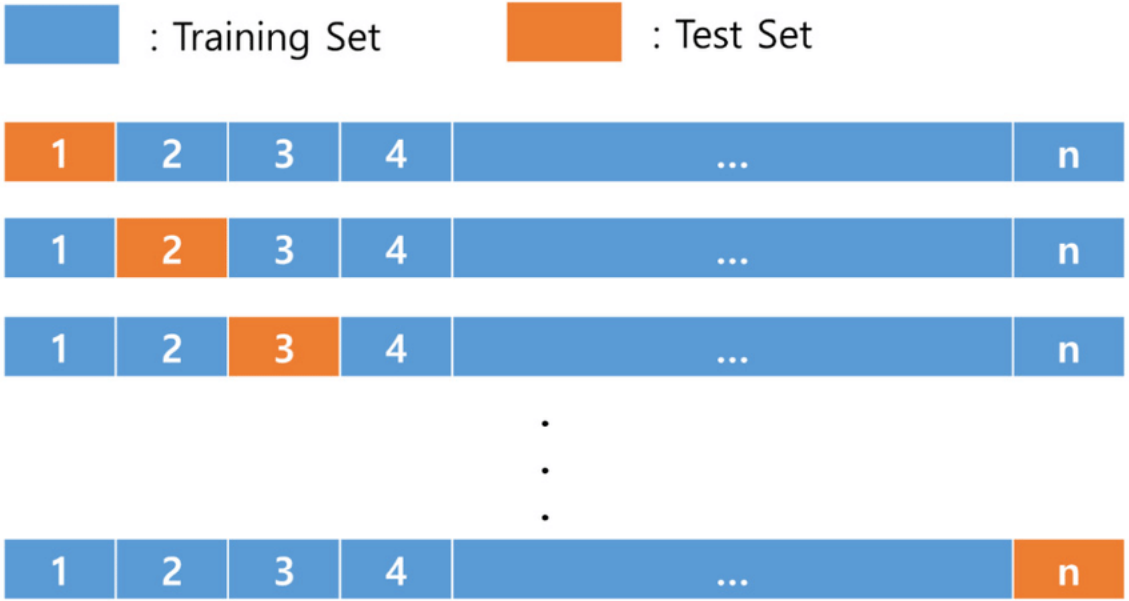


- Small k → low bias, high variance
- **Large k** → high bias, low variance



Choosing k in practice

Use leave-one-out (LOO) cross-validation:



K Fold Cross Validation

Choosing k in practice

- **Split:** Break data into train and test subsets, e.g. 80-20 % random split.
- **Predict:** For each point in the training set, predict using the k nearest neighbors from the set of all *other* points in training set. Measure the LOO error rate (classification) or squared error (regression).
- **Tune:** Try different values of k, and use the one that gives minimum leave-one-out error.
- **Evaluate:** Test on the test set to measure performance.

What is overfitting?

Overfitting is an undesirable machine learning behavior that occurs when the machine learning model gives accurate predictions for training data but not for new data. When data scientists use machine learning models for making predictions, they first train the model on a known data set.

What is underfitting?

Underfitting is another type of error that occurs when the model cannot determine a meaningful relationship between the input and output data. You get underfit models if they have not trained for the appropriate length of time on a large number of data points.

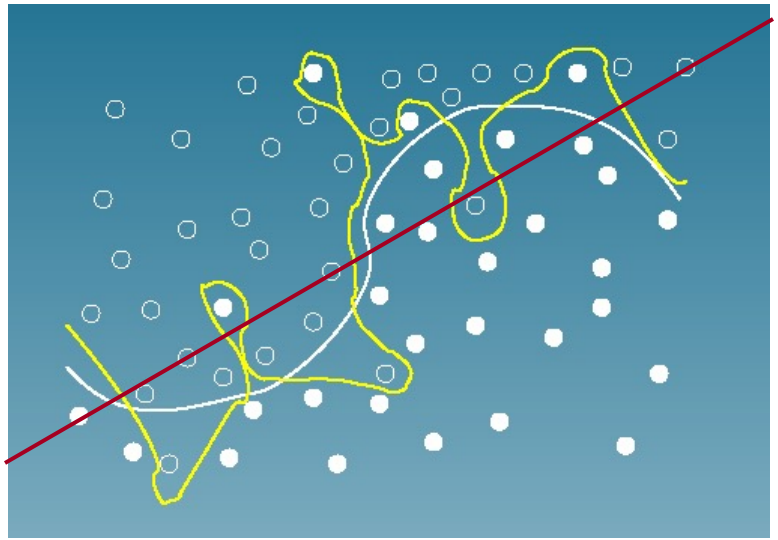
Overfitting vs Underfitting

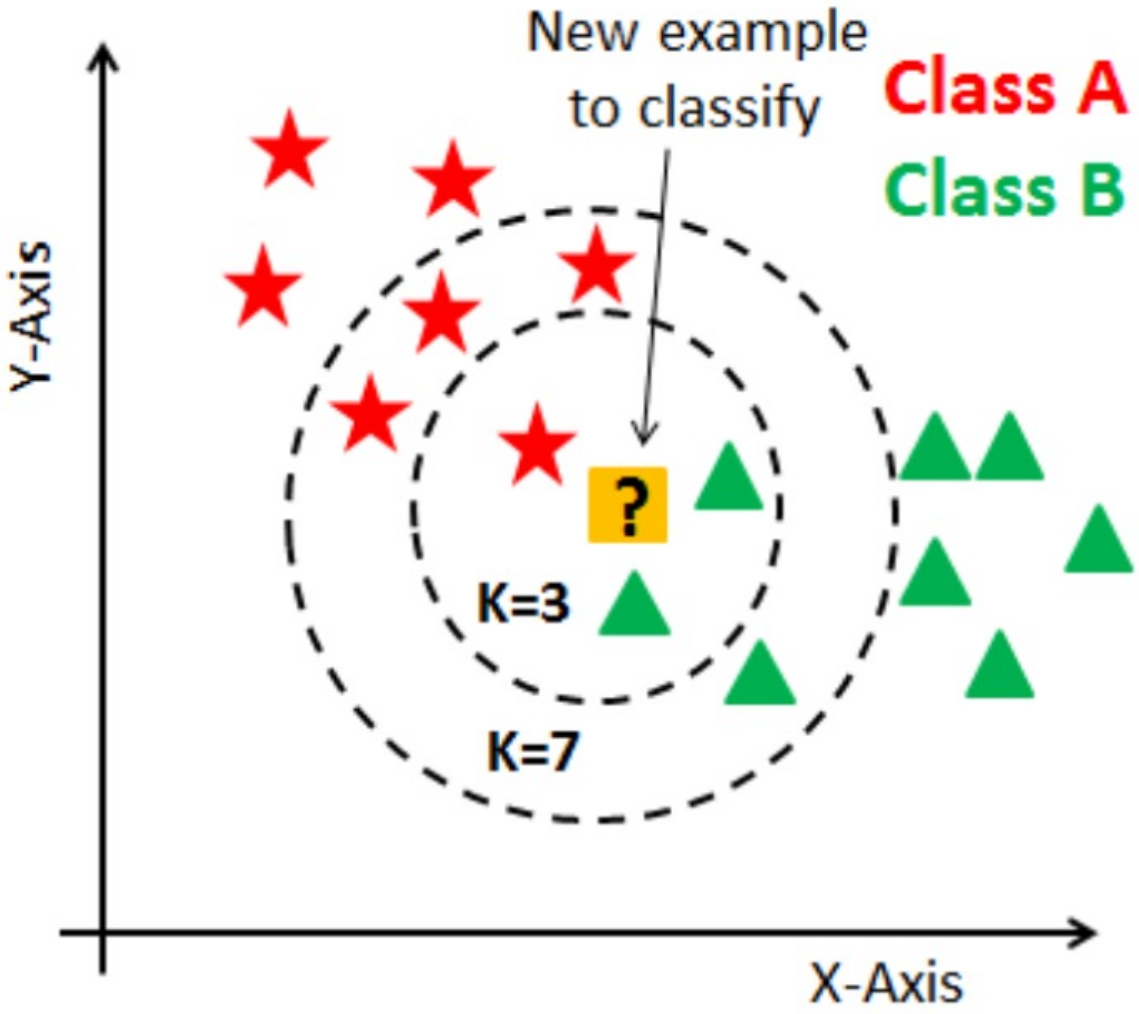
Underfit models experience **high bias**—they give inaccurate results for both the training data and test set.

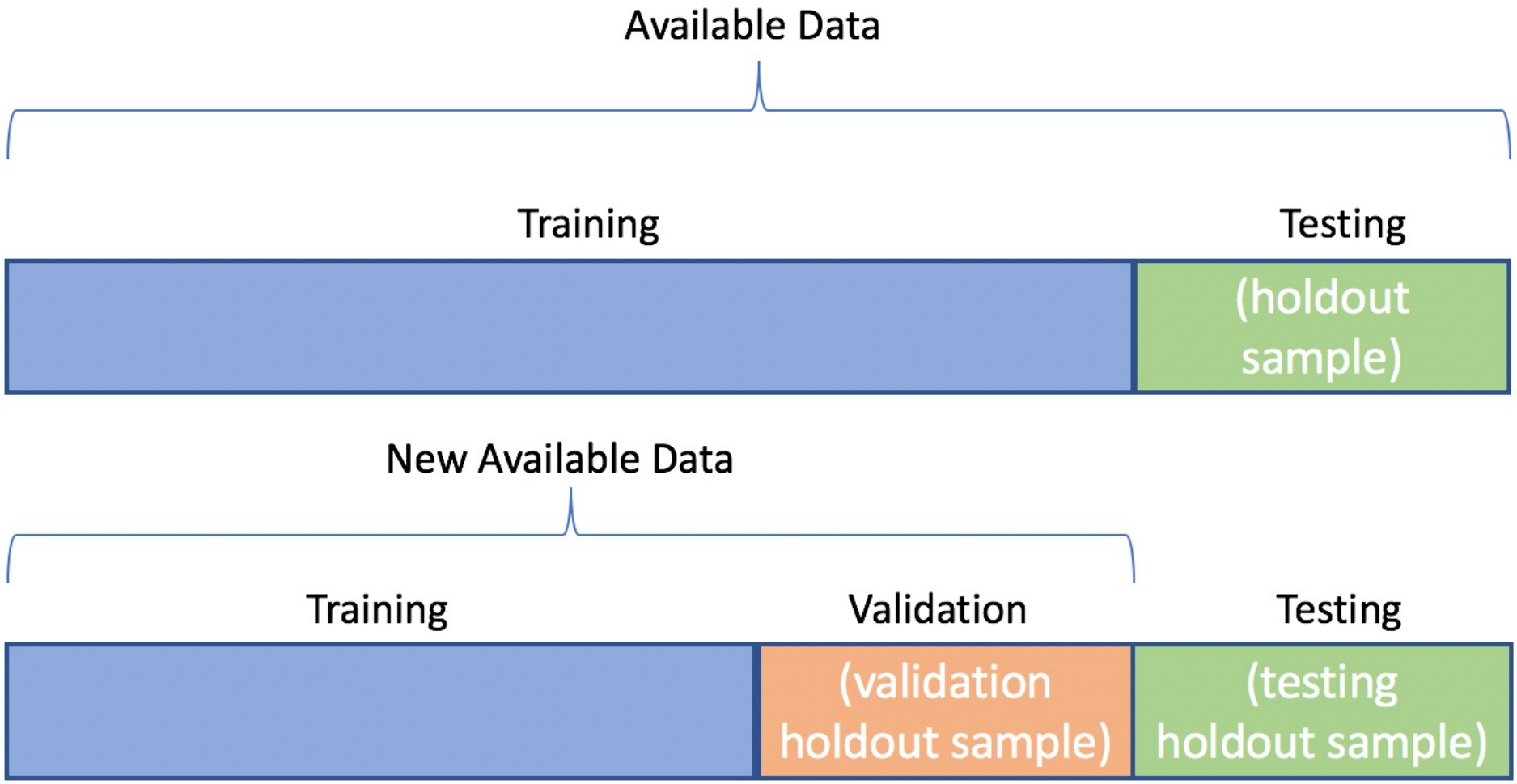
Overfit models experience **high variance**—they give accurate results for the training set but not for the test set. More model training results in less bias but variance can increase.

Data scientists aim to find the sweet spot between underfitting and overfitting when fitting a model. A well-fitted model can quickly establish the dominant trend for seen and unseen data sets.

Bias vs. variance:
Choosing the correct model capacity




New example
to classify
Class A
Class B
Y-Axis
?
K=3
K=7
X-Axis


Available Data
Training
Testing
(holdout sample)
New Available Data
Training
Validation
Testing
(validation holdout sample)
(testing holdout sample)

k-NN and the curse of dimensionality

The curse of dimensionality refers to “weird” phenomena that occur in high dimensions (100s to millions) that do not occur in low-dimensional (e.g. 3-dimensional) space.

In particular data in high dimensions are much sparser (less dense) than data in low dimensions.

For k-NN, this means there are fewer points that are very close in feature space (very similar) to the point X whose y we want to predict.

k-NN and the curse of dimensionality

From this perspective, it's surprising that kNN works at all in high dimensions.

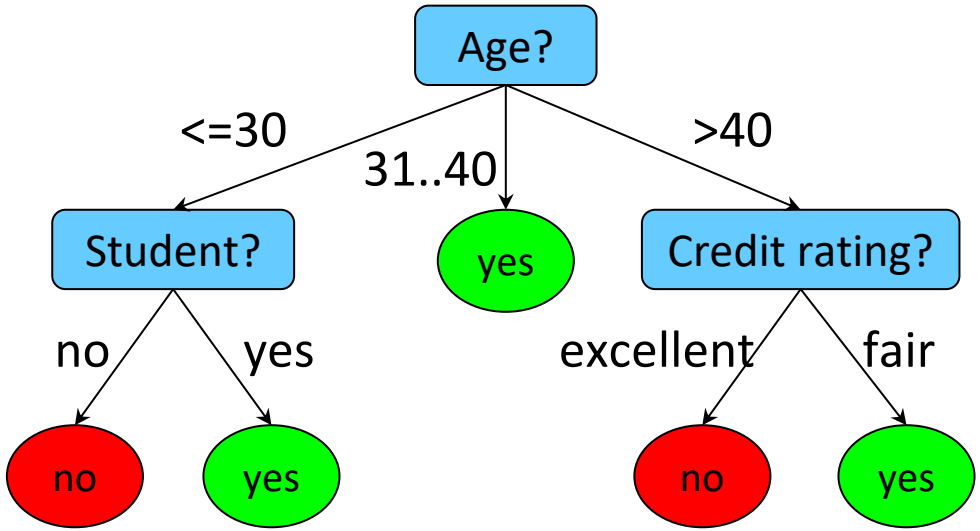
Luckily real data are not like random points in a high-dimensional cube. Instead they live in **dense clusters** and near **much lower-dimensional surfaces**.

Also, points can be very "similar" even if their Euclidean distance is large. E.g. documents with the same few dominant words are likely to be on the same topic (--> use different distance)

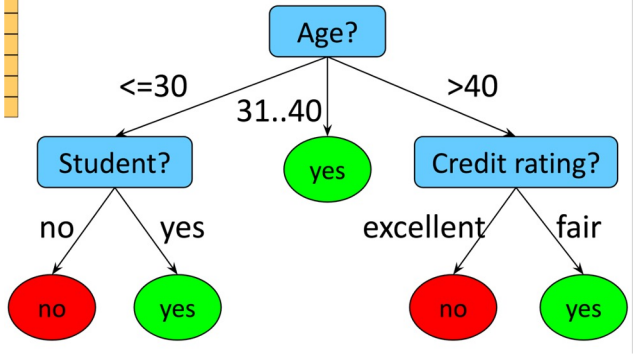
Decision trees

Decision trees: example

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no



Decision trees



Model: flow-chart-like tree structure

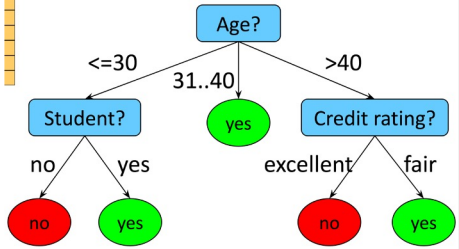
- Nodes are tests on a single attribute
- Branches are attribute values
- Leaves are marked with class labels

Score function: classification accuracy

Optimization:

- NP-hard
- Heuristic: greedy top-down tree construction + pruning

Decision tree induction



Tree construction (top-down divide-and-conquer strategy)

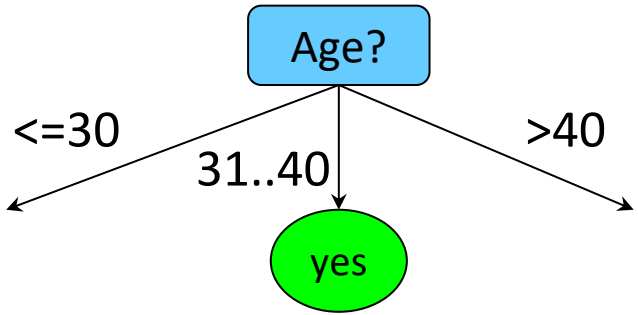
- At the beginning, all training samples belong to the root
- Examples are partitioned recursively based on selected “most discriminative” attributes
- Discriminative power based on information gain or Gini impurity (CART)

Partitioning stops if

- All samples belong to the same class → assign the class label to the leaf
- There are no attributes left → majority voting to assign the class label to the leaf

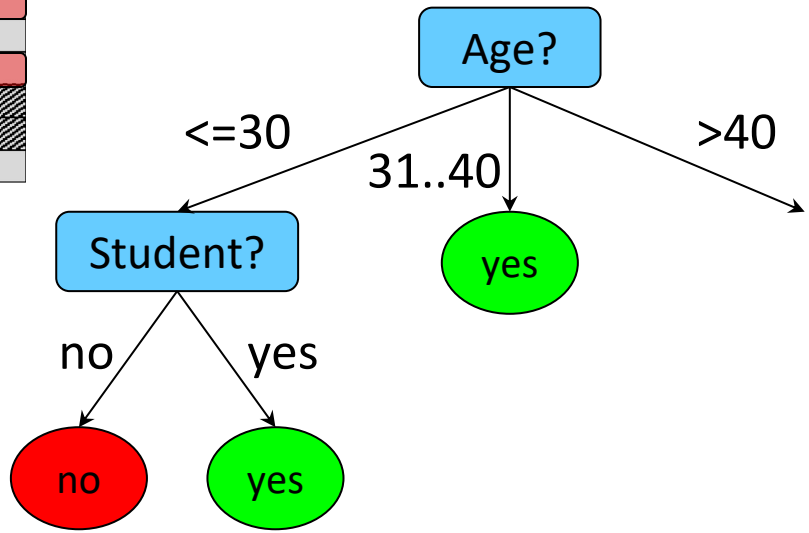
Decision tree induction

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no



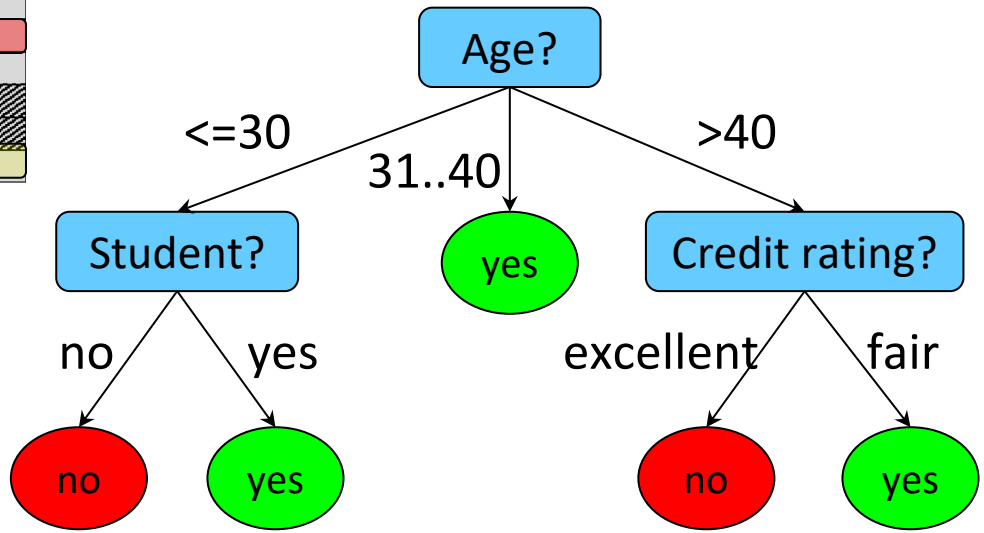
Decision tree induction

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



Decision tree induction

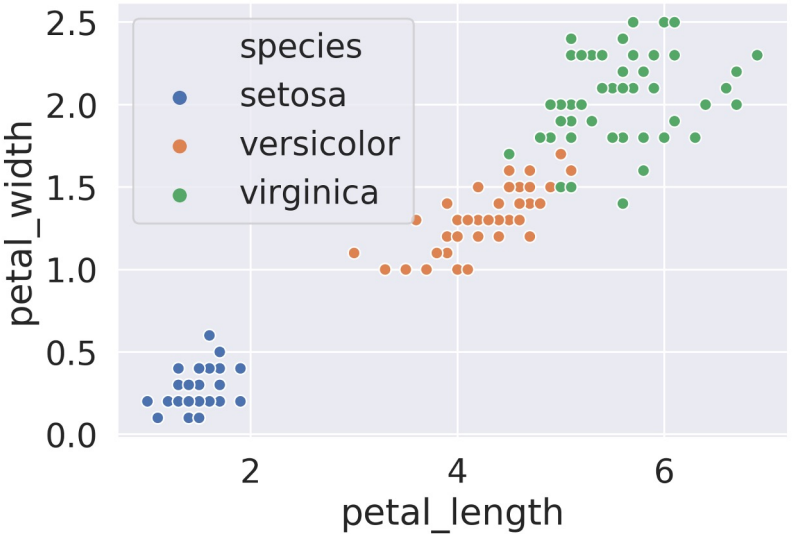
age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



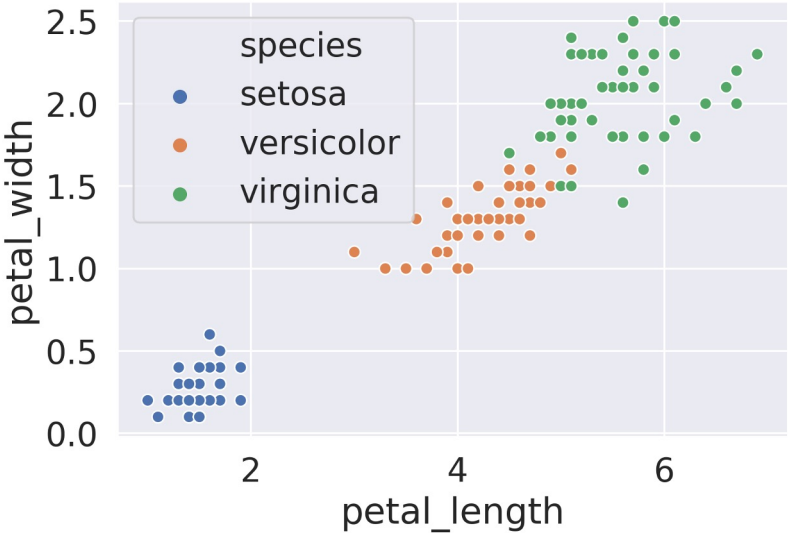
Example: Using Petal Data Only

The plot below shows the width and length of the petals of each flower, with the species annotated in the form of color.

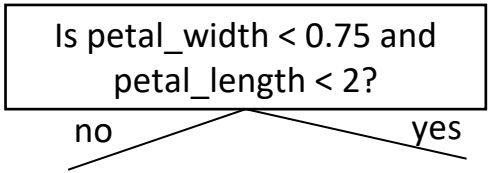
We can build a decision tree manually just by looking at this picture.

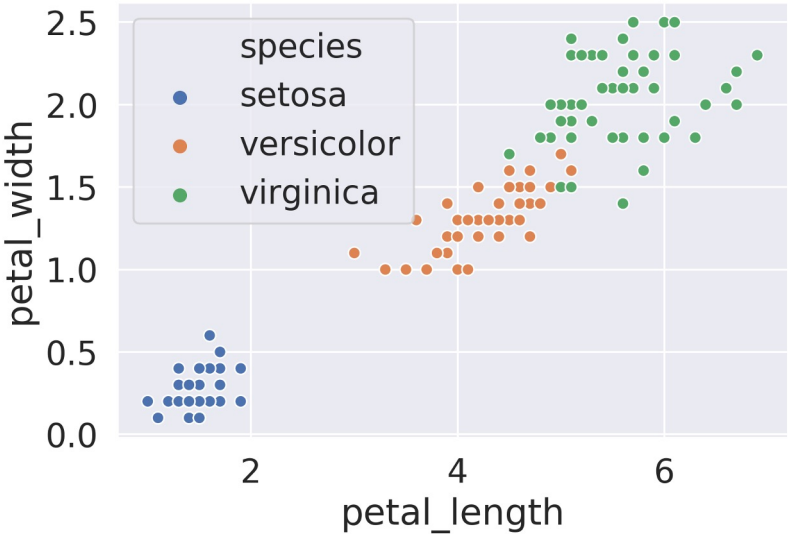


Example: Using Petal Data Only

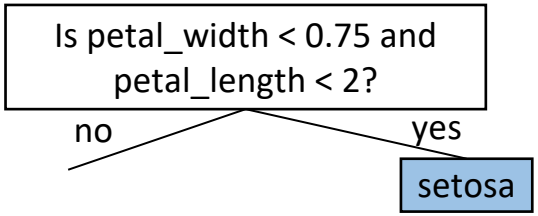


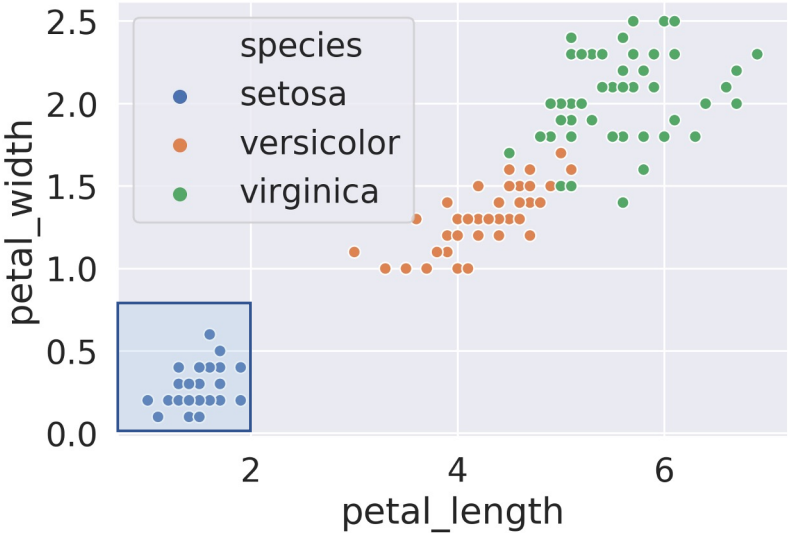
Example: Using Petal Data Only



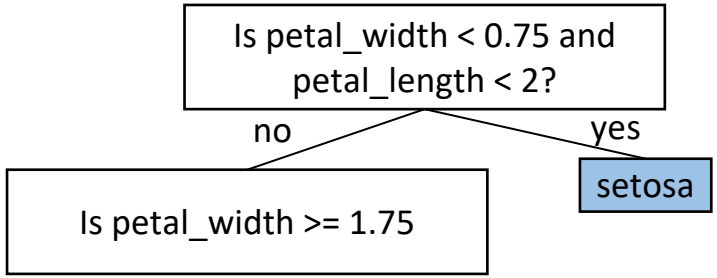


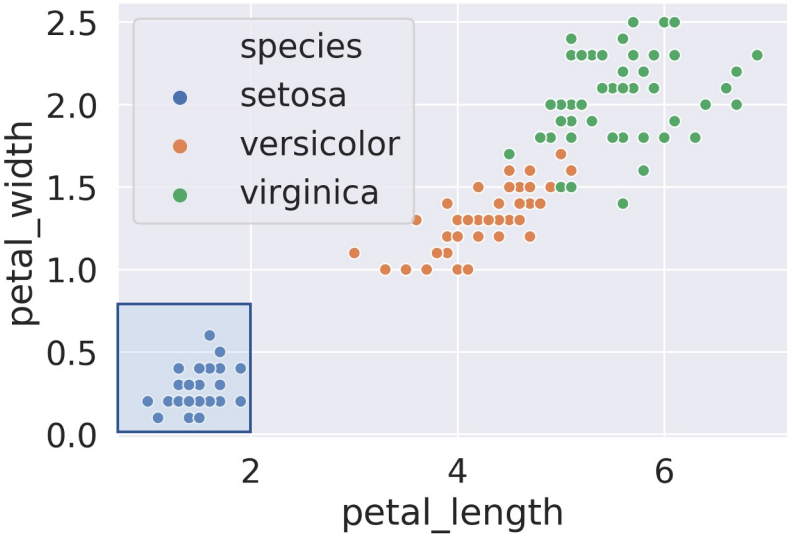
Example: Using Petal Data Only



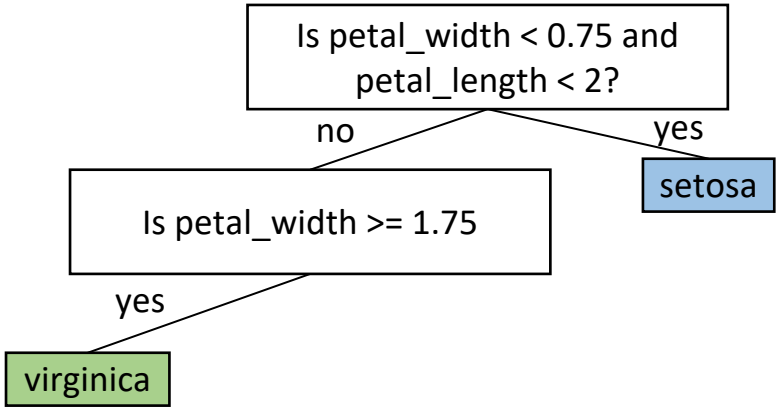


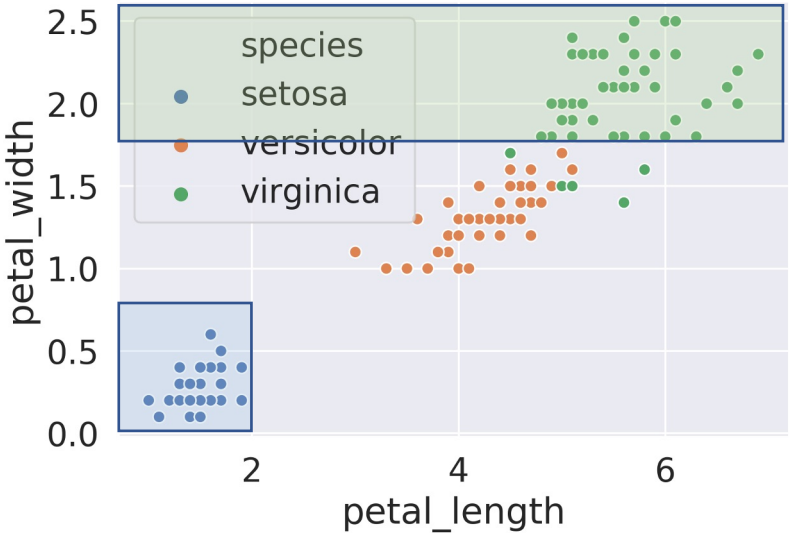
Example: Using Petal Data Only



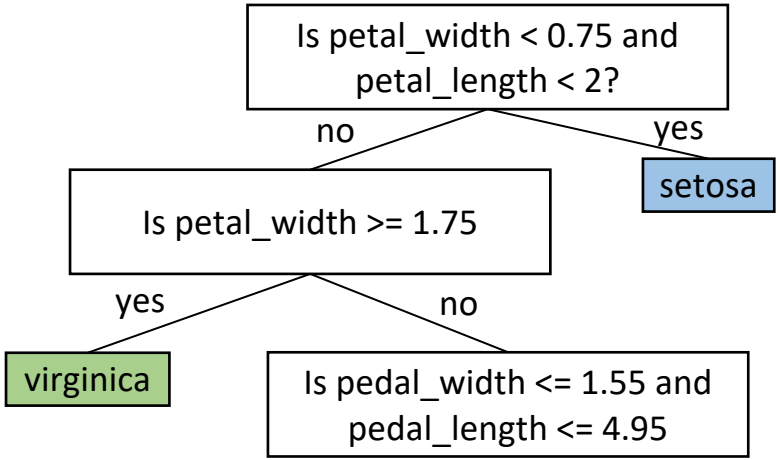


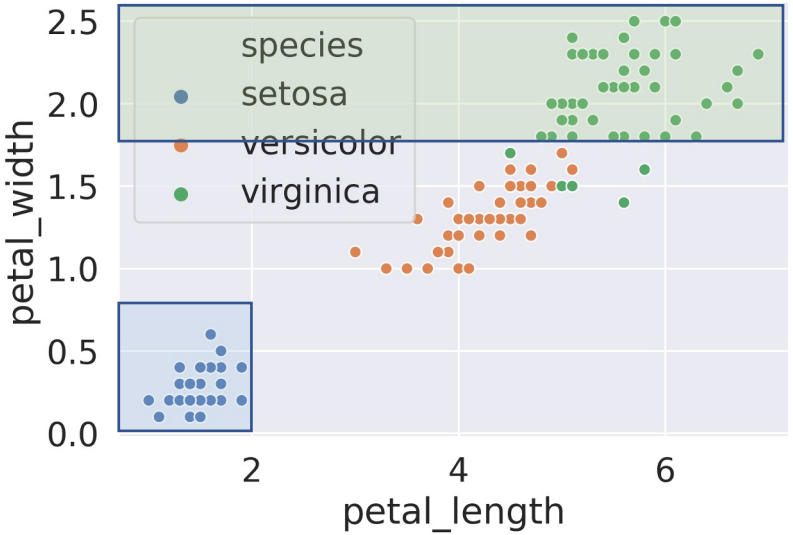
Example: Using Petal Data Only



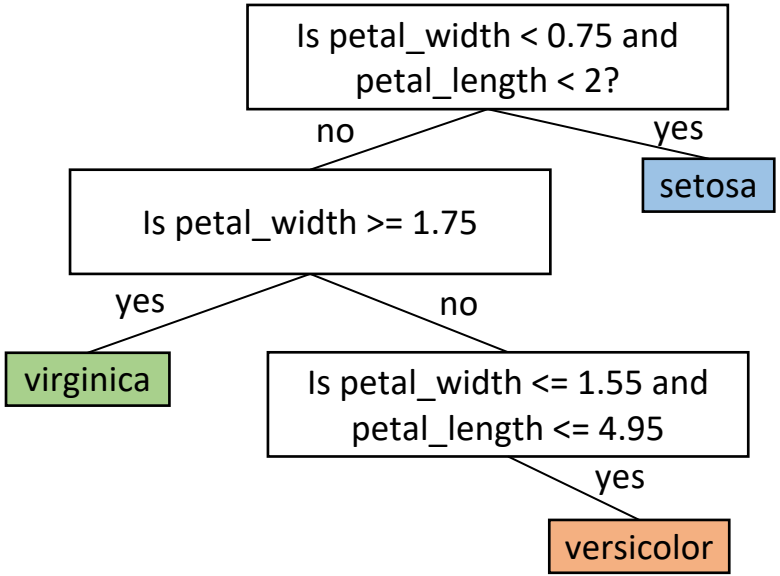


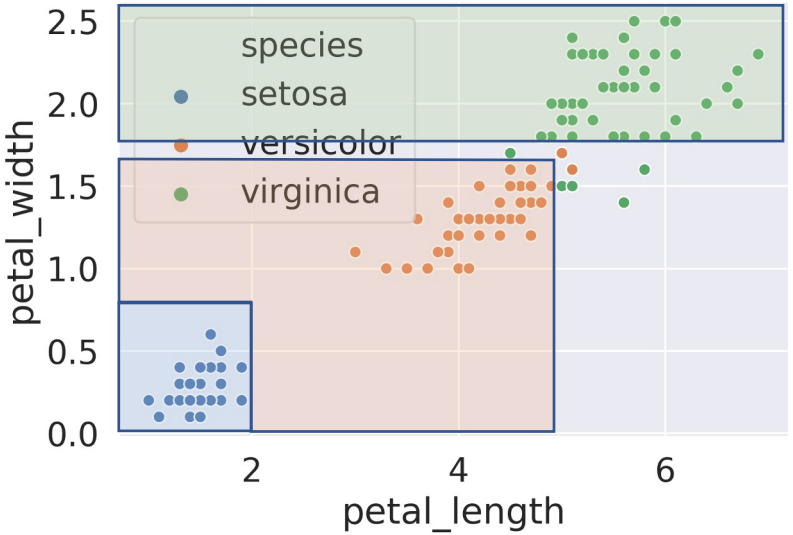
Example: Using Petal Data Only



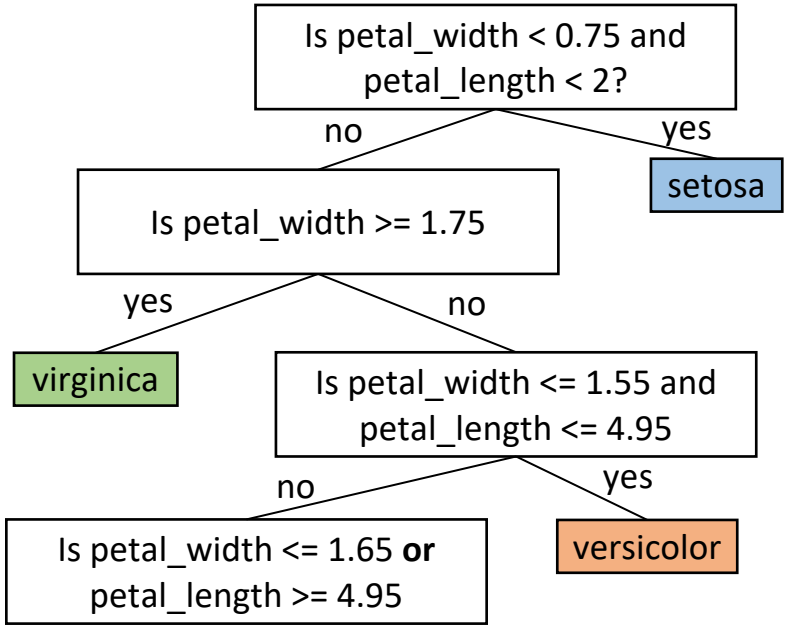


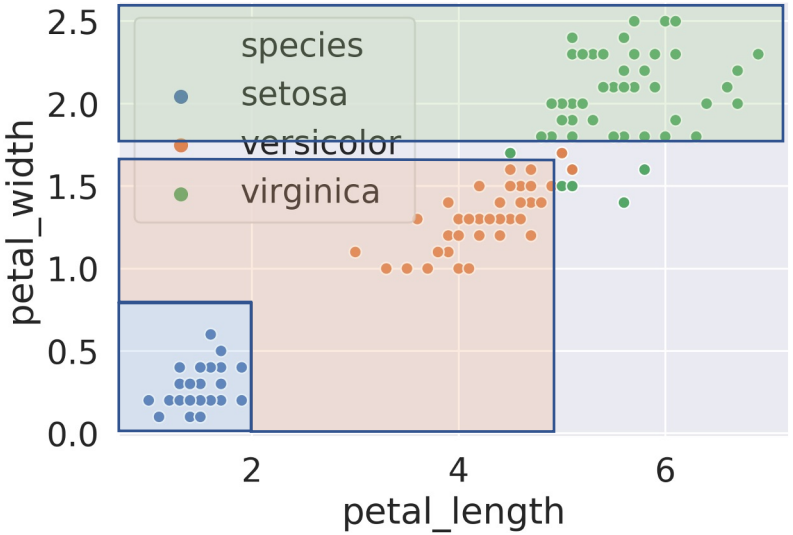
Example: Using Petal Data Only



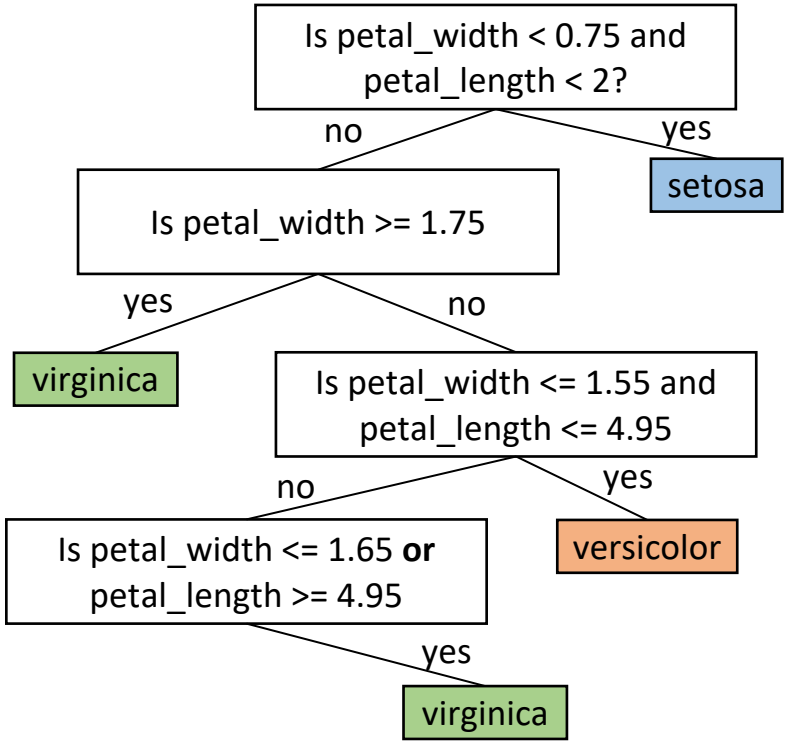


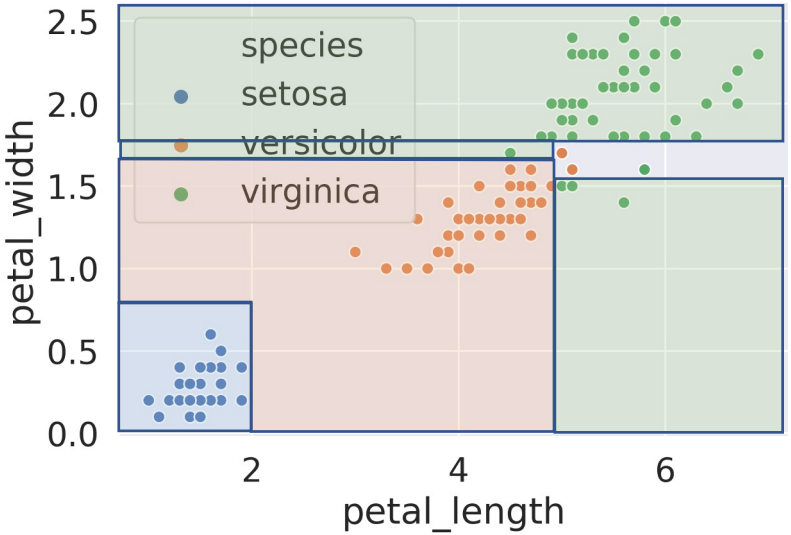
Example: Using Petal Data Only



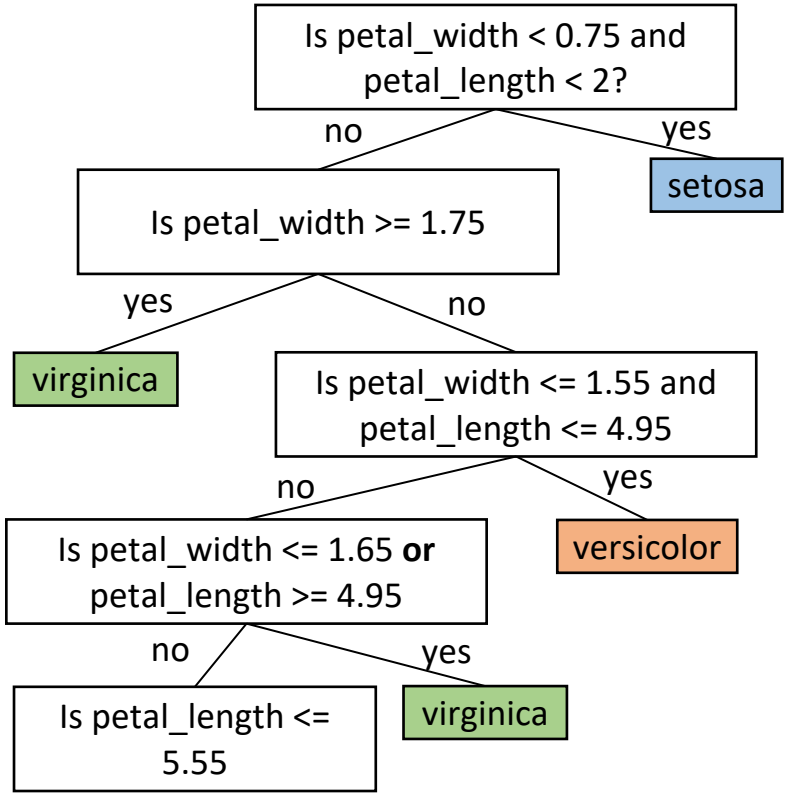


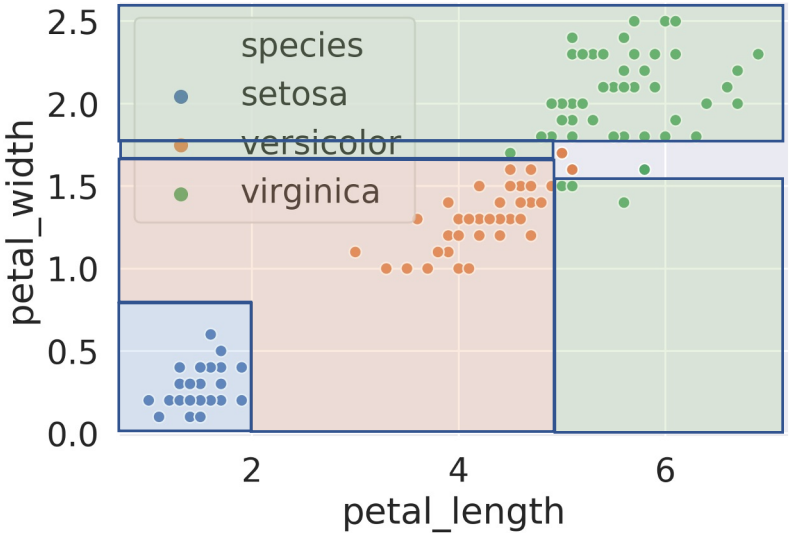
Example: Using Petal Data Only



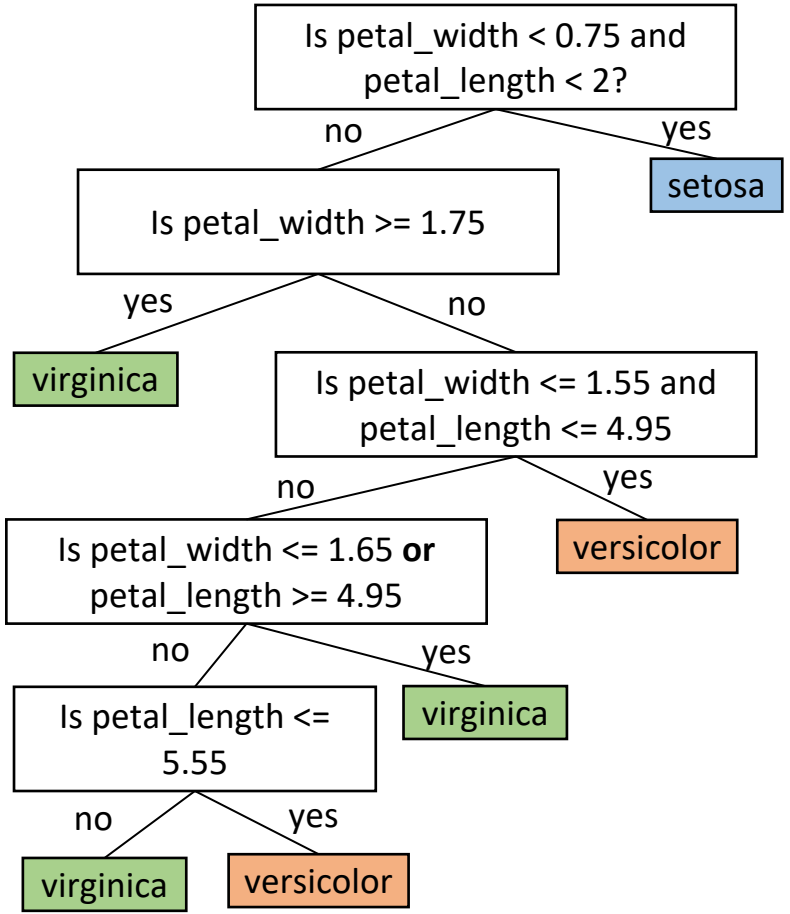


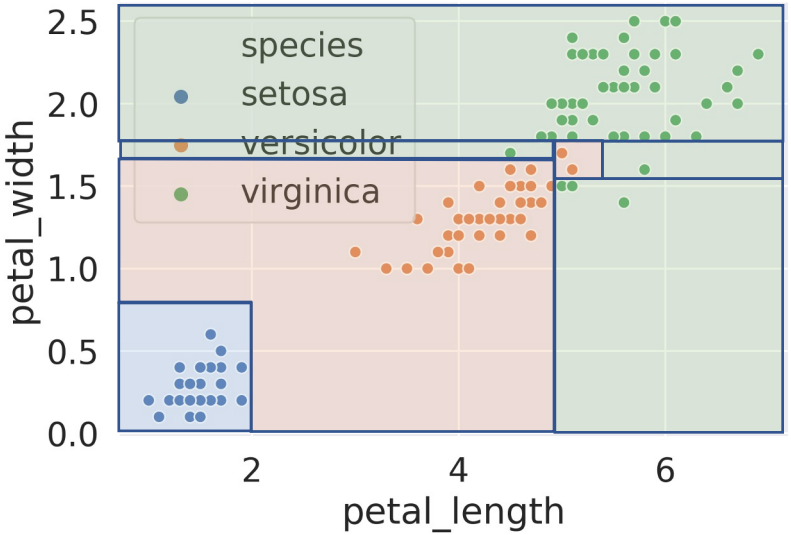
Example: Using Petal Data Only





Example: Using Petal Data Only





Sepal Decision Tree

Traditional decision tree generation algorithm:

- All of the data starts in the root node.
- Repeat until every node is either **pure** or **unsplittable**:
  - Pick the ***best feature*** x and ***best split value*** β, e.g. x = **“`petal_length`”**, β = 2.
  - *Split data into two nodes*, one where $x \leq \beta$, and one where $x > \beta$.

Pure? Unsplittable?

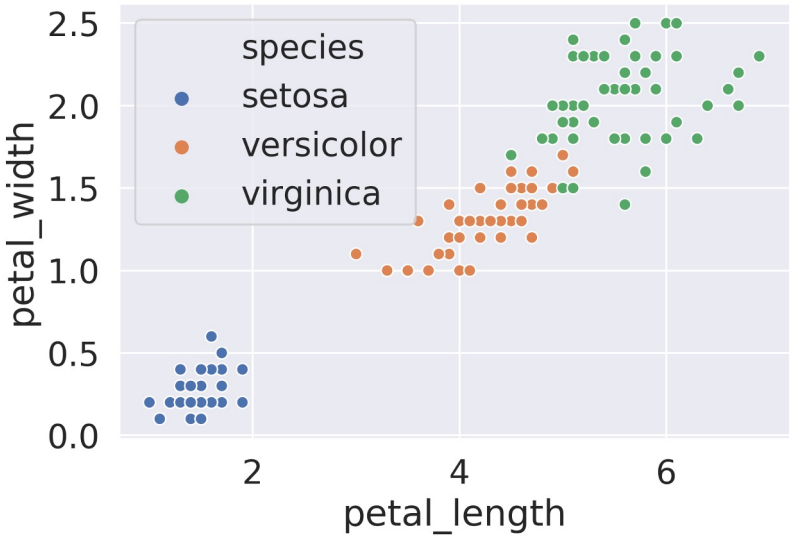
A node that has only samples from **one class** is called a “**pure**” node.

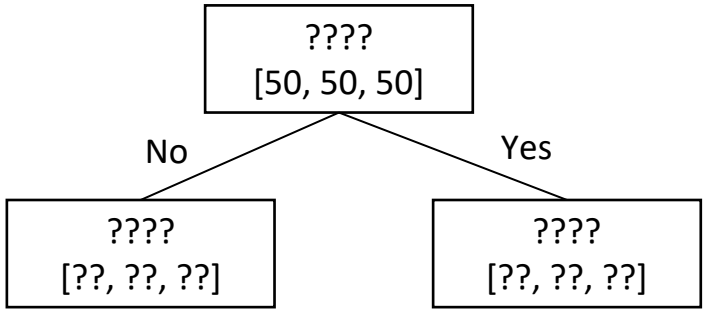
A node that has overlapping data points from different classes and thus that cannot be split is called “**unsplittable**”.

Defining a Best Feature

Question: Which feature and split value is best?

- Equivalently: Which horizontal or vertical line do we want to draw?

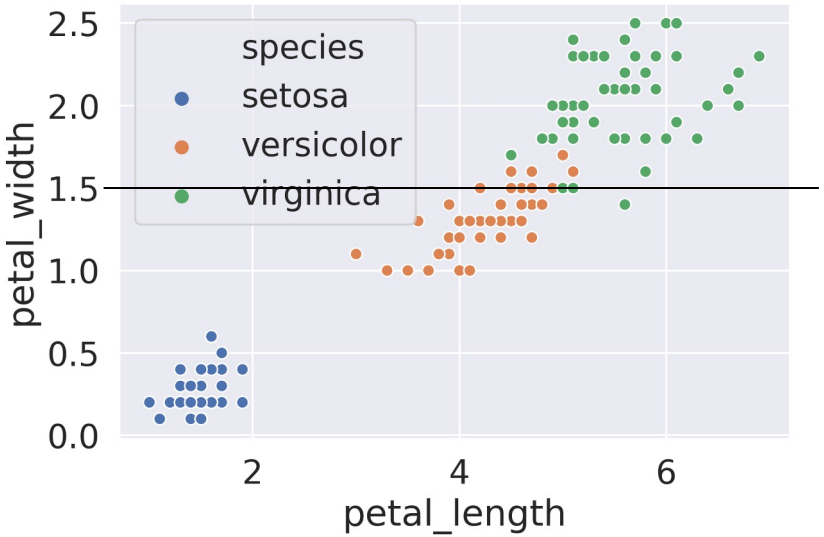


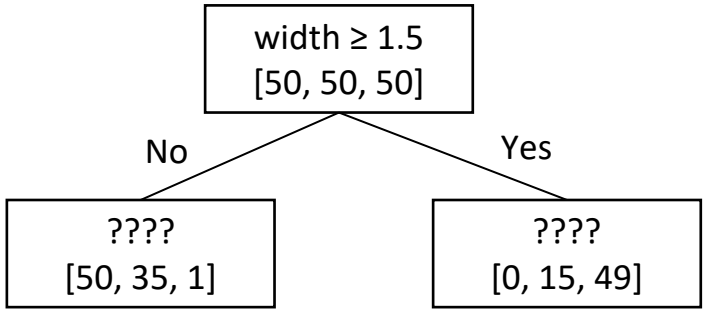


Defining a Best Feature

Question: Which feature and split value is best?

- Equivalently: Which horizontal or vertical line do we want to draw?



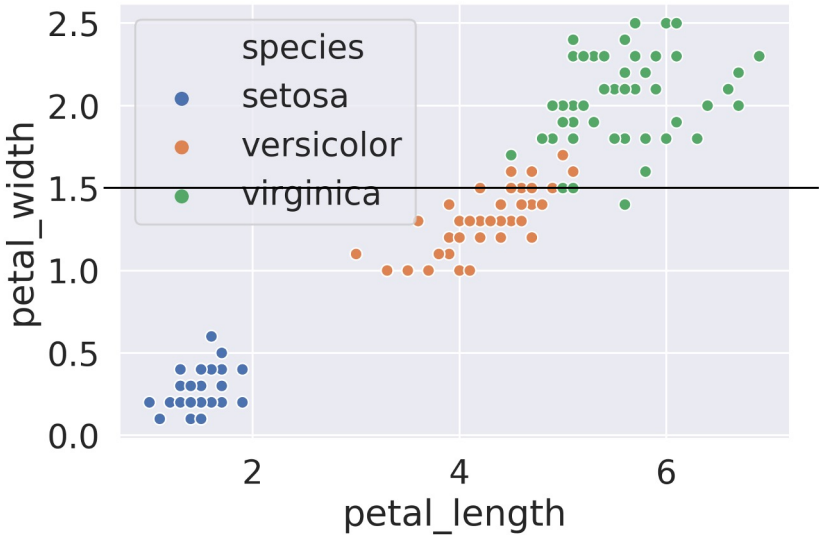


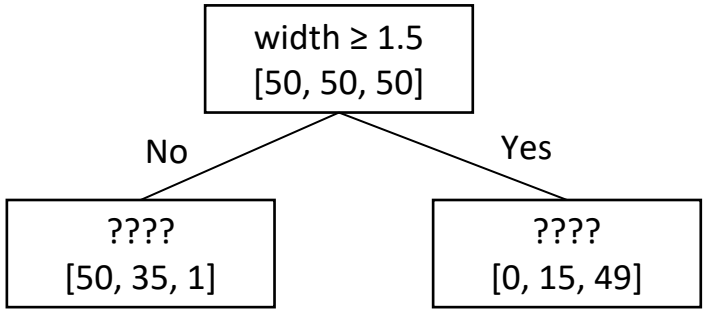
Is this good?

Defining a Best Feature

Question: Which feature and split value is best?

- Equivalently: Which horizontal or vertical line do we want to draw?

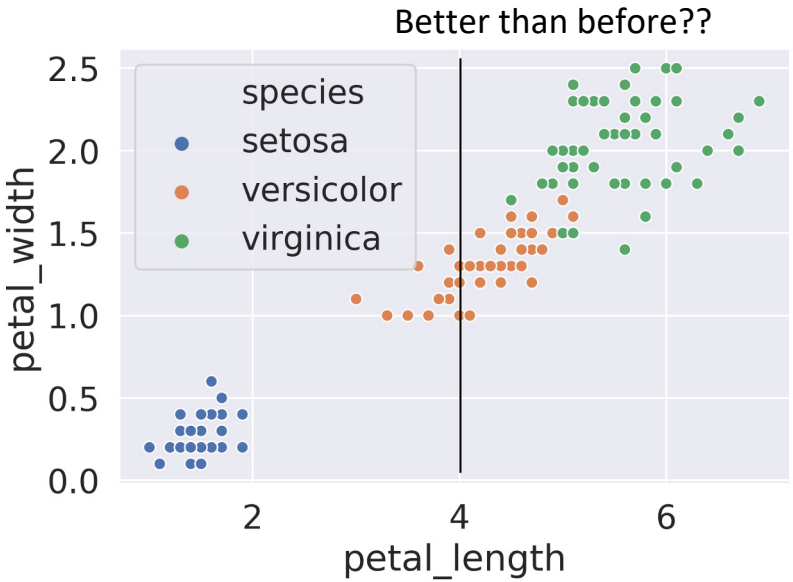


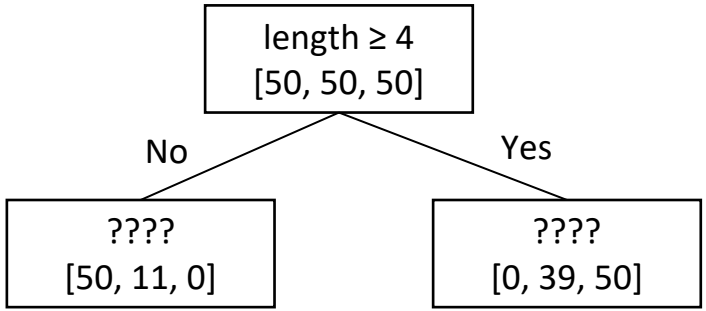


Is this good? It does help, but we could do better!

Question: Which feature and split value is best?

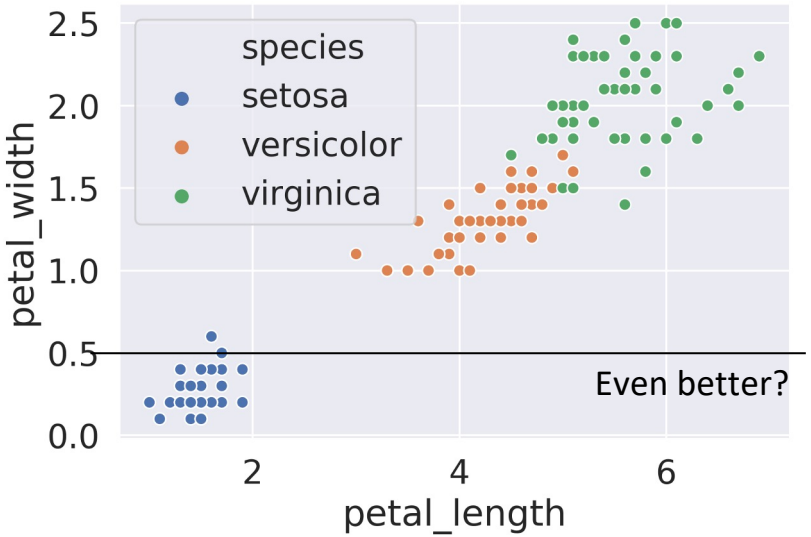
- Equivalently: Which horizontal or vertical line do we want to draw?

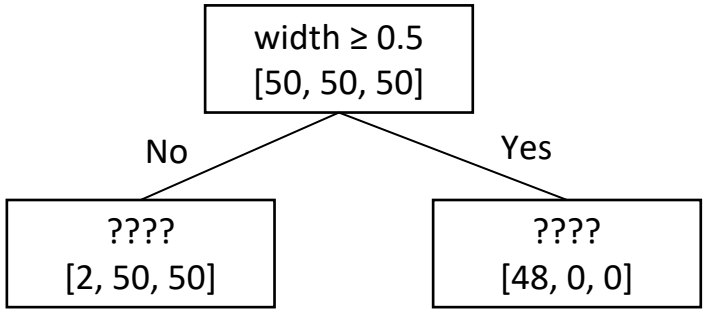




Question: Which feature and split value is best?

- Equivalently: Which horizontal or vertical line do we want to draw?



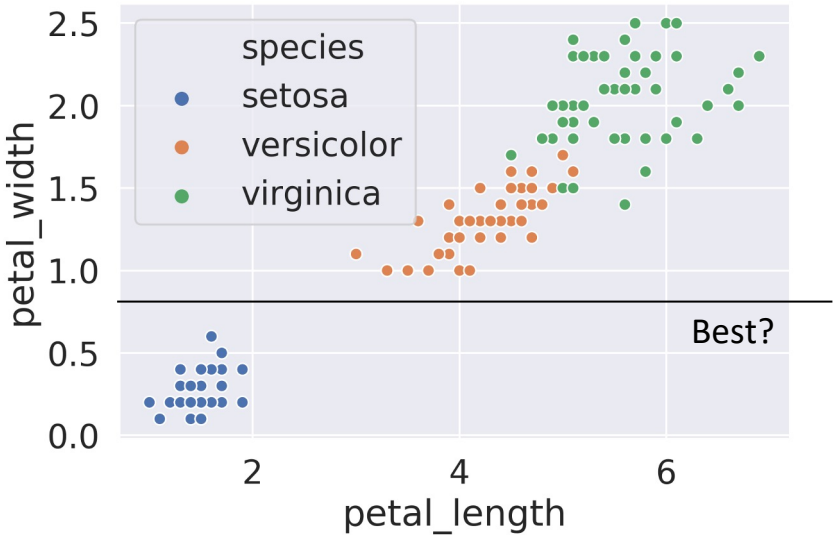


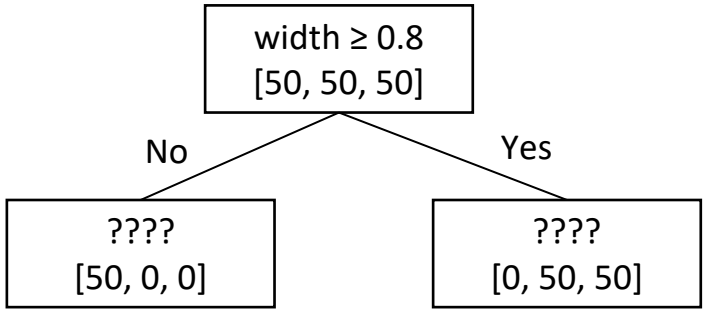
Defining a Best Feature

Question: Which feature and split value is best?

- Equivalently: Which horizontal or vertical line do we want to draw?

We need some sort of rigorous definition for a good split.





Attribute selection

At a given branch in the tree, the set of samples S to be classified has P positive and N negative samples

The amount of entropy in the set S is

$$H(P,\ N) = -\frac{P}{P+N}\log_2\frac{P}{P+N} - \frac{N}{P+N}\log_2\frac{N}{P+N}$$

Note that:

- If P=0 (or N=0), H(P, N) = 0 → no uncertainty
- If P=N, H(P, N) = 1 → max uncertainty

Attribute Selection: Information Gain

Data Mining Book slide

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

$\frac{5}{14}I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes's and 3 no's. Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

Attribute Selection by Information Gain Computation

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"
- I(p, n) = I(9, 5) =0.940
- Compute the entropy for

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

$$E(age) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.694$$

Hence

$$Gain(age) = I(p,n) - E(age)$$

Gain(age)=0.246

Similarly

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit\_rating) = 0.048$$

The attribute "age" becomes the root.

Attribute selection

Attribute A partitions S into S_1, S_2, ... S_v

Entropy of attribute A is

$$H(A) = \sum_{i=1}^{v} \frac{P_i + N_i}{P + N} H(P_i, N_i)$$

The information gain obtained by splitting S using A is

$$Gain(A) = H(P, N) - H(A)$$

Gain(Age) = 0.94 – 0.69 = 0.25 ← split on age

Gain(Income) = 0.94 – 0.91 = 0.03

Gain(Student) = 0.94 – 0.78 = 0.16

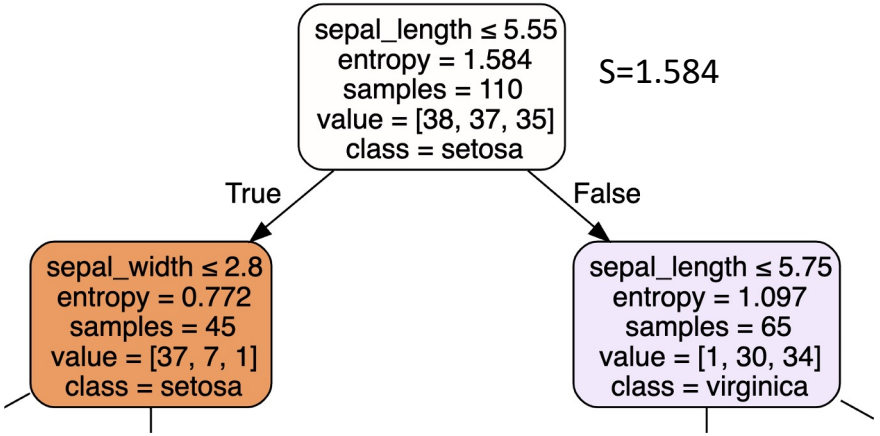
Gain(Rating) = 0.94 – 0.89 = 0.05

Node Entropy

Let p_c be the proportion of data points in a node with label C.

For example, for the node at the top of the decision tree,

- p_0 = 38/110 = 0.35,
- p_1 = 37/110 = 0.34,
- p_2 = 35/110 = 0.32.



Define the **entropy** S of a node (in bits) as:

$$S = -\sum_c p_c \log_2 (p_c)$$

For example, S for the top node is:

$-0.35 \log_2 0.35 - 0.34 \log_2 0.34 - 0.32 \log_2 0.32 = 0.52 + 0.53 + 0.53 = 1.584$ bits

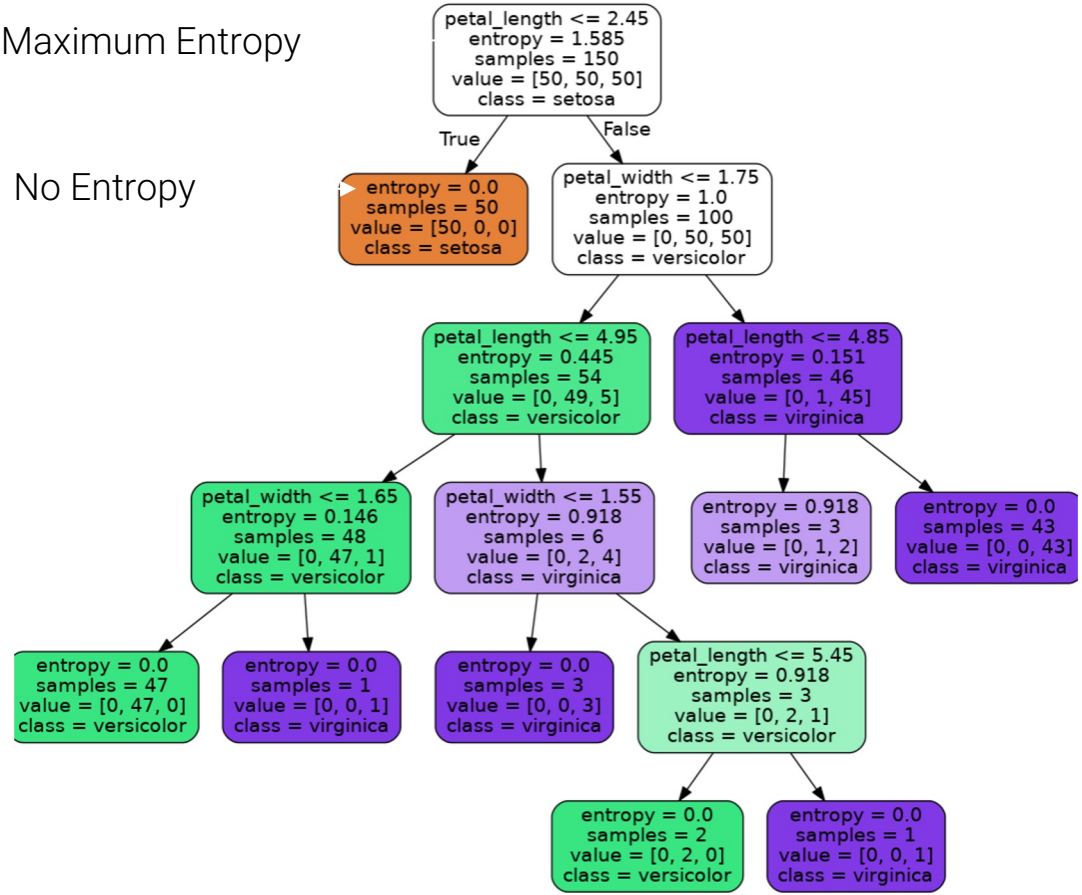
What is the entropy of this node?

Can think of entropy as how unpredictable a node is. Low entropy means more predictable. High entropy means less predictable.

Entropy of Nodes in Our First Decision Tree

Maximum Entropy

No Entropy



Recall: Entropy means uncertainty/surprise!

Exploring Entropy

Observations about entropy:

- A node where all data are part of the same class has zero entropy.

  $-1 \log_2 \quad 1 = 0$ bits.

- A node where data are evenly split between two classes has entropy 1.

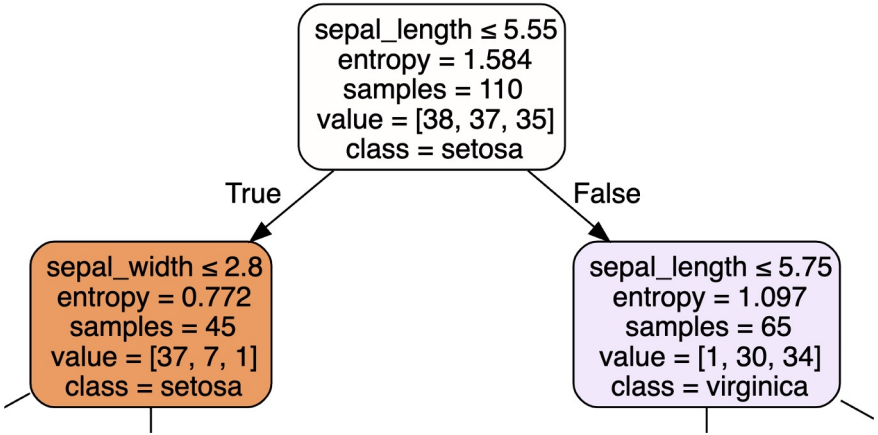
  $-0.5 \log_2 \quad 0.5 - 0.5 \log_2 \quad 0.5 = 1$ bit.

- A node where data are evenly split between 3 classes has entropy 1.58.

  $3 \times (-0.33 \log_2 \quad 0.33) = 1.58$ bits.

- A node where data are evenly split into C classes has entropy $\log_2 C$.

  $C \times (-1/C \log_2 \quad 1/C) = -\log_2 \quad 1/C = \log_2 \quad C$



For more on entropy, see wiki or this youtube video.

Weighted Entropy

We can use **Weighted Entropy** as a loss function in helping us decide which split to take

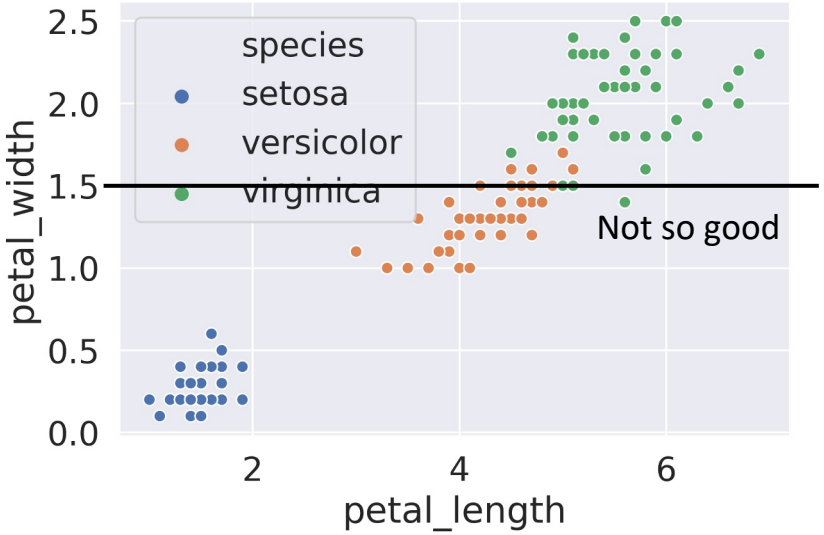
Suppose a given split results in two nodes X and Y with N_1 and N_2 total samples each. The loss of that split is given by:

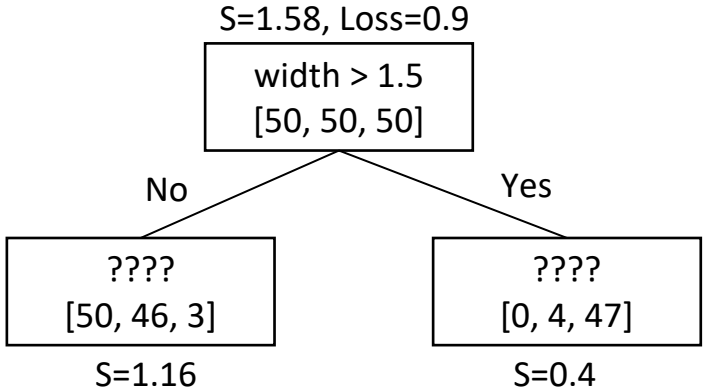
$$L = \frac{N_1 S(X) + N_2 S(Y)}{N_1 + N_2}$$

Defining a Best Feature

Split choice #1: width > 1.5. Compute entropy of child nodes:

- entropy([50, 46, 3]) = 1.16
- entropy([4, 47]) = 0.4
- Weighted average: 99/150 × 1.16 + 51/150 × 0.4 = **0.9**

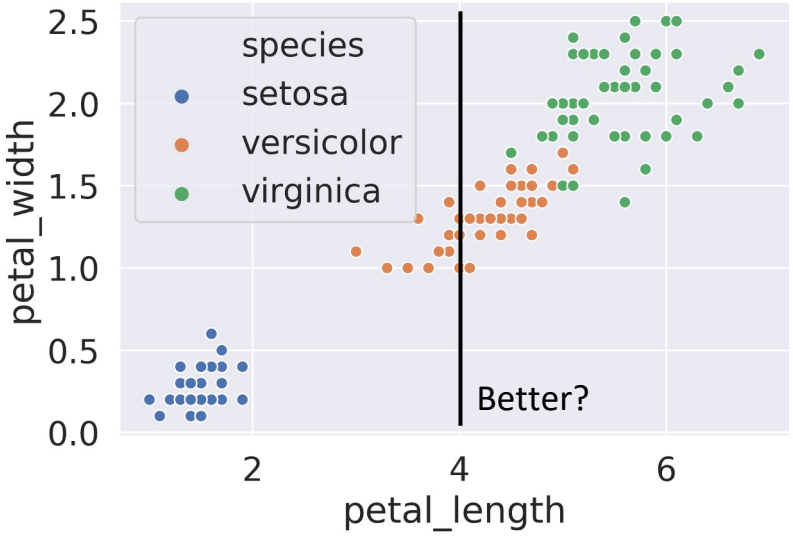


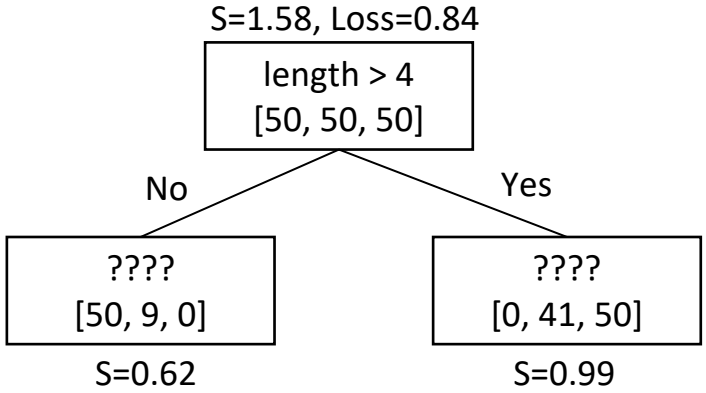


Defining a Best Feature

Split choice #2: length > 4. Compute entropy of child nodes:

- entropy([50, 9]) = 0.62
- entropy([41, 50]) = 0.99
- Weighted Average: **0.84**: Better than split choice #1!

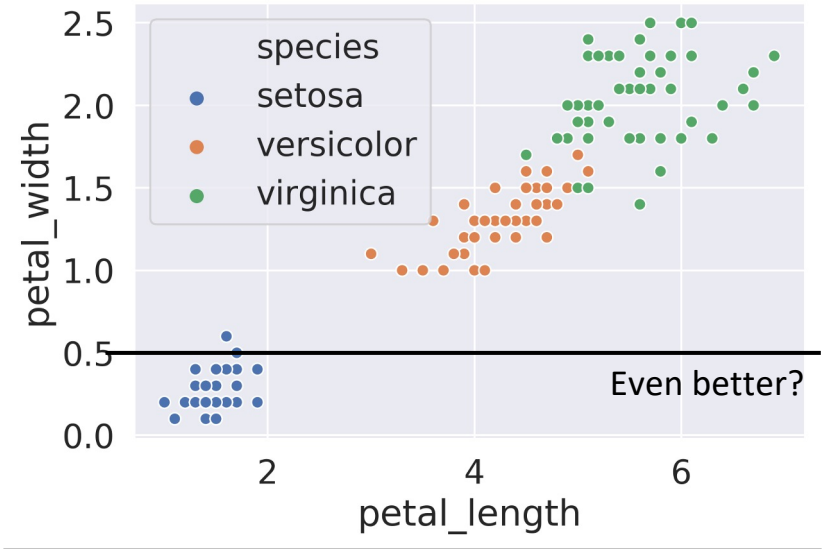


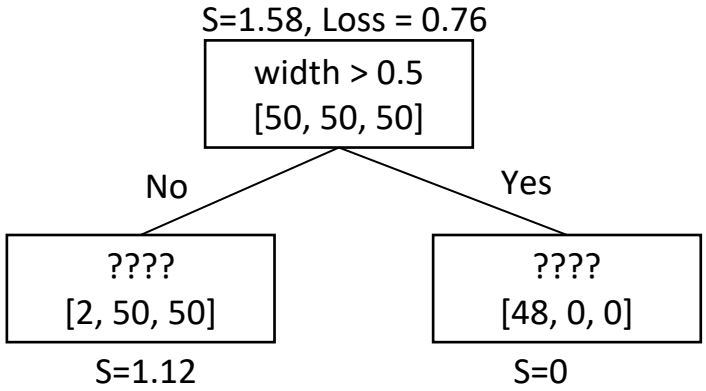


Defining a Best Feature

Split choice #3: width > 0.5. Compute entropy of child nodes:

- entropy([2, 50, 50]) = 1.12
- entropy([48]) = 0
- Weighted average: **0.76**: Lower than split choice #2!

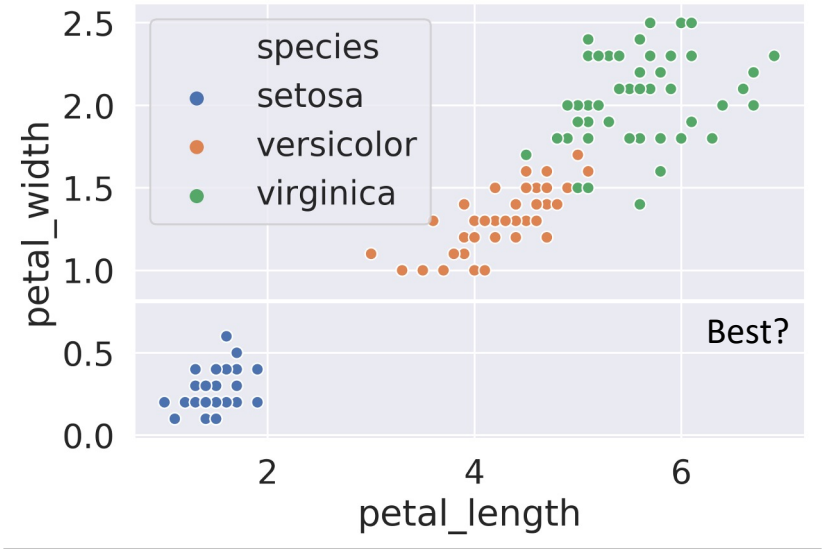


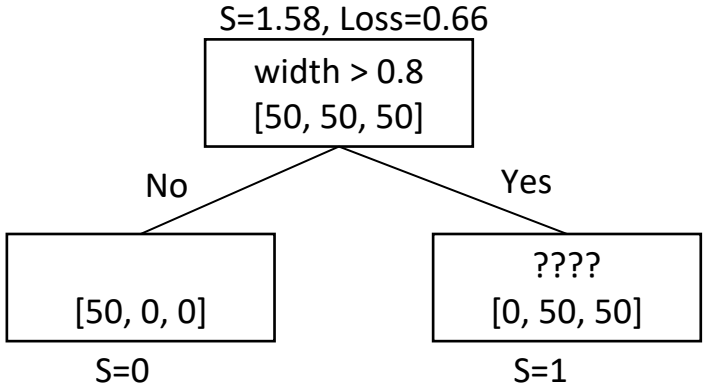


Defining a Best Feature

Split choice #4: width > 0.9. Compute entropy of child nodes:

- entropy([50, 50]) = 1
- entropy([50]) = 0
- Weighted average: **0.66**: Lower than split choice #3!





Pruning

The construction phase does not filter out noise → **overfitting**

Many possible pruning strategies

- Stop partitioning a node when the corresponding number of samples assigned to a leaf goes below a threshold
- Bottom-up cross validation: Build the full tree and replace nodes with leaves labeled with the majority class if classification accuracy on **validation set (!)** does not get worse this way

Comments

Decision trees are just an example of classification algorithm

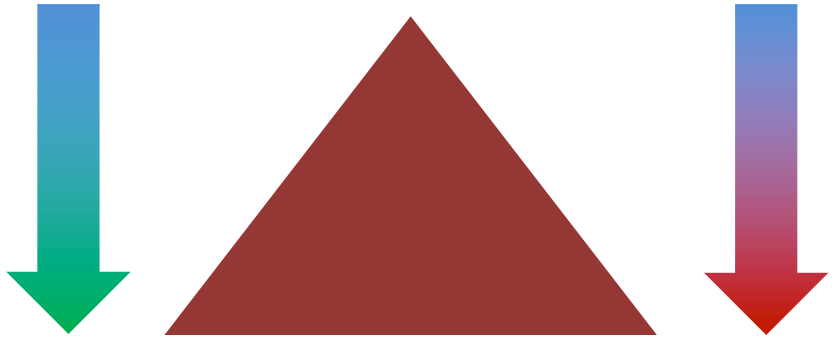
- Many other out there (k-NN, naive Bayes, SVM, neural networks, logistic regression, random forest ...)

Maybe not the best one ...

- Sensitive to small perturbation in the data (high variance)
- Tend to overfit
- Non-incremental: Need to be re-trained from scratch if new training data becomes available

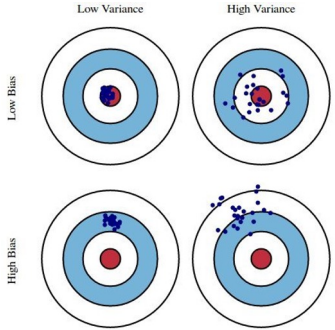
Decision tree models

- As tree depth increases, bias decreases and variance generally increases. Why? (Hint: think about k-NN)



Bias decreases with tree depth

Variance increases with tree depth



Ensemble methods

Are, metaphorically, like **crowdsourced machine learning algorithms**:

- Take a collection of simple or *weak* learners
- Combine their results to make a single, better learner

Types:

- **Bagging:** train learners in parallel on different samples of the data, then combine by voting (discrete output) or by averaging (continuous output).
- **Stacking:** combine outputs from various models using a second-stage learner like linear regression.
- **Boosting:** train learner again, but after filtering/weighting samples based on output of previous train/test runs.

Random forests

Grow K trees on datasets sampled from the original dataset (size N) with replacement (bootstrap samples), p = number of features.

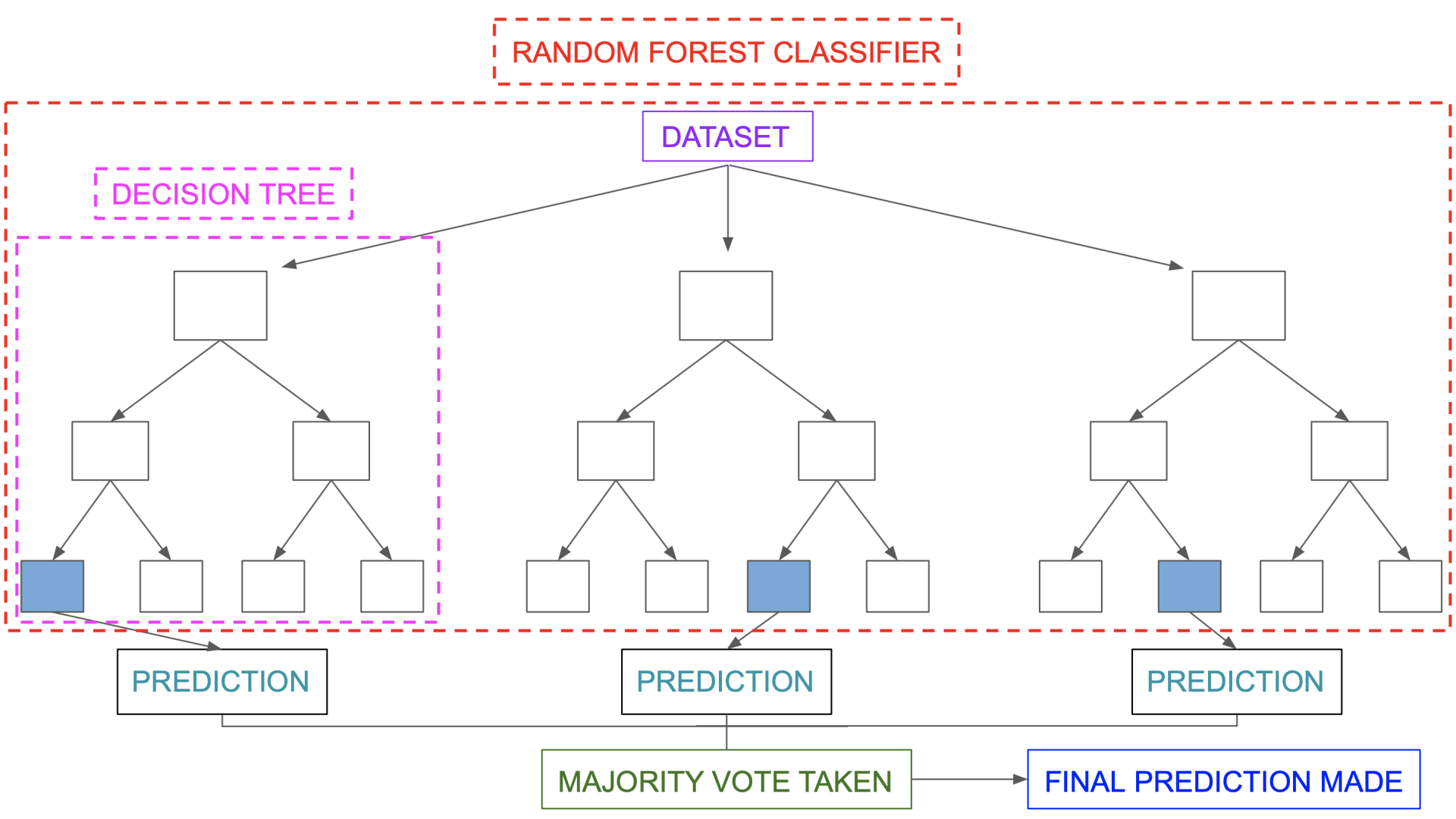
- Draw K bootstrap samples of size N
- Grow each decision tree by selecting a random set of m out of p features at each node, and choosing the best feature to split on.
- Aggregate the predictions of the trees (most popular vote, or average) to produce the final class (example of bagging).

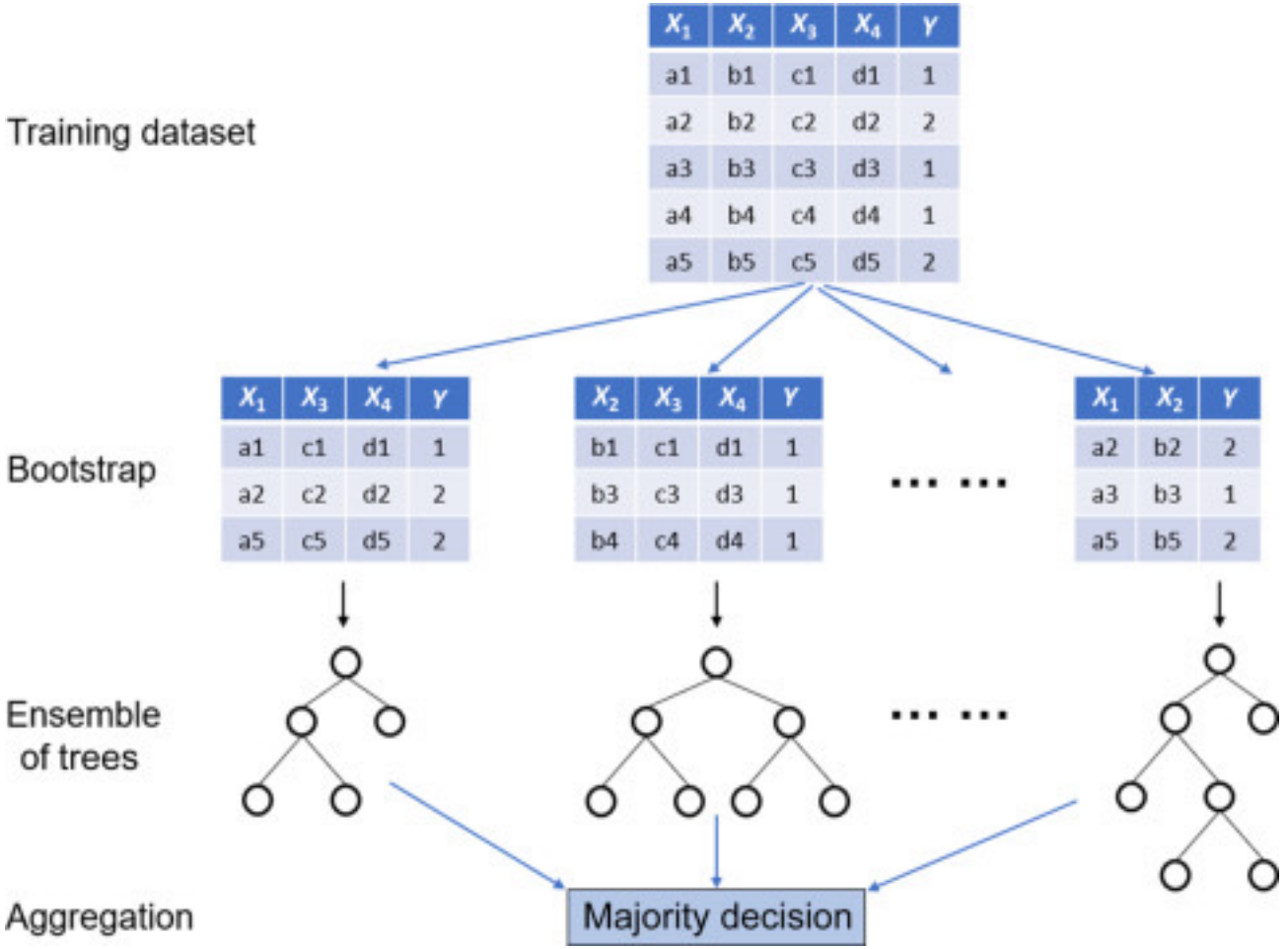
Typically m might be e.g. sqrt(p), but can be smaller.

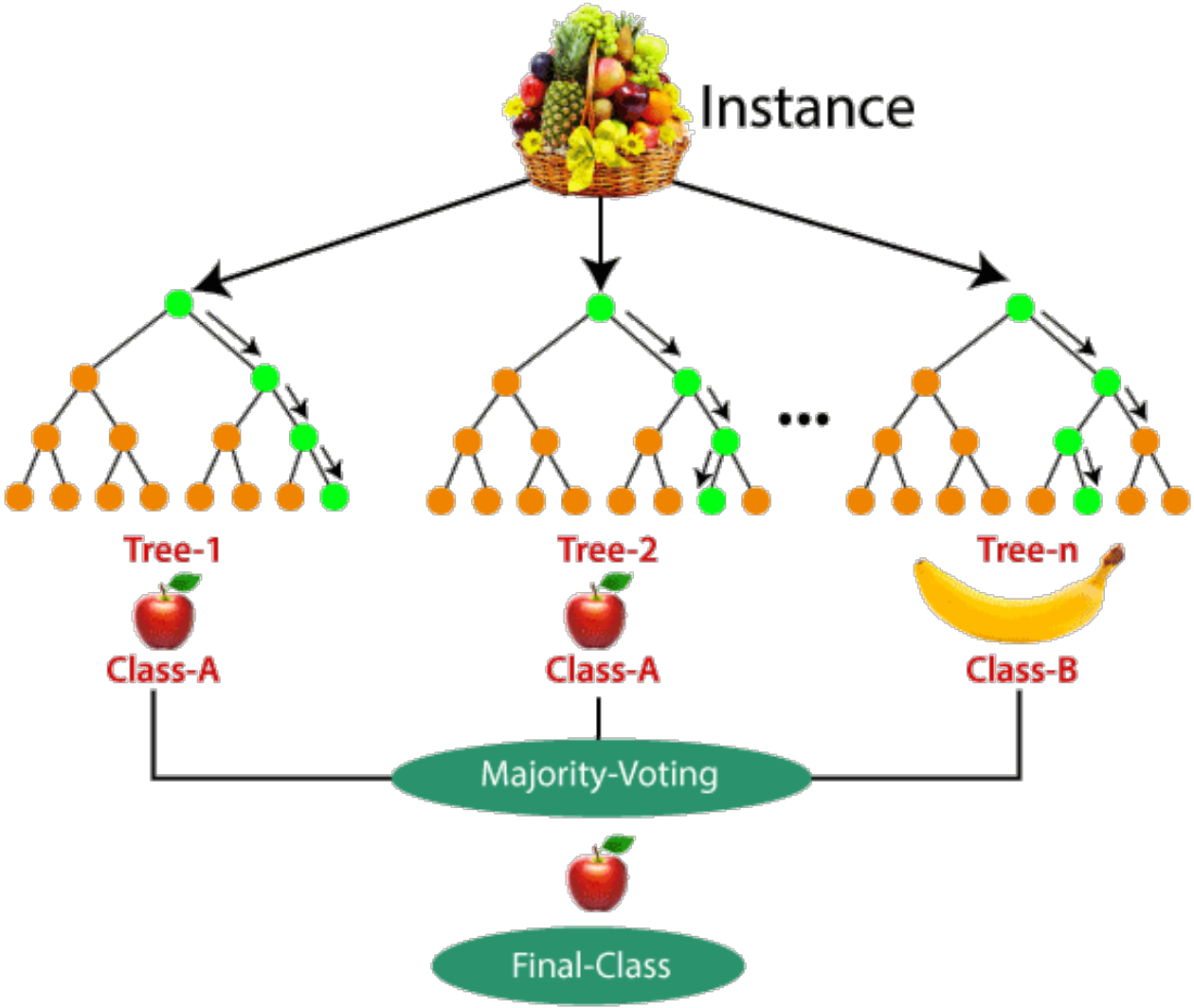
Random forests

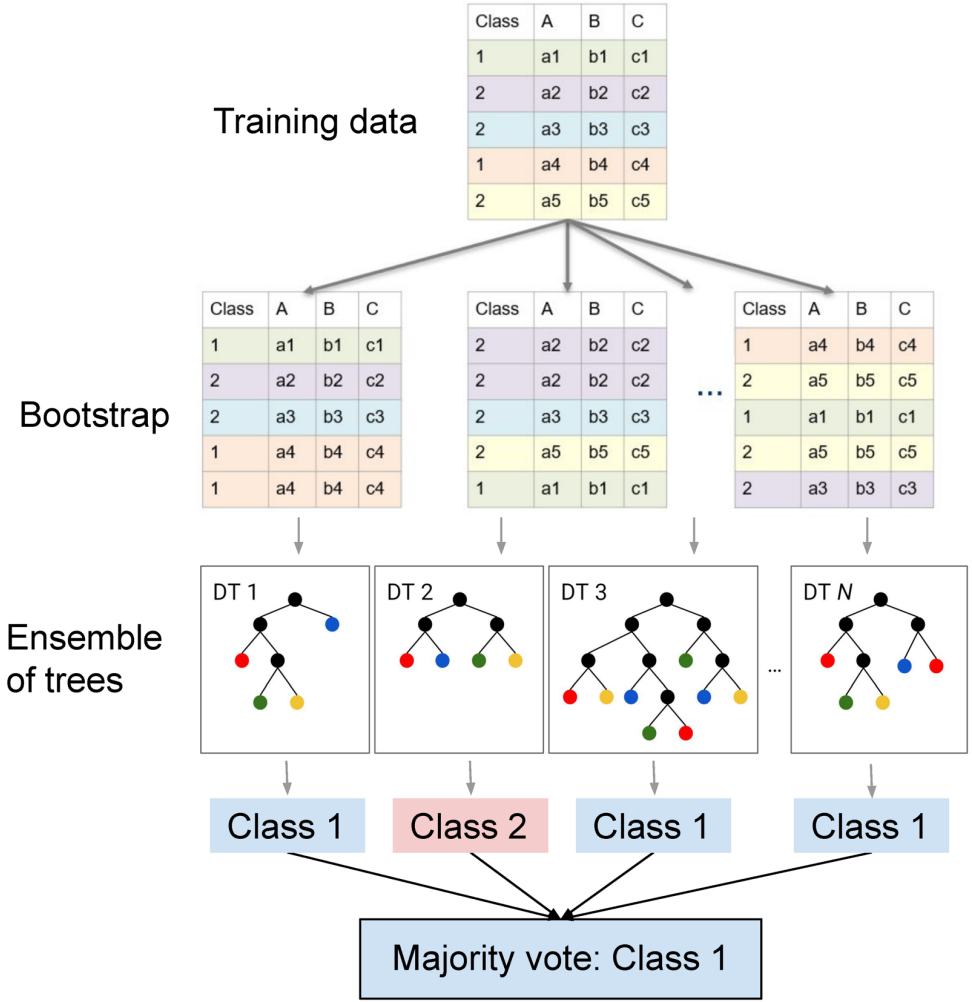
Principles: we want to take a vote between different learners so we don't want the models to be too similar. The following two criteria ensure diversity in the individual trees:

- Draw K bootstrap samples of size N:
  - Each tree is trained on different data.
- Grow a decision tree by selecting a random set of m out of p features at each node, and choosing the best feature to split on.
  - Corresponding nodes in different trees (usually) can't use the same feature to split.


RANDOM FOREST CLASSIFIER
DATASET
DECISION TREE
PREDICTION
PREDICTION
PREDICTION
MAJORITY VOTE TAKEN
FINAL PREDICTION MADE


Training dataset
X1 X2 X3 X4 Y
a1 b1 c1 d1 1
a2 b2 c2 d2 2
a3 b3 c3 d3 1
a4 b4 c4 d4 1
a5 b5 c5 d5 2
Bootstrap
X1 X3 X4 Y
a1 c1 d1 1
a2 c2 d2 2
a5 c5 d5 2
X2 X3 X4 Y
b1 c1 d1 1
b3 c3 d3 1
b4 c4 d4 1
X1 X2 Y
a2 b2 2
a3 b3 1
a5 b5 2
Ensemble of trees
Aggregation
Majority decision


Instance
Tree-1
Tree-2
Tree-n
Class-A
Class-A
Class-B
Majority-Voting
Final-Class


Training data
Class A B C
1 a1 b1 c1
2 a2 b2 c2
2 a3 b3 c3
1 a4 b4 c4
2 a5 b5 c5
Bootstrap
Class A B C
1 a1 b1 c1
2 a2 b2 c2
2 a3 b3 c3
1 a4 b4 c4
1 a4 b4 c4
Class A B C
2 a2 b2 c2
2 a2 b2 c2
2 a3 b3 c3
2 a5 b5 c5
1 a1 b1 c1
...
Class A B C
1 a4 b4 c4
2 a5 b5 c5
1 a1 b1 c1
2 a5 b5 c5
2 a3 b3 c3
Ensemble of trees
DT 1
DT 2
DT 3
...
DT N
Class 1
Class 2
Class 1
Class 1
Majority vote: Class 1

Random forests

- **Very popular in practice,** probably the most popular classifier for dense data (up to a few thousand features)
- **Easy to implement** (simply train many normal decision trees)
- **Parallelizes easily**

- **Needs many passes over the data** – at least the max depth of the trees (<< boosted trees though, cf. next slide)

Application of Random forests

Banking

Random forest is used in banking to predict the creditworthiness of a loan applicant. This helps the lending institution make a good decision on whether to give the customer the loan or not. Banks also use the random forest algorithm to detect fraudsters.

Health care

Health professionals use random forest systems to diagnose patients. Patients are diagnosed by assessing their previous medical history. Past medical records are reviewed to establish the right dosage for the patients.

Stock market

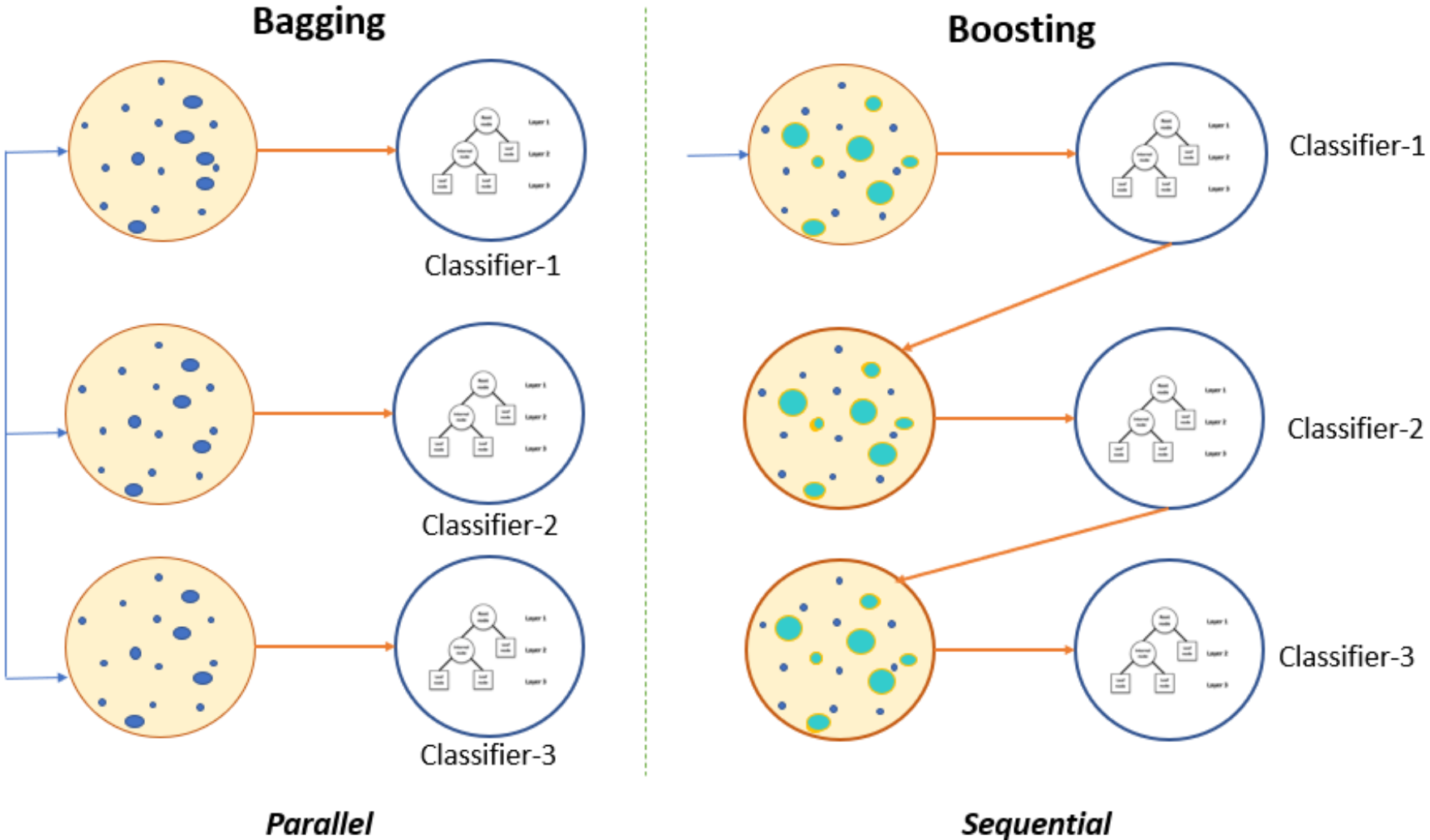
Financial analysts use it to identify potential markets for stocks. It also enables them to identify the behaviour of stocks.

E-commerce

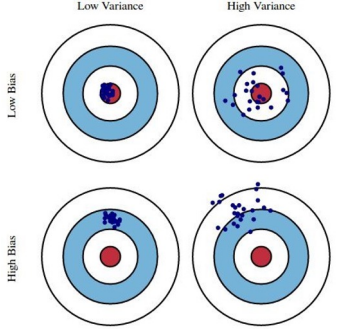
Through rain forest algorithms, e-commerce vendors can predict the preference of customers based on past consumption behaviour.

Boosted decision trees

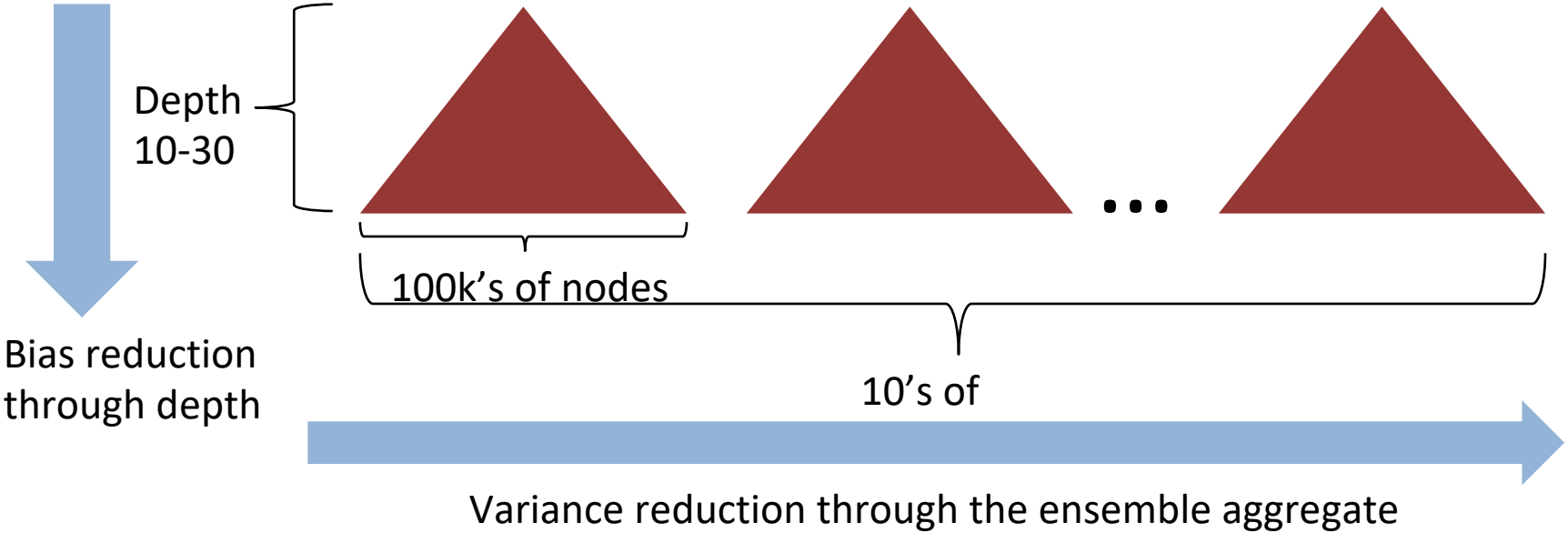
- A more recent alternative to random forests (RF) [good intro here and here]
- In contrast to RFs, whose trees are trained **independently**, BDT trees are trained **sequentially** by **boosting**: Each tree is trained to predict ("correct") error residuals of previous trees (--> bias reduction).
- Both RF and boosted trees can produce very high-quality models. Superiority of one method or the other is very dataset-dependent.
- Resource requirements are very different as well, so it's actually non-trivial to compare the methods.


Bagging
Classifier-1
Classifier-2
Classifier-3
Parallel
Boosting
Classifier-1
Classifier-2
Classifier-3
Sequential

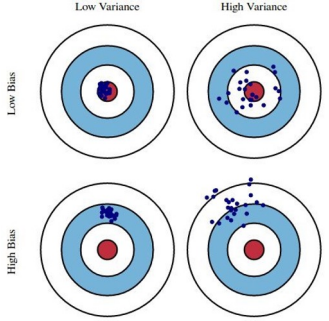
Random forests vs. boosted trees



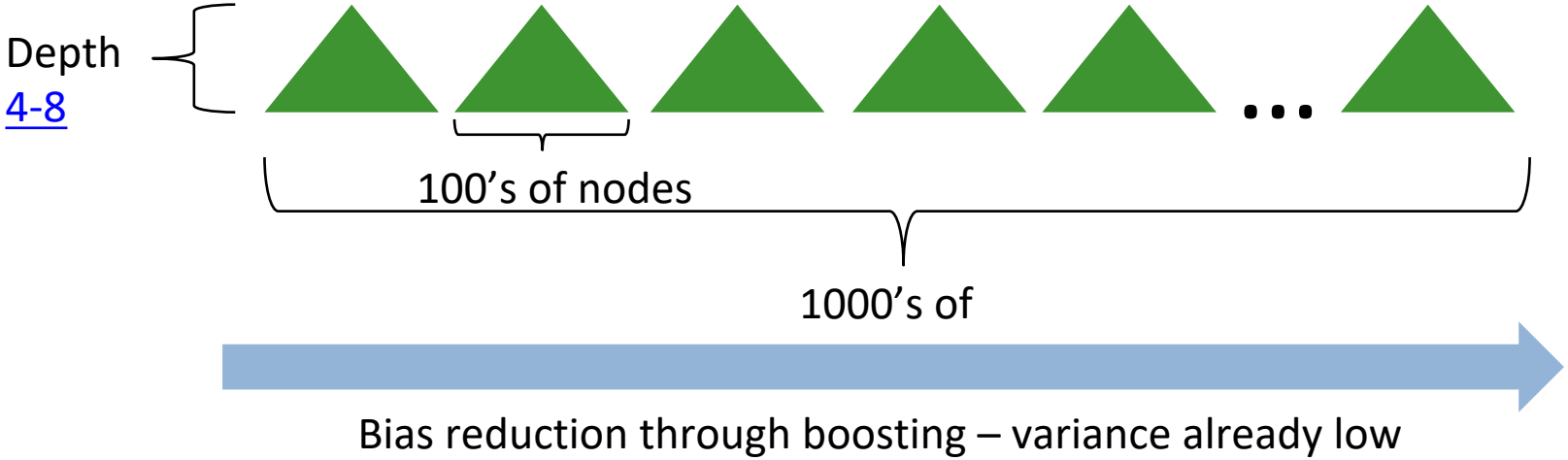
- The “geometry” of the methods is very different:
- Random forests use 10’s of deep, large trees:



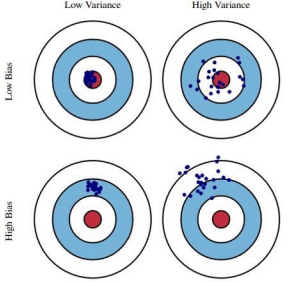
Random forests vs. boosted trees



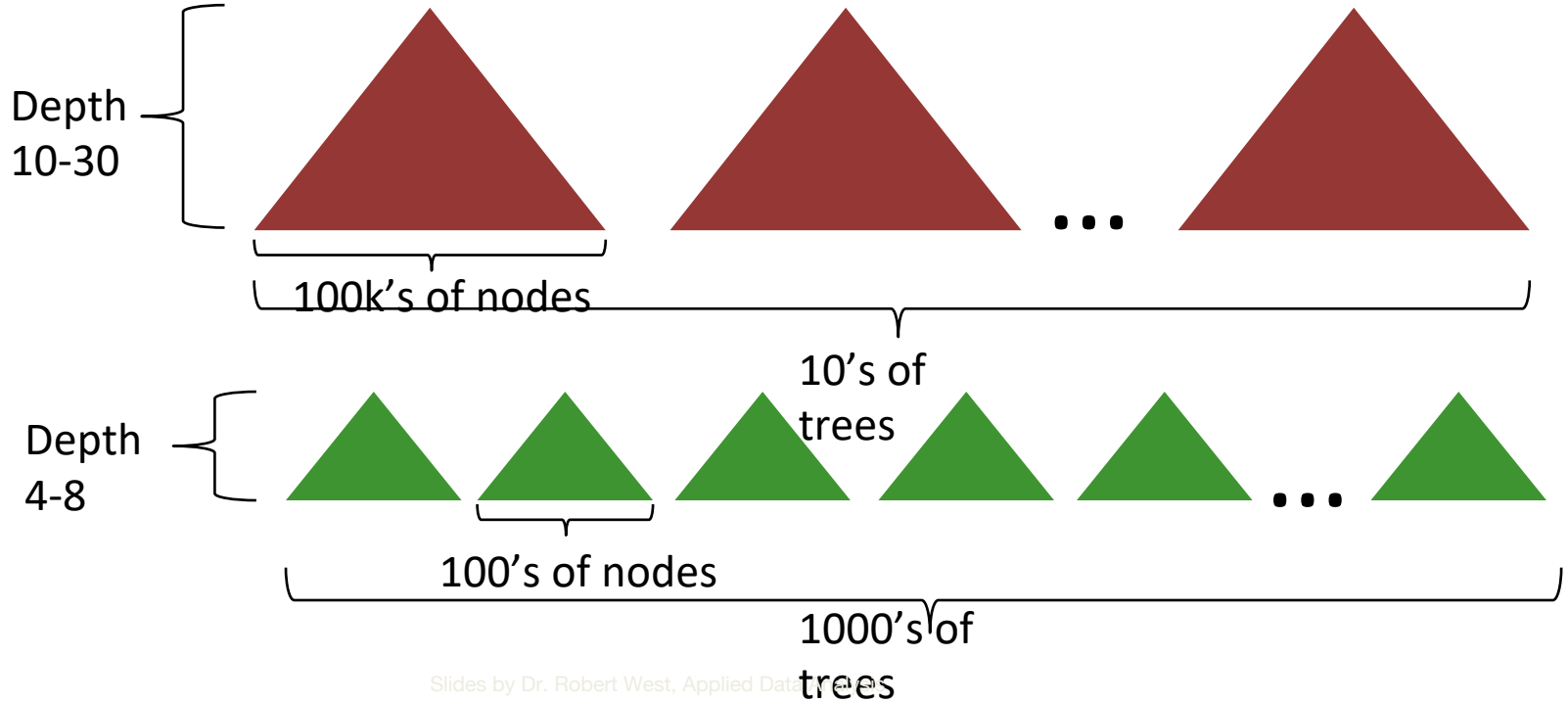
- The “geometry” of the methods is very different:
- Boosted decision trees use 1000’s of shallow, small trees:



Random forests vs. boosted trees



- RF training embarrassingly parallel, can be very fast
- Evaluation of trees (runtime) also much faster for RFs



Standing queries

- The path from IR to text classification:
  - You have an information need to monitor, say:
    - Unrest in the Niger delta region
  - You want to rerun an appropriate query periodically to find new news items on this topic
  - You will be sent new documents that are found
    - I.e., it's not ranking but classification (relevant vs. not relevant)
- Such queries are called **standing queries**
  - Long used by "information professionals"
  - A modern mass instantiation is **Google Alerts**
- Standing queries are (hand-written) text classifiers

From: Google Alerts
Subject: **Google Alert - stanford -neuro-linguistic nlp OR "Natural Language Processing" OR parser OR tagger OR ner OR "named entity" OR segmenter OR classifier OR dependencies OR "core nlp" OR corenlp OR phrasal**
Date: May 7, 2012 8:54:53 PM PDT
To: Christopher Manning

Web

3 new results for **stanford -neuro-linguistic nlp OR "Natural Language Processing" OR parser OR tagger OR ner OR "named entity" OR segmenter OR classifier OR dependencies OR "core nlp" OR corenlp OR phrasal**

Twitter / **Stanford NLP** Group: @Robertoross If you only n ...
@Robertoross If you only need tokenization, java -mx2m edu.**stanford.nlp**. process.PTBTokenizer file.txt runs in 2MB on a whole file for me.... 9:41 PM Apr 28th ...
twitter.com/stanfordnlp/status/196459102770171905

[Java] LexicalizedParser lp = LexicalizedParser.loadModel("edu ...
loadModel("edu/**stanford**/**nlp**/models/lexparser/englishPCFG.ser.gz");. String[] sent = { "This", "is", "an", "easy", "sentence", "." };. Tree parse = lp.apply(Arrays.
pastebin.com/az14R9nd

More Problems with Statistical **NLP** || kuro5hin.org
Tags: nlp, ai, coursera, **stanford**, **nlp**-class, cky, nltk, reinventing the wheel, ... Programming Assignment 6 for **Stanford's nlp**-class is to implement a CKY parser .
www.kuro5hin.org/story/2012/5/5/11011/68221

Tip: Use quotes ("like this") around a set of words in your query to match them exactly. Learn more.

Delete this alert.
Create another alert.
Manage your alerts.

Spam filtering
Another text classification task

From: "" <takworlld@hotmail.com>

Subject: real estate is the only way... gem oalvgkay

Anyone can buy real estate with no money down

Stop paying rent TODAY !

There is no need to spend hundreds or even thousands for similar courses

I am 22 years old and I have already purchased 6 properties using the
methods outlined in this truly INCREDIBLE ebook.

Change your life NOW !

==========

Click Below to order:

http://www.wholesaledaily.com/sales/nmd.htm

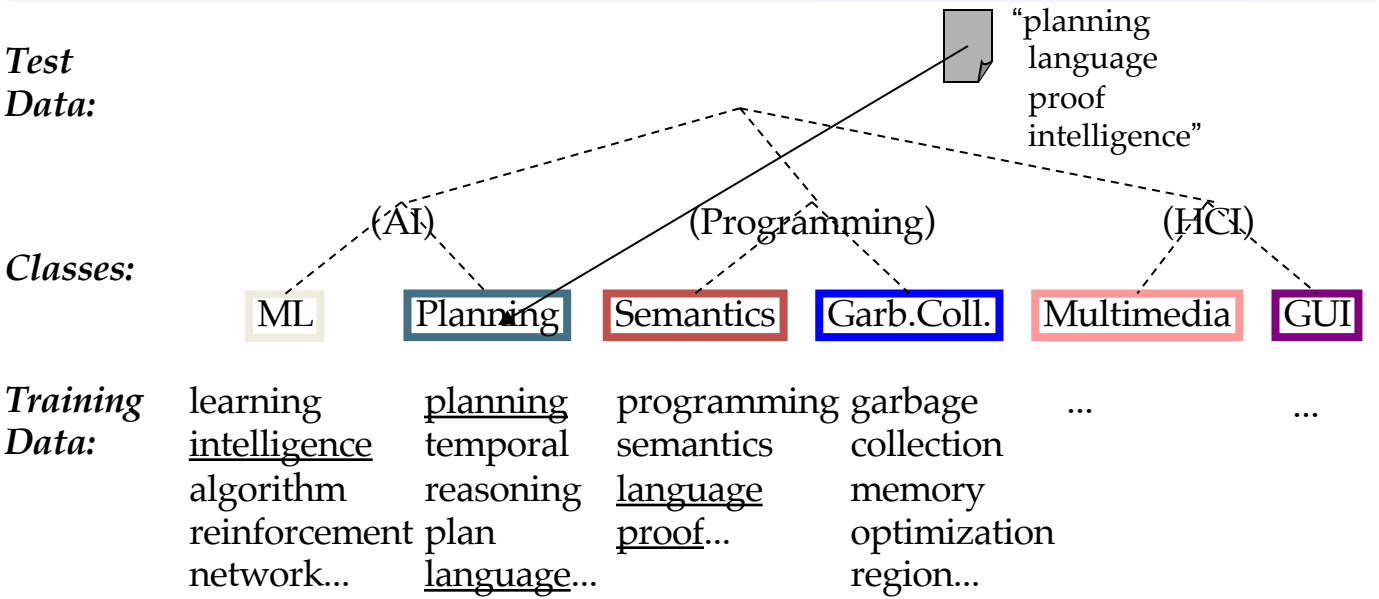
==========

Categorization/Classification

- Given:
  - A representation of a document *d*
    - Issue: how to represent text documents.
    - Usually some type of high-dimensional space – bag of words
  - A fixed set of classes:

    $C = \{c_1, c_2, \ldots, c_J\}$
- Determine:
  - The category of *d:* $\gamma(d) \in C$, where $\gamma(d)$ is a classification function
  - We want to build classification functions ("classifiers").

Document Classification



Classification Methods (2)

- Hand-coded rule-based classifiers
  - Commercial systems have complex query languages
  - Accuracy can be high if a rule has been carefully refined over time by a subject expert
  - Building and maintaining these rules is expensive

Classification Methods (3): Supervised learning

- Given:
  - A document *d*
  - A fixed set of classes:

    $C = \{c_1, c_2, \ldots, c_J\}$
  - A training set *D* of documents each with a label in *C*
- Determine:
  - A learning method or algorithm which will enable us to learn a classifier *γ*
  - For a test document *d*, we assign it the class

$\gamma(d) \in C$

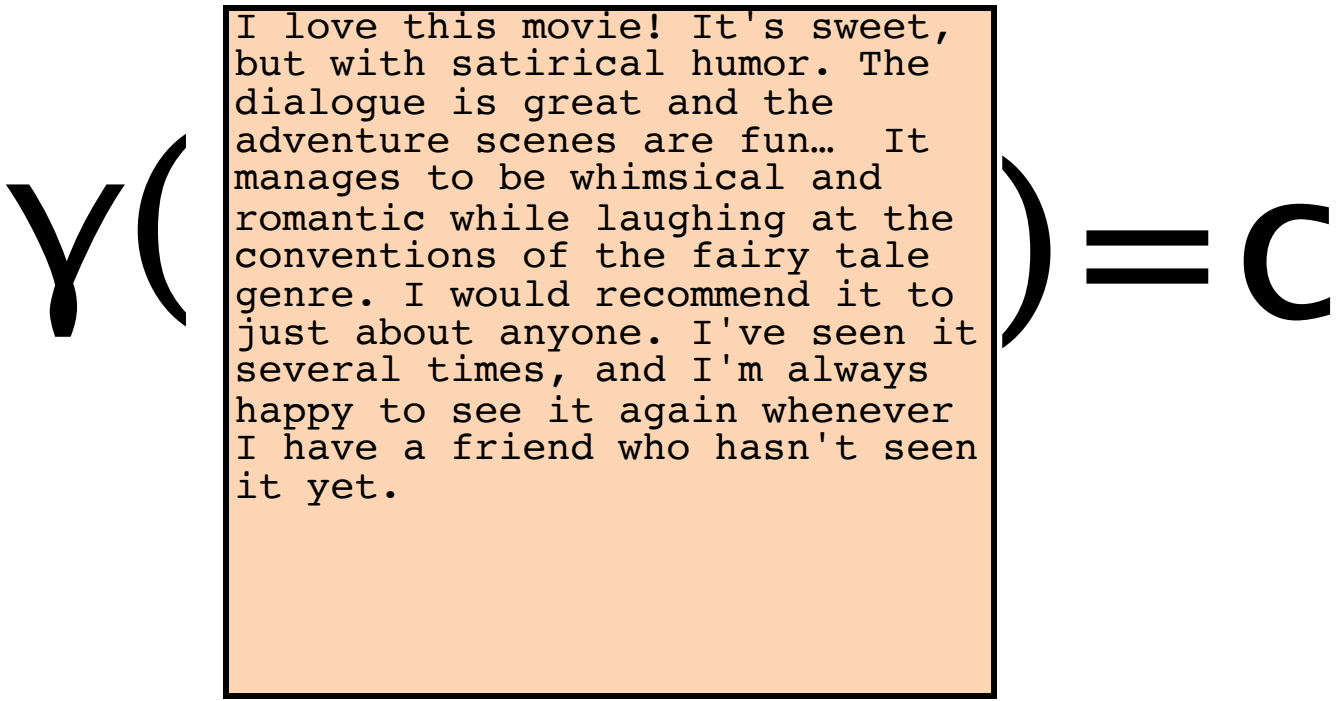
Classification Methods (3)

- Supervised learning
  - Naive Bayes (simple, common)
  - k-Nearest Neighbors (simple, powerful)
  - Decision trees → random forests → gradient-boosted decision trees (e.g., xgboost)
  - … plus many other methods
  - No free lunch: need hand-classified training data
  - But data can be built up by amateurs
- Many commercial systems use a mix of methods

Features

- Supervised learning classifiers can use any sort of feature
  - URL, email address, punctuation, capitalization, dictionaries, network features
- In the simplest bag of words view of documents
  - We use **only** word features
  - we use **all** of the words in the text (not a subset)

The bag of words representation



The bag of words representation

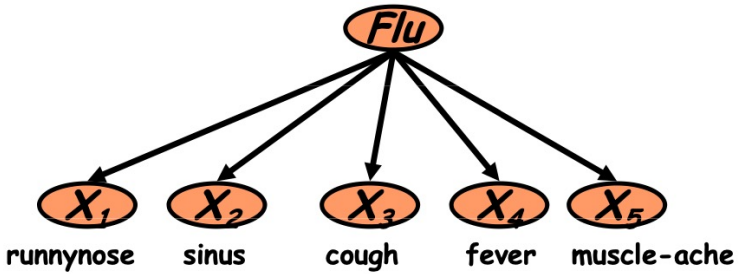
Naïve Bayes Classifier: Naïve Bayes Assumption

- $P(c_j)$
  - Can be estimated from the frequency of classes in the training examples.
- $P(x_1, x_2, \ldots, x_n | c_j)$
  - O($|X|^n \bullet |C|$) parameters
  - Could only be estimated if a very, very large number of training examples was available.

Naïve Bayes Conditional Independence Assumption:

- Assume that the probability of observing the conjunction of attributes is equal to the product of the individual probabilities $P(x_i | c_j)$.

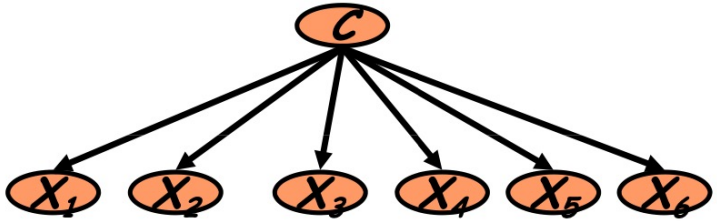
The Naïve Bayes Classifier



- **Conditional Independence Assumption:** features detect term presence and are independent of each other given the class:

$$P(X_1,\ldots,X_5 \mid C) = P(X_1 \mid C) \bullet P(X_2 \mid C) \bullet \cdots \bullet P(X_5 \mid C)$$

Learning the Model

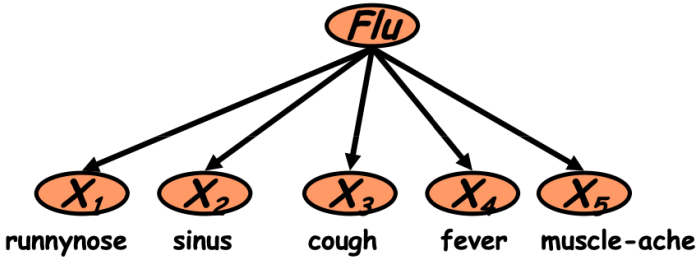


- First attempt: maximum likelihood estimates
  - simply use the frequencies in the data

$$\hat{P}(c_j) = \frac{N(C = c_j)}{N}$$

$$\hat{P}(x_i \mid c_j) = \frac{N(X_i = x_i, C = c_j)}{N(C = c_j)}$$

Problem with Maximum Likelihood



$$P(X_1,\ldots,X_5 \mid C) = P(X_1 \mid C) \bullet P(X_2 \mid C) \bullet \cdots \bullet P(X_5 \mid C)$$

- What if we have seen no training documents with the word ***muscle-ache*** and classified in the topic ***Flu***?

$$\hat{P}(X_5 = t \mid C = nf) = \frac{N(X_5 = t, C = nf)}{N(C = nf)} = 0$$

- Zero probabilities cannot be conditioned away, no matter the other evidence!

$$\ell = \arg\max_c \hat{P}(c) \prod_i \hat{P}(x_i \mid c)$$

Slides by Manning, Raghavan, Schutze

Naïve Bayes

- Classify based on prior weight of class and conditional parameter for what each word says:

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} \left[\log P(c_j) + \sum_{i \in positions} \log P(x_i \mid c_j) \right]$$

- Training is done by counting and dividing:

$$P(c_j) \leftarrow \frac{N_{c_j}}{N} \qquad P(x_k \mid c_j) \leftarrow \frac{T_{c_j x_k} + \alpha}{\sum_{x_i \in V} [T_{c_j x_i} + \alpha]}$$

- Don't forget to smooth

Exercise

	docID	words in document	in c = *China*?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

- Estimate parameters of Naive Bayes classifier
- Classify test document

Example: Parameter estimates

Priors: $\hat{P}(c) = 3/4$ and $\hat{P}(\overline{c}) = 1/4$ Conditional probabilities:

$$\begin{aligned}
\hat{P}(\text{CHINESE}|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\
\hat{P}(\text{TOKYO}|c) = \hat{P}(\text{JAPAN}|c) &= (0+1)/(8+6) = 1/14 \\
\hat{P}(\text{CHINESE}|\overline{c}) &= (1+1)/(3+6) = 2/9 \\
\hat{P}(\text{TOKYO}|\overline{c}) = \hat{P}(\text{JAPAN}|\overline{c}) &= (1+1)/(3+6) = 2/9
\end{aligned}$$

The denominators are (8 + 6) and (3 + 6) because the lengths of $text_c$ and $text_{\overline{c}}$ are 8 and 3, respectively, and because the constant B is 6 as the vocabulary consists of six terms.

Example: Classification

$$\hat{P}(c|d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003$$
$$\hat{P}(\overline{c}|d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001$$

Thus, the classifier assigns the test document to c = *China*. The reason for this classification decision is that the three occurrences of the positive indicator `CHINESE` in d_5 outweigh the occurrences of the two negative indicators `JAPAN` and `TOKYO`.

Naive Bayes is Not So Naive

- Very fast learning and testing (basically just count words)
- Low storage requirements
- Very good in domains with many equally important features
- More robust to irrelevant features than many learning methods

  Irrelevant features cancel out without affecting results

Naive Bayes is Not So Naive

- More robust to concept drift (changing class definition over time)
- Naive Bayes won 1st and 2nd place in KDD-CUP 97 competition out of 16 systems

  Goal: Financial services industry direct mail response prediction: Predict if the recipient of mail will actually respond to the advertisement – 750,000 records.
- A good dependable baseline for text classification (but not the best)!

Evaluating Categorization

- Evaluation must be done on test data that are independent of the training data
  - Sometimes use cross-validation (averaging results over multiple training and test splits of the overall data)
- Easy to get good performance on a test set that was available to the learner during training (e.g., just memorize the test set)

Evaluating Categorization

Key: **tp** = True Positive, **tn** = True Negative, **fp** = False Positive, **fn** = False Negative

Metric Name	Metric Forumla
Accuracy	$\textbf{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$
Precision	$\textbf{Precision} = \frac{tp}{tp + fp}$
Recall	$\textbf{Recall} = \frac{tp}{tp + fn}$
F1-score	$\textbf{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$

- Measures: precision, recall, F1, classification accuracy
- Classification accuracy: *r/n* where n is the total number of test docs and *r* is the number of test docs correctly classified

Vector Space Representation

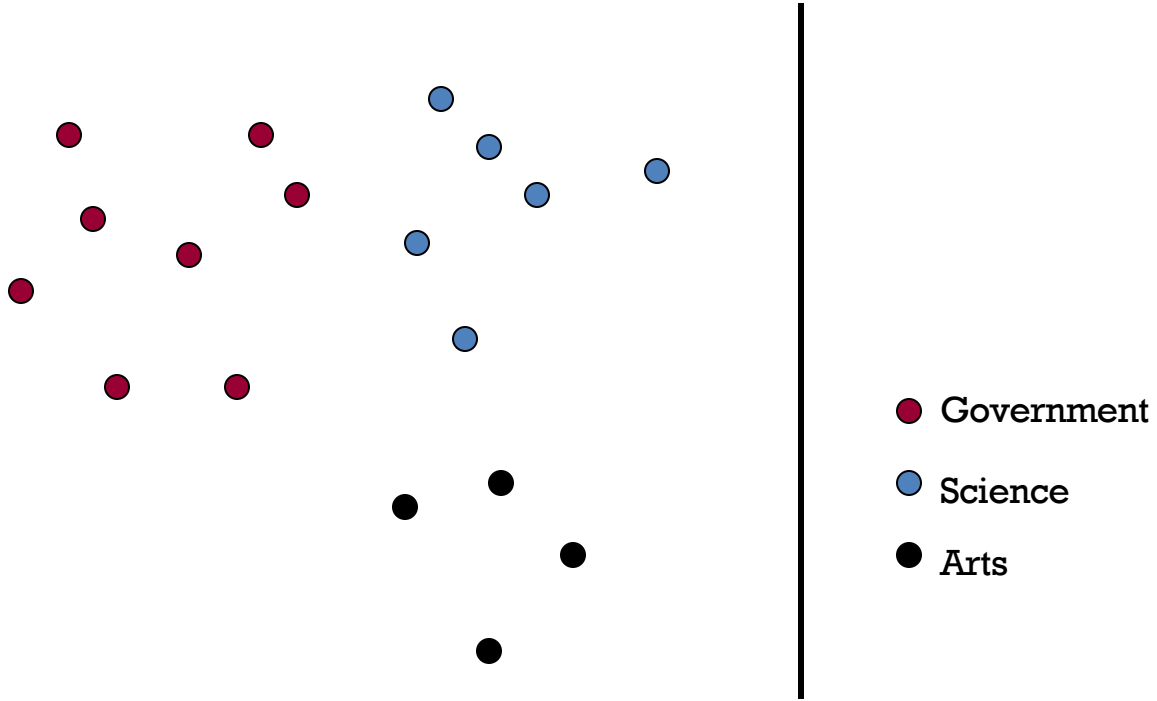
- Each document is a vector, one component for each term (= word).
- Normally normalize vectors to unit length.
- High-dimensional vector space:
  - Terms are axes
  - 10,000+ dimensions, or even 100,000+
  - Docs are vectors in this space

- How can we do classification in this space?

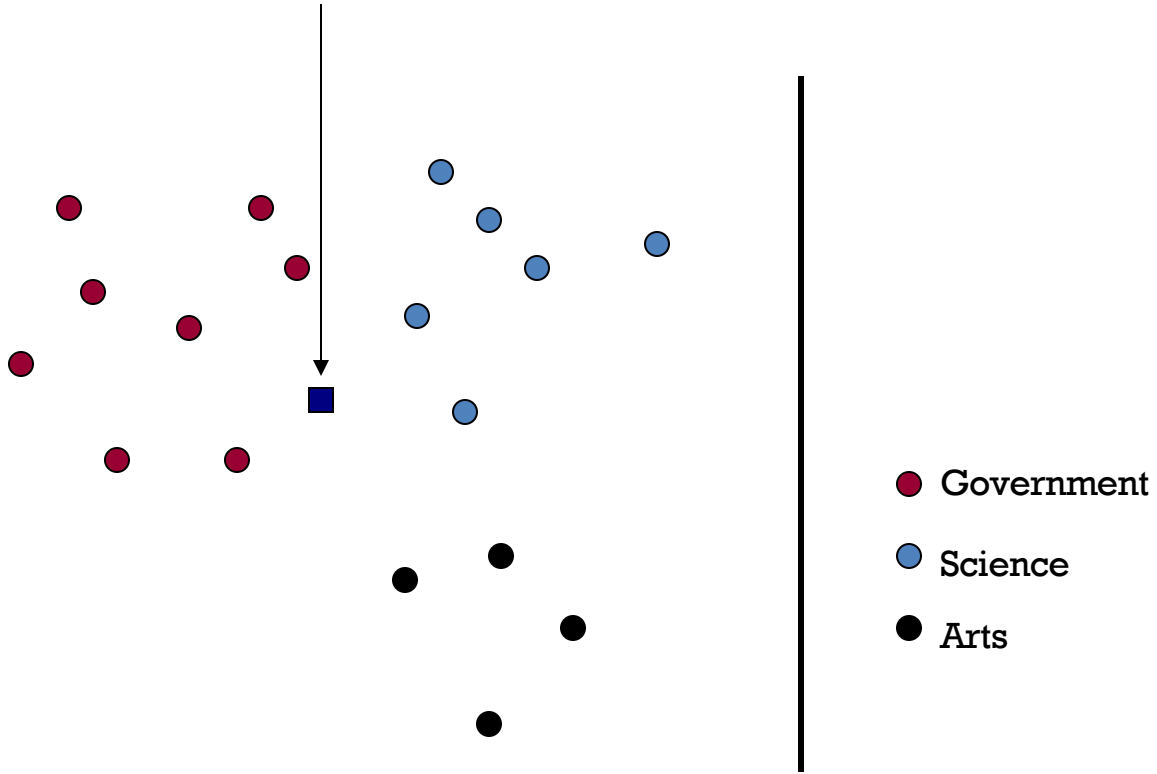
Classification Using Vector Spaces

- In vector space classification, training set corresponds to a labeled set of points (equivalently, vectors)
- Premise 1: Documents in the same class form a contiguous region of space
- Premise 2: Documents from different classes don't overlap (much)
- Learning a classifier: build surfaces to delineate classes in the space

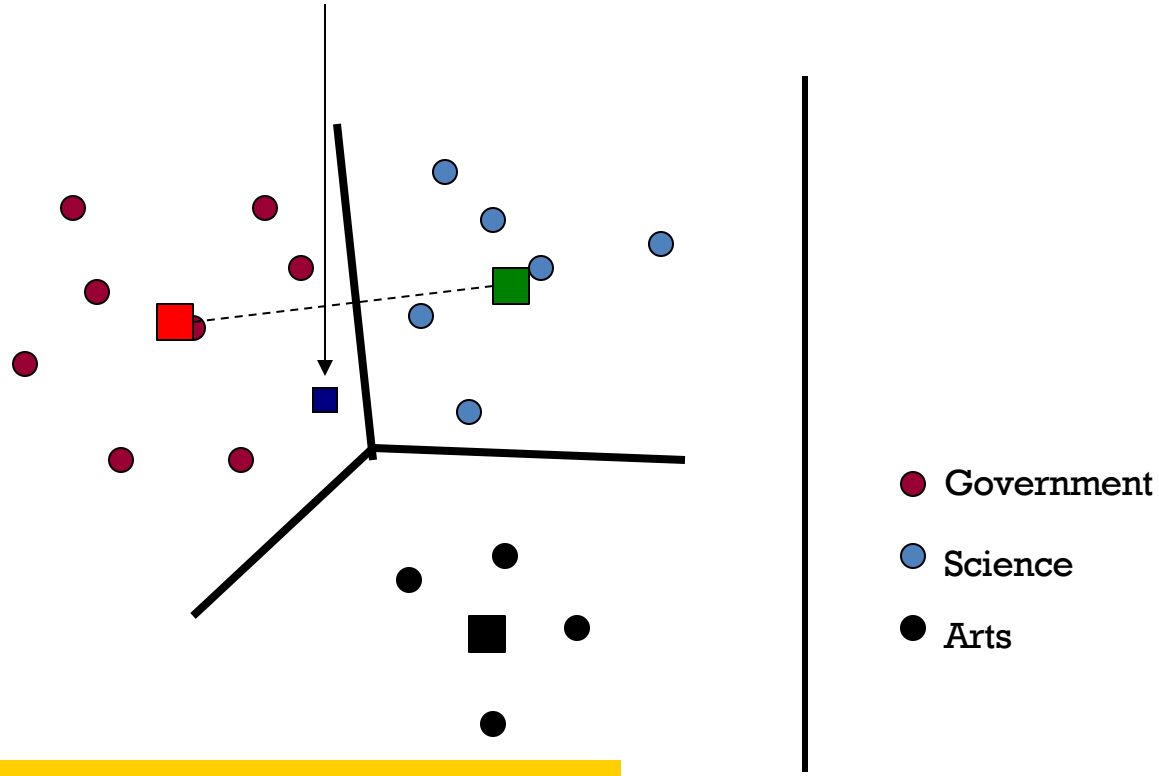
Documents in a Vector Space



Test Document of what class?



Test Document = Government



Our focus: how to find good separators

Definition of centroid

$$\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d)$$

- Where D_c is the set of all documents that belong to class *c* and *v*(*d*) is the vector space representation of *d.*

- *Note that centroid will in general* not *be a unit vector even when the inputs are unit vectors.*

Rocchio classification

- Rocchio forms a simple representative for each class: the centroid/prototype
- Classification: nearest prototype/centroid
- It does not guarantee that classifications are consistent with the given training data

Two-class Rocchio as a linear classifier

- Line or hyperplane defined by:

$$\sum_{i=1}^{M} w_i d_i = \theta$$

- For Rocchio, set:

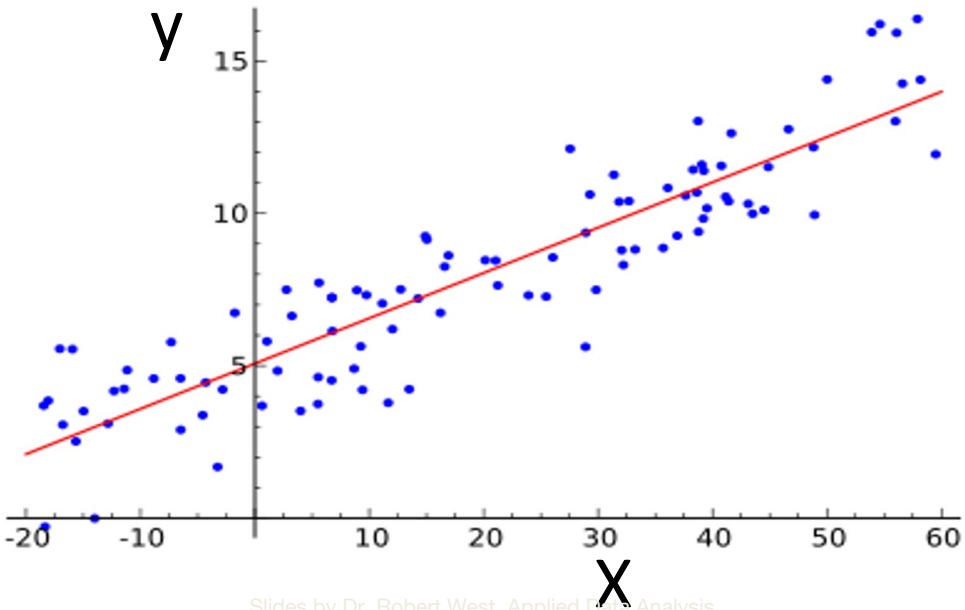
$$\vec{w} = \vec{\mu}(c_1) - \vec{\mu}(c_2)$$

$$\theta = 0.5 \times (|\vec{\mu}(c_1)|^2 - |\vec{\mu}(c_2)|^2)$$

Linear and logistic regression

Linear regression

- Goal: find the “best” line (linear function y=f(X)) to explain the data

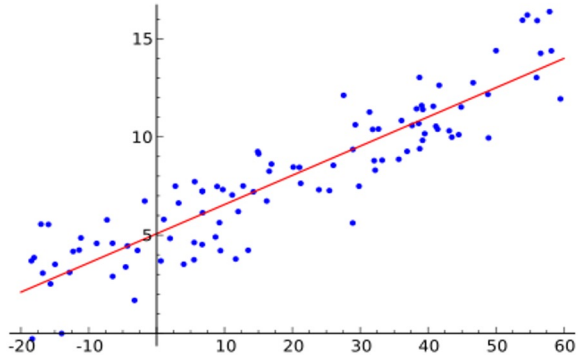


Linear regression

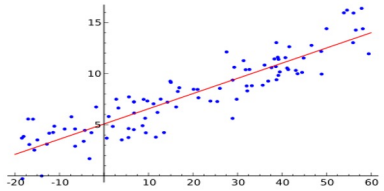
The predicted value of y is given by:

$$\hat{y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$$

The vector of coefficients $\hat{\beta}$ is the regression model.



Least-squares solution



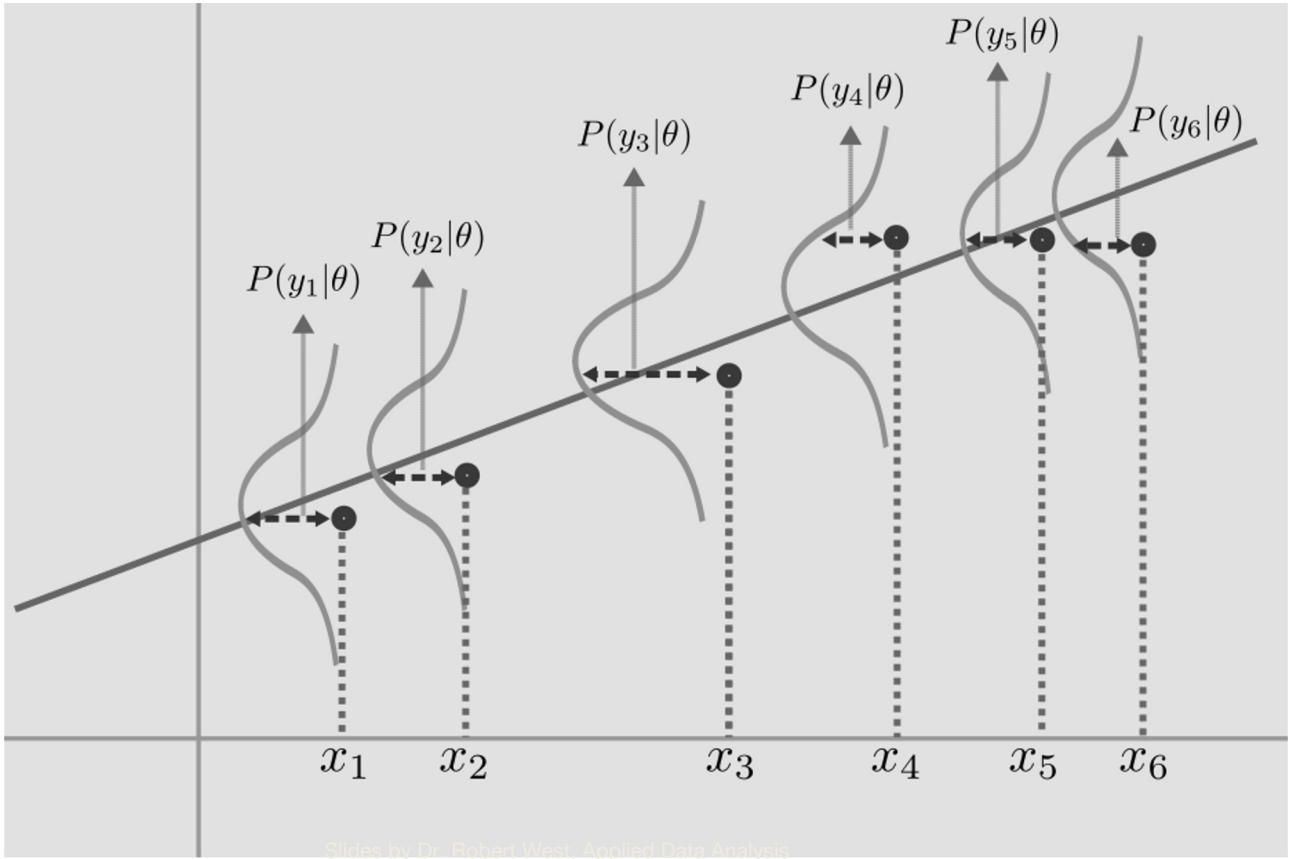
The most common measure of fit between the line and the data is the **least-squares fit**.

There is a good reason for this: If the points are generated by an ideal line with additive Gaussian noise, the least squares solution is the ***maximum likelihood solution***.

Probability of a point y_j is $\Pr(y_j) = \exp\left(\frac{-(y_j - X_j\beta)^2}{2\sigma^2}\right)$ and the probability for all points is the product over j of $\Pr(y_j)$.

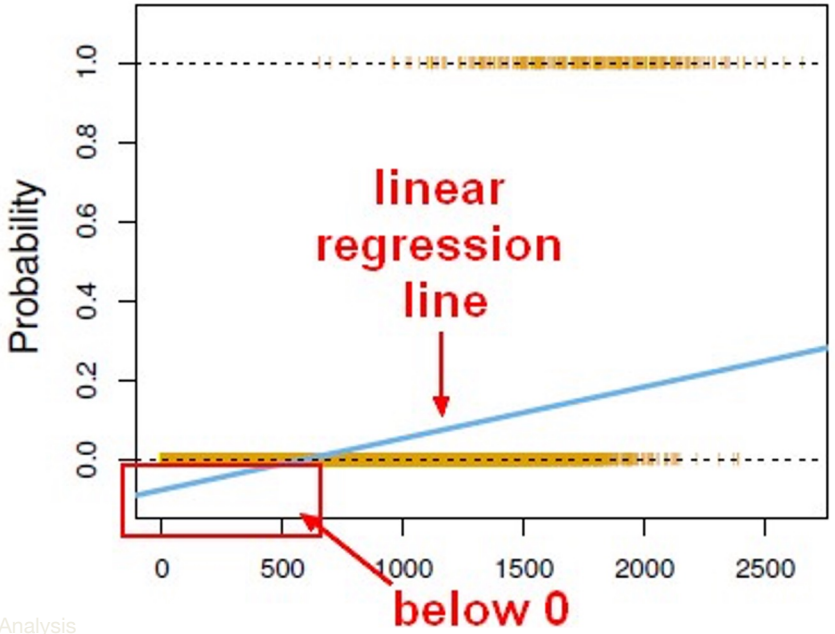
We can **easily maximize the log** of this expression $\frac{-(y_j - X_j\beta)^2}{2\sigma^2}$ for one point, or the sum of this expression at all points.

Least-squares solution

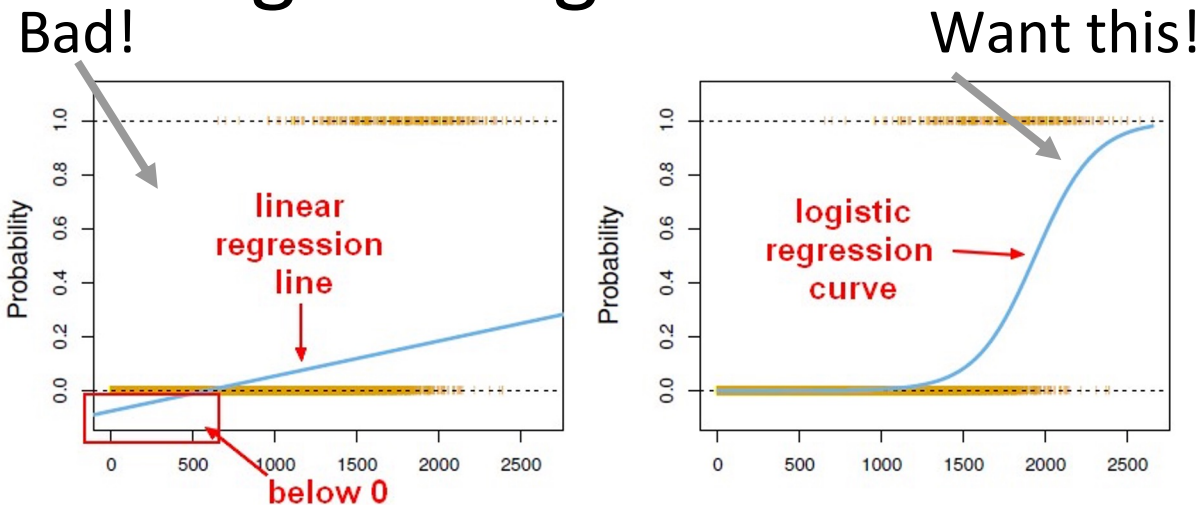


How to model binary events?

- E.g., X: student features; y: did student pass DSC510?
- Desired output: f(X) = probability of passing DSC510, given feats X
- Problem with linear regression: f(X) can be below 0 or above 1



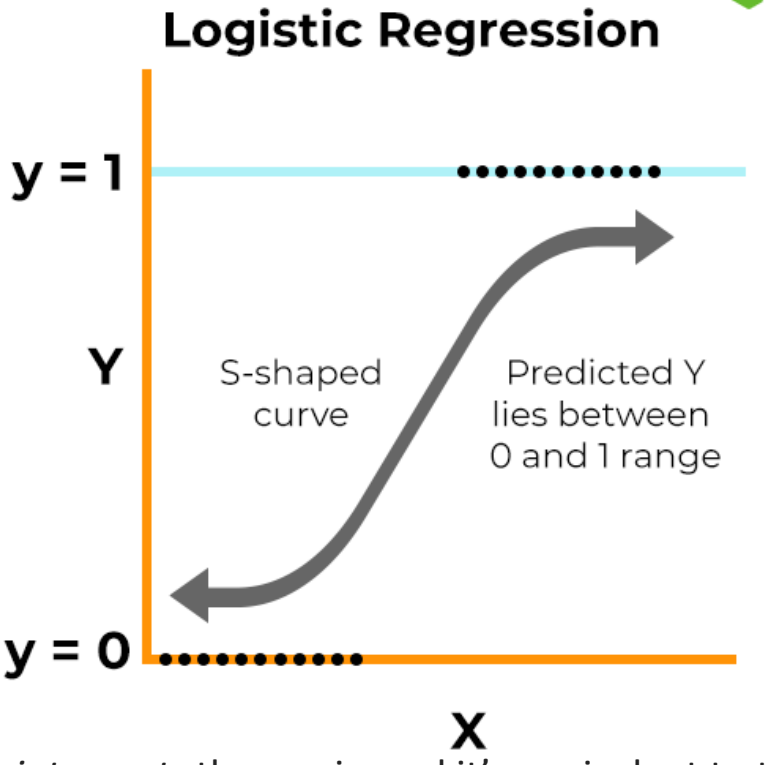
Logistic regression



- Trick: don't deal with probabilities, which range from 0 to 1, but with log odds, which range from -inf to +inf
- Probability y ⇔ odds y/(1-y) ⇔ log odds log[y/(1-y)]
- Model log odds as a linear function of X

$$y = \frac{e^{(b_0 + b_1 X)}}{1 + e^{(b_0 + b_1 X)}}$$

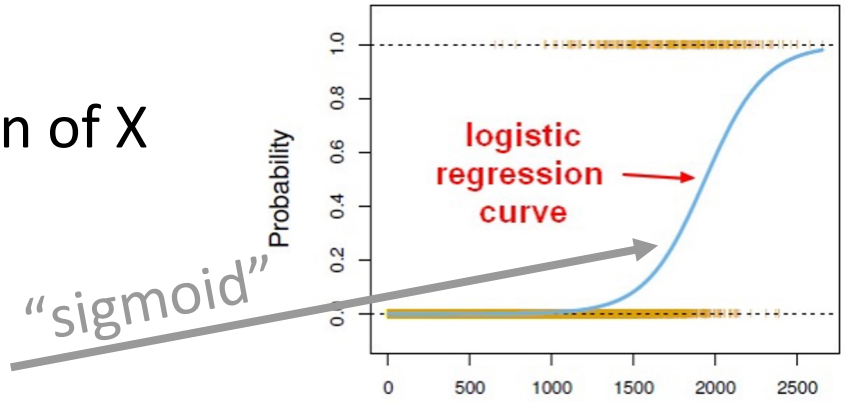
x = input value
y = predicted output
b0 = bias or intercept term
b1 = coefficient for input (x)



- b_0 is called the **intercept**. This is where the line *intercepts* the y-axis, and it's equivalent to the predicted value of *y* when *x*=0
- b_1 is the **coefficient** of the input feature *x*, and it's the slope of the line. It represents the effect x has on y. Therefore the linear regression model assumes that if x increases by 1, y increases by β_1 (This is only true when x and y have a perfect linear relationship, which is rarely the case)

Logistic regression

- Model log odds as a linear function of X
- $\beta^T X = \log[y/(1-y)]$
- Solve for y: $y = 1 / (1 + \exp(-\beta^T X))$
- Finding best model β via maximum likelihood:
  - Don't use square loss as in linear regression
  - Use cross-entropy loss instead



TOOLBOX™

KEY ADVANTAGES OF LOGISTIC REGRESSION

TOOLBOX™

KEY ASSUMPTIONS FOR IMPLEMENTING LOGISTIC REGRESSION

The dependent/response variable is binary or dichotomous

Little or no multicollinearity between the predictor/explanatory variables

Linear relationship of independent variables to log odds

Requires sufficiently large sample size

No extreme outliers

Should have independent observations

TOOLBOX™

LOGISTIC REGRESSION BEST PRACTICES

Identify dependent variables to ensure the model's consistency

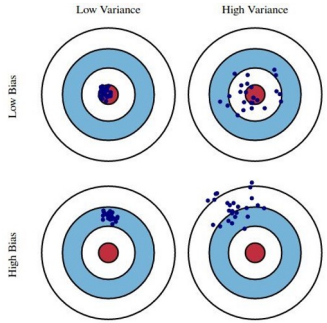
Discover the technical requirements of the model

Estimate the model and evaluate the goodness of the fit

Appropriately interpret the results

Validate observed results

Overfitting



- The more features the better?
  - **NO!**
  - More features mean less bias, but more variance
  - Overfitting
- Carefully selected features can improve model accuracy
  - E.g., keep features that correlate with the label y
  - Forward/backward feature selection
  - Regularization (e.g., penalize norm of weight vector)

(1958)
F. Rosenblatt

The perceptron: a probabilistic model for information storage and organization in the brain
Psychological Review 65:386–408

Linear models: Perceptron

- **Example: Spam filtering**

	viagra	learning	the	dating	nigeria	*spam?*
$\vec{x}_1 = ($	1	0	1	0	0 $)$	$y_1 = 1$
$\vec{x}_2 = ($	0	1	1	0	0 $)$	$y_2 = -1$
$\vec{x}_3 = ($	0	0	0	0	1 $)$	$y_3 = 1$

- **Instance space x ∈ X** (|**X**| = **n** data points)
  - **Binary or real-valued feature vector *x* of word occurrences**
  - ***d*** features (words + other things, **d**~100,000)
- **Class y ∈ Y**
  - ***y***: Spam (+1), Ham (-1)

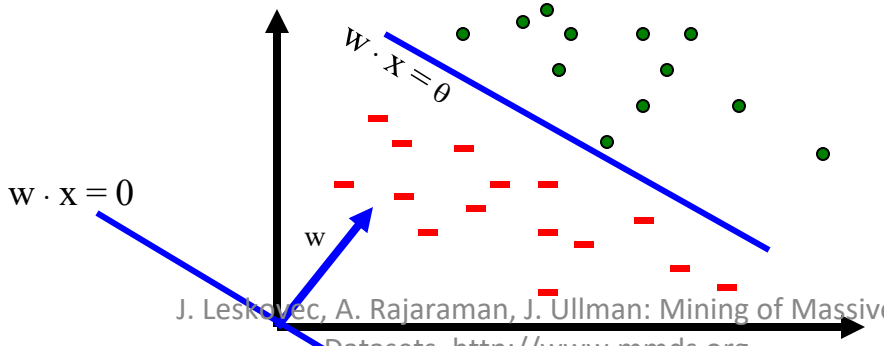
Linear models for classification

- **Binary classification:**

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{w}_1 \mathbf{x}_1 + \mathbf{w}_2 \mathbf{x}_2 + \ldots \mathbf{w}_d \mathbf{x}_d \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

Decision boundary is **linear**

- **Input:** Vectors $\boldsymbol{x}^{(j)}$ and labels $\boldsymbol{y}^{(j)}$
  - Vectors $\boldsymbol{x}^{(j)}$ are real valued where $\|\boldsymbol{x}\|_2 = 1$
- **Goal:** Find vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_d)$
  - Each $\boldsymbol{w}_i$ is a real number



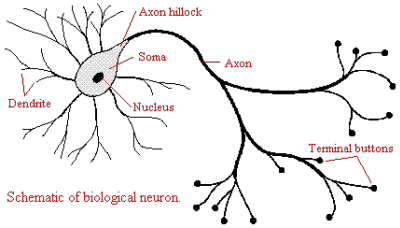
Note:

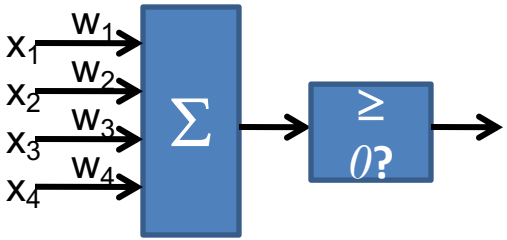
$$\mathbf{x} \Leftrightarrow \langle \mathbf{x}, 1 \rangle \quad \forall \mathbf{x}$$

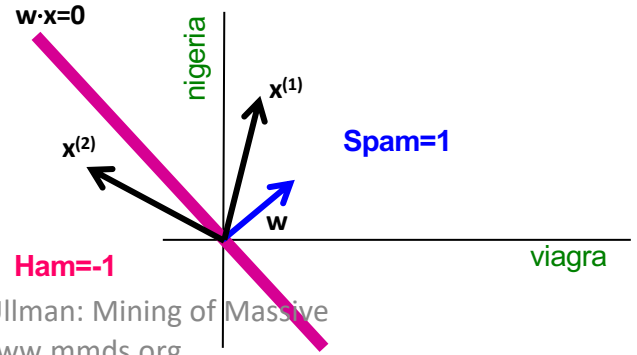
$$\mathbf{w} \Leftrightarrow \langle \mathbf{w}, -\theta \rangle$$

Perceptron [Rosenblatt '58]

- **(very) Loose motivation: Neuron**
- Inputs are feature values
- Each feature has a weight w_i
- **Activation is the sum:**
  - $f(x) = \Sigma_i w_i x_i = w \cdot x$
- If the ***f(x)*** is:
  - **Positive:** Predict **+1**
  - **Negative:** Predict **-1**







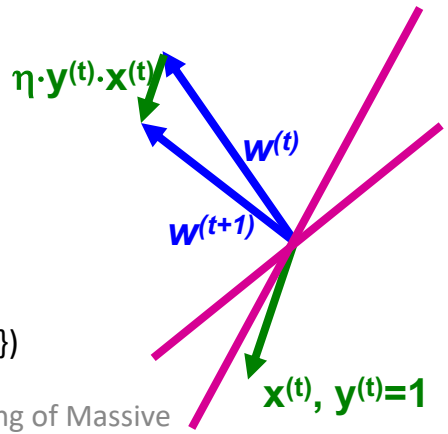
Perceptron: Estimating *w*

Note that the Perceptron is a conservative algorithm: it ignores samples that it classifies correctly.

- **Perceptron:** $y' = sign(w \cdot x)$
- **How to find parameters *w*?**
  - Start with $w_0 = 0$
  - Pick training examples $x^{(t)}$ **one by one (from disk)**
  - Predict class of $x^{(t)}$ using current weights
    - $y' = sign(w^{(t)} \cdot x^{(t)})$
  - **If *y'* is correct (i.e., $y_t = y'$)**
    - No change: $w^{(t+1)} = w^{(t)}$
  - **If *y'* is wrong:** adjust $w^{(t)}$

$$w^{(t+1)} = w^{(t)} + \eta \cdot y^{(t)} \cdot x^{(t)}$$

  - η is the learning rate parameter
  - $x^{(t)}$ is the t-th training example
  - $y^{(t)}$ is true t-th class label ({+1, -1})



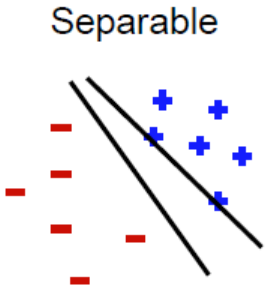
Perceptron Convergence

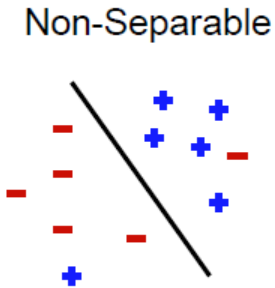
- **Perceptron Convergence Theorem:**
  - If there exist a set of weights that are consistent (i.e., the data is linearly separable) the Perceptron learning algorithm will converge
- **How long would it take to converge?**
- **Perceptron Cycling Theorem:**
  - If the training data is not linearly separable the Perceptron learning algorithm will eventually repeat the same set of weights and therefore enter an infinite loop
- **How to provide robustness, more expressivity?**

Properties of Perceptron

- **Separability:** Some parameters get training set perfectly
- **Convergence:** If training set is separable, perceptron will converge
- **(Training) Mistake bound:**
  Number of mistakes $< \frac{1}{\gamma^2}$
  - **where** $\gamma = \min_{t,u} |x^{(t)} u|$

  and $\|u\|_2 = 1$
    - Note we assume **x** Euclidean length **1**, then **γ** is the minimum distance of any example to plane **u**





Updating the Learning Rate

- **Perceptron will oscillate and won't converge**
- **When to stop learning?**
- **(1)** Slowly decrease the learning rate η
  - A classic way is to: $\eta = c_1/(t + c_2)$
    - But, we also need to determine constants c_1 and c_2
- **(2)** Stop when the training error stops chaining
- **(3)** Have a small test dataset and stop when the test set error stops decreasing
- **(4)** Stop when we reached some maximum number of passes over the data

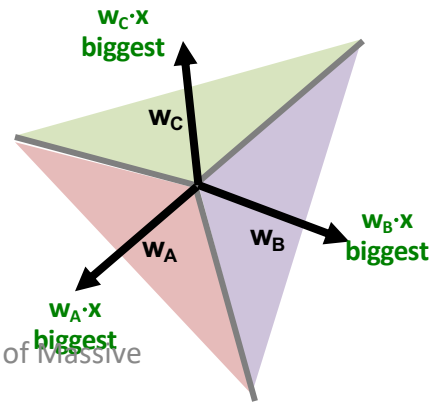
Multiclass Perceptron

- **What if more than 2 classes?**
- Weight vector $\boldsymbol{w_c}$ for each class **c**
  - **Train one class vs. the rest:**
    - Example: 3-way classification **y = {A, B, C}**
    - Train 3 classifiers: $\mathbf{w_A}$: A vs. B,C; $\mathbf{w_B}$: B vs. A,C; $\mathbf{w_C}$: C vs. A,B
- **Calculate activation for each class**

$$f(x,c) = \Sigma_i \; w_{c,i} \; x_i \; = \; w_c \cdot x$$

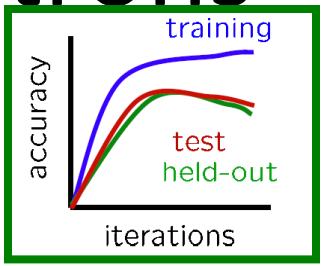
- **Highest activation wins**

$$c = \arg\max_c f(x,c)$$

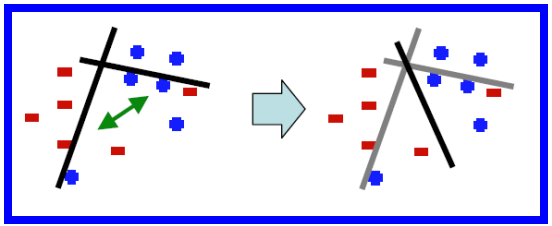


Issues with Perceptrons

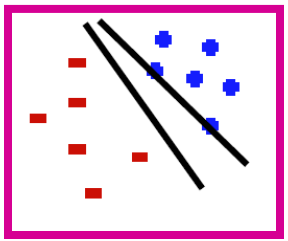
- **Overfitting:**



- **Regularization:** If the data is not separable weights dance around



- **Mediocre generalization:**
  - Finds a "barely" separating solution



Improvement: Winnow Algorithm

- **Winnow :** Predict **f(x) = +1** iff $\boldsymbol{w} \cdot \boldsymbol{x} \geq \boldsymbol{\theta}$
  - Similar to perceptron, just different updates
  - **Assume *x* is a real-valued feature vector,** $\|\boldsymbol{x}\|_2 = 1$
    - Initialize: $\boldsymbol{\theta} = \frac{d}{2},\ \boldsymbol{w} = \left[\frac{1}{d}, \ldots, \frac{1}{d}\right]$
    - **For every training example** $x^{(t)}$
      - **Compute** $y' = f(x^{(t)})$
      - If no mistake ($y^{(t)} = y'$): do nothing
      - If mistake then: $w_i \leftarrow w_i \frac{\exp\left(\eta y^{(t)} x_i^{(t)}\right)}{Z^{(t)}}$
  - **w** … weights **(can never get negative!)**
  - $Z^{(t)} = \sum_i w_i \exp\left(\eta y^{(t)} x_i^{(t)}\right)$ is the normalizing const.

Improvement: Winnow Algorithm

- **About the update:** $w_i \leftarrow w_i \frac{\exp\left(\eta y^{(t)} x_i^{(t)}\right)}{Z^{(t)}}$
  - If **x** is false negative, increase $\mathbf{w_i}$ **(promote)**
  - If **x** is false positive, decrease $\mathbf{w_i}$ **(demote)**
  - **In other words:** Consider $\boldsymbol{x_i^{(t)}} \in \{-1, +1\}$
  - Then $\boldsymbol{w_i^{(t+1)}} \propto \boldsymbol{w_i^{(t)}}$.
    $$\begin{cases} \boldsymbol{e^{\eta}} & if\ x_i^{(t)} = y^{(t)} \\ \boldsymbol{e^{-\eta}} & else \end{cases}$$
    - **Notice: This is a weighted majority algorithm of "experts" x_i agreeing with y**

Perceptron vs. Winnow

- **How to compare learning algorithms?**
- **Considerations:**
  - Number of features **d** is **very large**
  - **The instance space is sparse**
    - Only few features per training example are non-zero
  - **The model is sparse**
    - Decisions depend on a small subset of features
    - In the "true" model on a few $\mathbf{w}_i$ are non-zero
  - Want to learn from a number of examples that is small relative to the dimensionality **d**

Perceptron vs. Winnow

Perceptron

- **Online:** Can adjust to changing target, over time
- **Advantages**
  - Simple
  - Guaranteed to learn a linearly separable problem
  - **Advantage with few relevant features per training example**
- **Limitations**
  - Only linear separations
  - Only converges for linearly separable data
  - Not really "efficient with many features"

Winnow

- **Online:** Can adjust to changing target, over time
- **Advantages**
  - Simple
  - Guaranteed to learn a linearly separable problem
  - **Suitable for problems with many irrelevant attributes**
- **Limitations**
  - Only linear separations
  - Only converges for linearly separable data
  - Not really "efficient with many features"

Online Learning

- **New setting: Online Learning**
  - Allows for modeling problems where we have a continuous stream of data
  - We want an algorithm to learn from it and slowly adapt to the changes in data
- **Idea: Do slow updates to the model**
  - Both our methods Perceptron and Winnow make updates if they misclassify an example
  - **So:** First train the classifier on training data. Then for every example from the stream, if we misclassify, update the model (using small learning rate)

Example: Shipping Service

- **Protocol:**
  - User comes and tell us origin and destination
  - We offer to ship the package for some money ($10 - $50)
  - Based on the price we offer, sometimes the user uses our service (**y = 1**), sometimes they don't (**y = -1**)
- **Task:** Build an algorithm to optimize what price we offer to the users
- **Features *x* capture:**
  - Information about user
  - Origin and destination
- **Problem: Will user accept the price?**

Example: Shipping Service

- **Model whether user will accept our price:**
  y = f(x; w)
  - **Accept: y =1**, **Not accept: y=-1**
  - Build this model with say Perceptron or Winnow
- **The website that runs continuously**
- **Online learning algorithm would do something like**
  - User comes
  - She is represented as an **(x,y)** pair where
    - **x:** Feature vector including price we offer, origin, destination
    - **y:** If they chose to use our service or not
  - The algorithm updates **w** using just the **(x,y)** pair
  - Basically, we update the **w** parameters every time we get some new data

Example: Shipping Service

- We discard this idea of a data "set"
- Instead we have a continuous stream of data
- **Further comments:**
  - For a major website where you have a massive stream of data then this kind of algorithm is pretty reasonable
  - Don't need to deal with all the training data
  - If you had a small number of users you could save their data and then run a normal algorithm on the full dataset
    - Doing multiple passes over the data

Online Algorithms

- An online algorithm can adapt to changing user preferences
- For example, over time users may become more price sensitive
- **The algorithm adapts and learns this**
- So the system is dynamic