Classification

(Slides largely based on Prof. Sandra Avila's Machine Learning Course)

Prof. Rosa Paccotacya Yanque

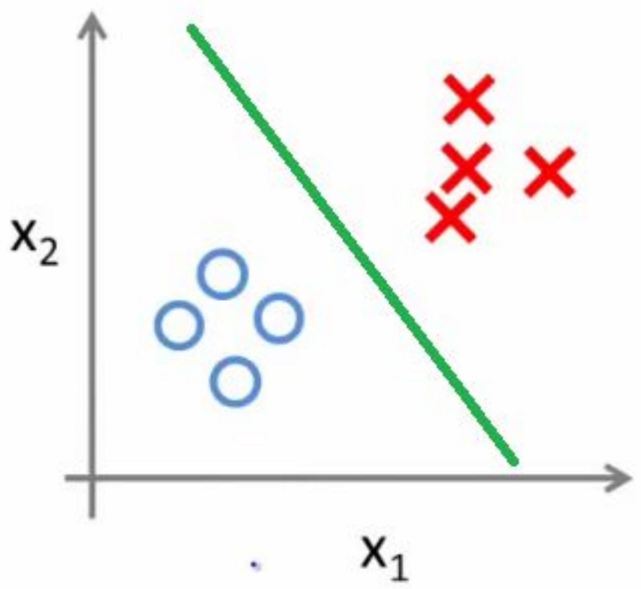
Agenda

- Logistic Regression
- Ensemble Learning
- Support Vector Machines (SVMs)
- Decision Tree
- Random Forest

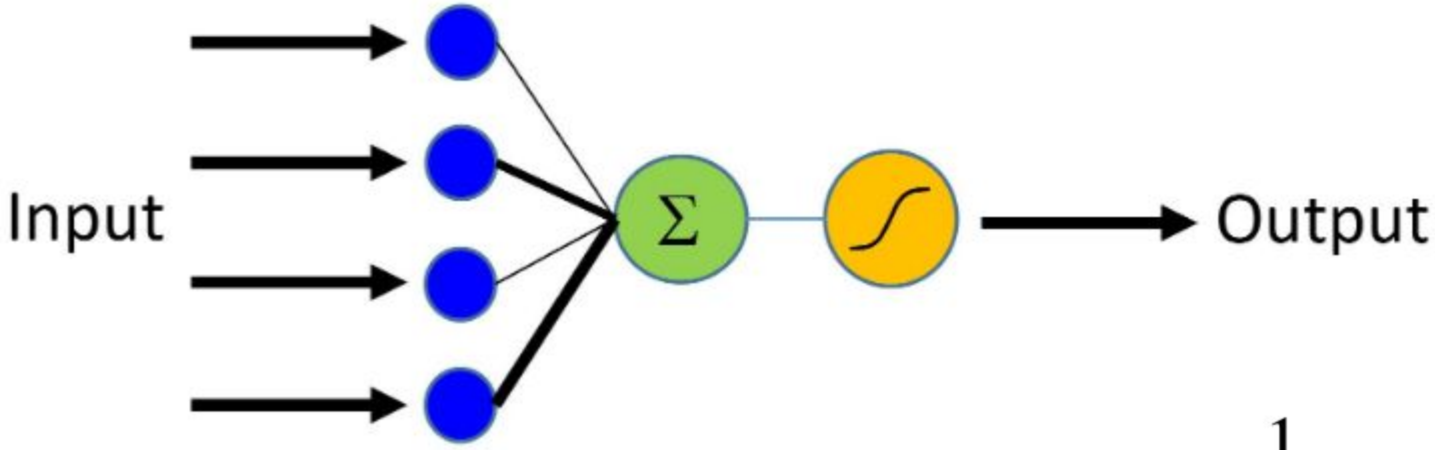
Logistic Regression

Binary Classification

- Transactions(fraudulent or no)
- People with disease or no
- Tumors (benign or not)
- E-mail (spam or not)



Model



$$h_\theta(x) = g(\theta^T x) \qquad g(z) = \frac{1}{1+e^{-z}}$$

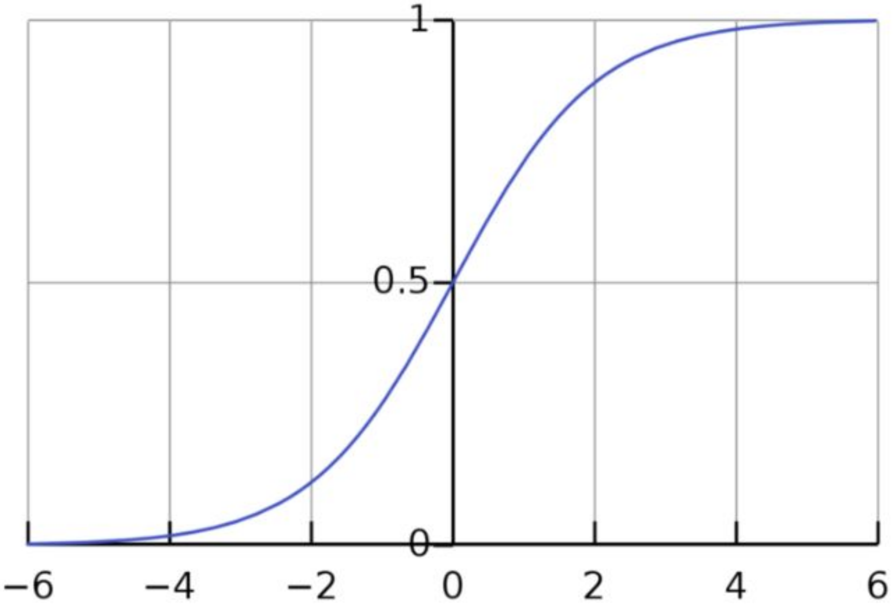
$$z = \theta^T x$$

Sigmoid or Logistic Function

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$z = \theta^T x$$

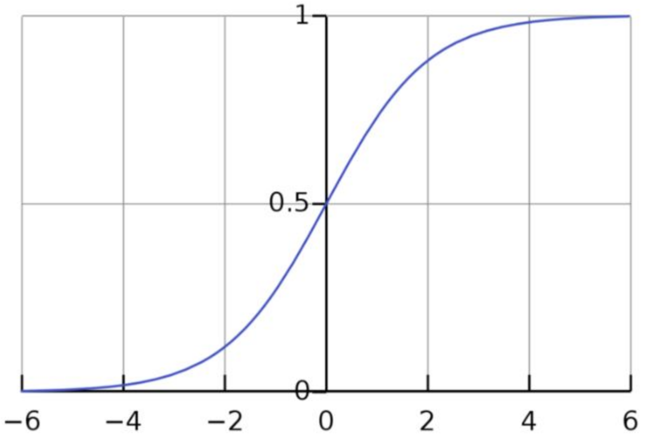
$$z = 0, e^0 = 1 \Rightarrow g(z) = 1/2$$
$$z \to \infty, e^{-\infty} \to 0 \Rightarrow g(z) = 1$$
$$z \to -\infty, e^{\infty} \to \infty \Rightarrow g(z) = 0$$



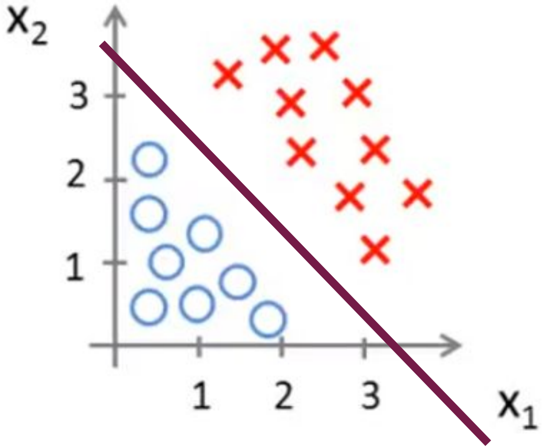
Interpretation

$$h_\theta(x) \geq 0.5 \rightarrow y = 1$$
$$h_\theta(x) < 0.5 \rightarrow y = 0$$

$$h_\theta(x) = g(\theta^T x) \geq 0.5$$
$$when\ \theta^T x \geq 0$$



$$\theta^T x \geq 0 \Rightarrow y = 1$$
$$\theta^T x < 0 \Rightarrow y = 0$$



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

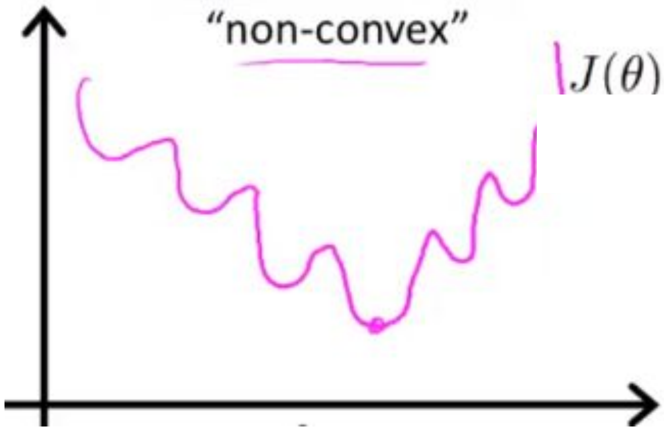
m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

How to choose parameters θ ?

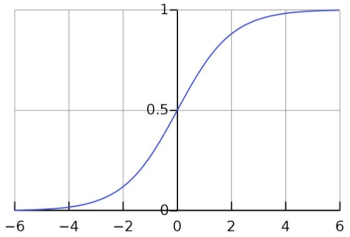
$$\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Linear regression: $J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$



Cost Function

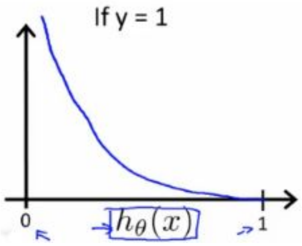
LR COST FUNCTION

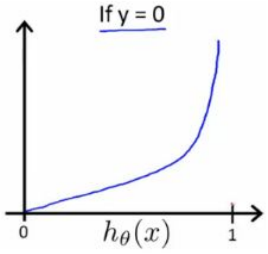


$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\mathrm{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if y} = 1$$

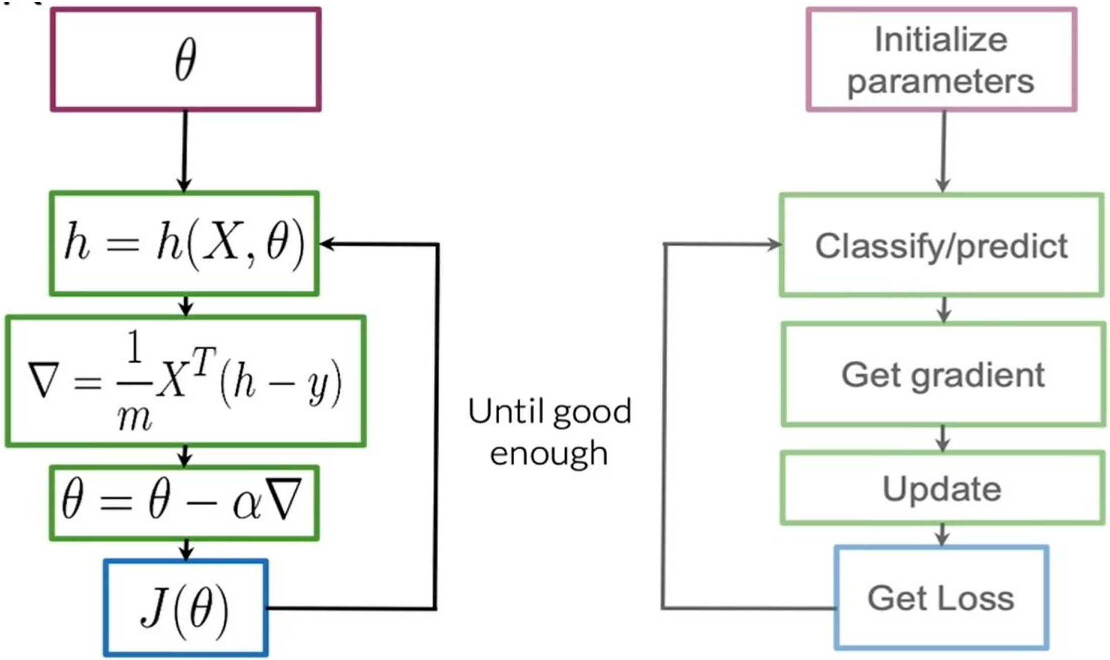
$$\mathrm{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if y} = 0$$





$$\mathrm{Cost}(h_\theta(x), y) = -y \, \log(h_\theta(x)) - (1 - y)\log(1 - h_\theta(x))$$

Training:Gradient Descent



$$
\begin{aligned}
&Repeat\ \{ \\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\
&\}
\end{aligned}
$$

Multiclass Classification

- Email tagging: Work, Friends, Family
- Skin Lesion: Melanoma, Carcinoma, Nevus, Keratosis
- Video: Pornography, Violence, Gore scenes, Child abuse

Classification

Email tagging: Work, Friends, Family

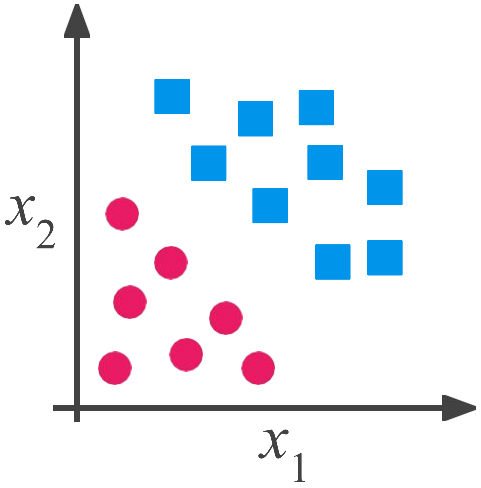
$y = 1 \; y = 2 \quad y = 3$

Skin Lesion: Melanoma, Carcinoma, Nevus, Keratosis

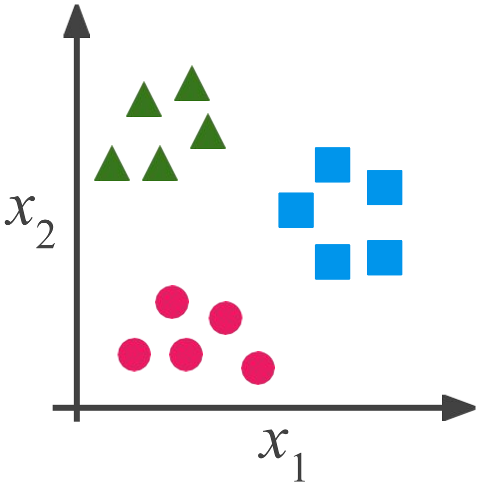
$y = 1 \; y = 2 \quad y = 3 \quad y = 4$

Video: Pornography, Violence, Gore scenes, Child abuse

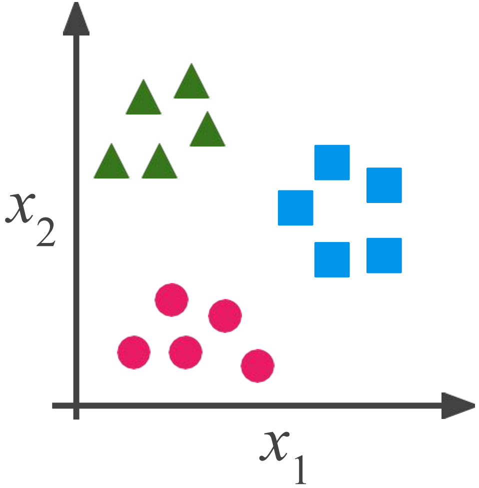
Binary Classification



Multi-class Classification



One-vs-All (One-vs-Rest)

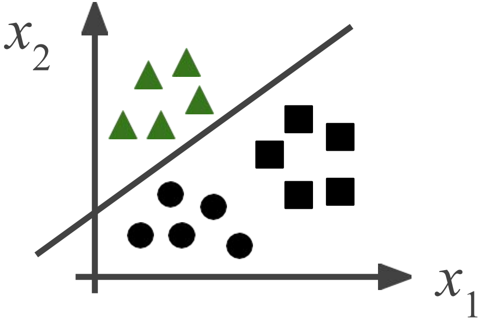


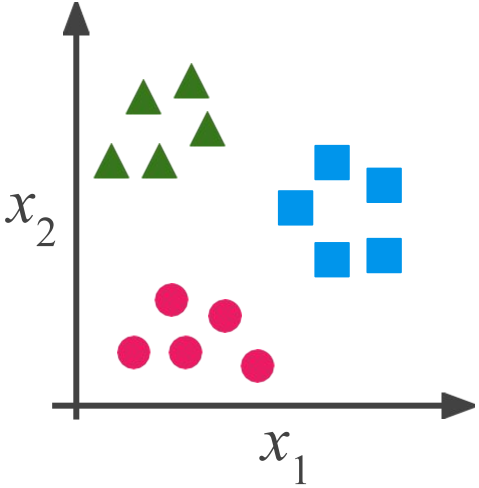
Class 1: ▲

Class 2: ■

Class 3: ●

One-vs-All (One-vs-Rest)



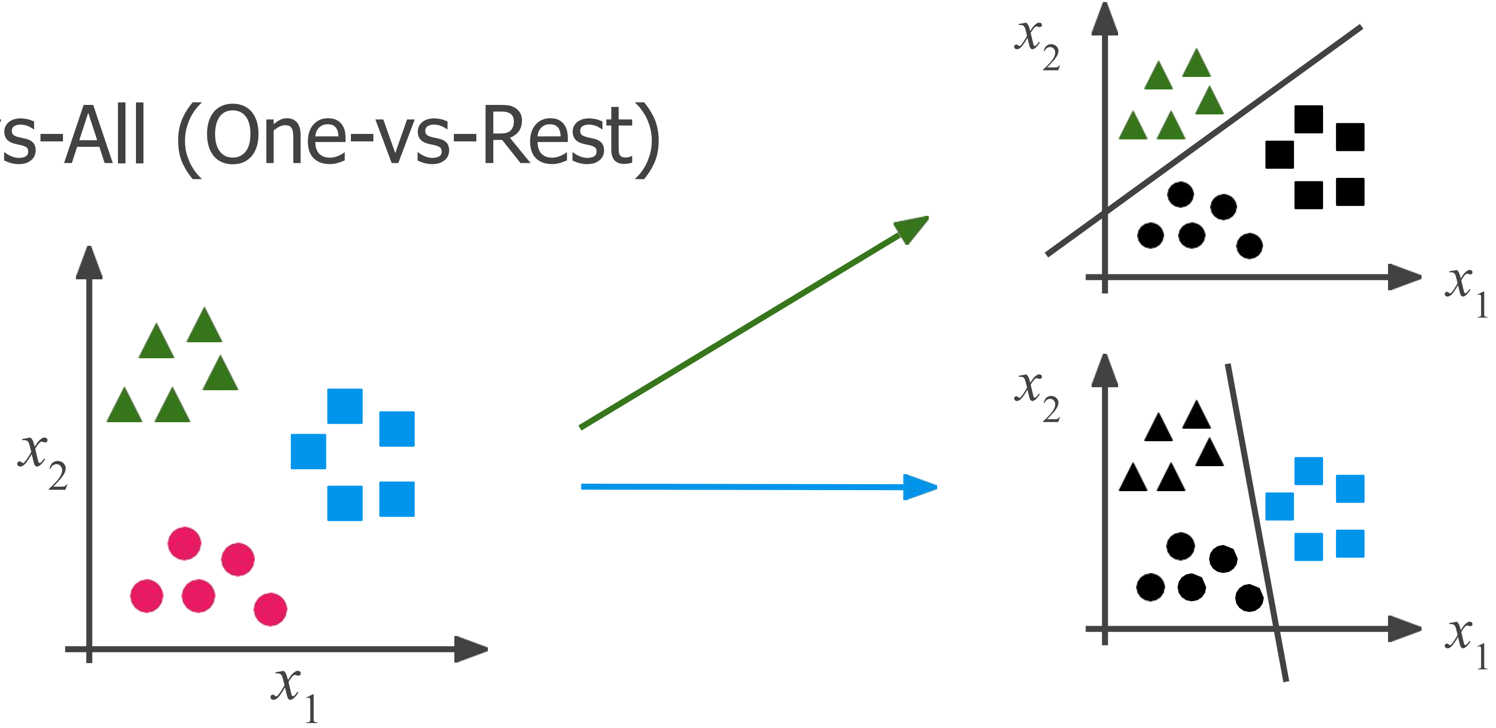


Class 1: ▲

Class 2: ■

Class 3: ●

One-vs-All (One-vs-Rest)

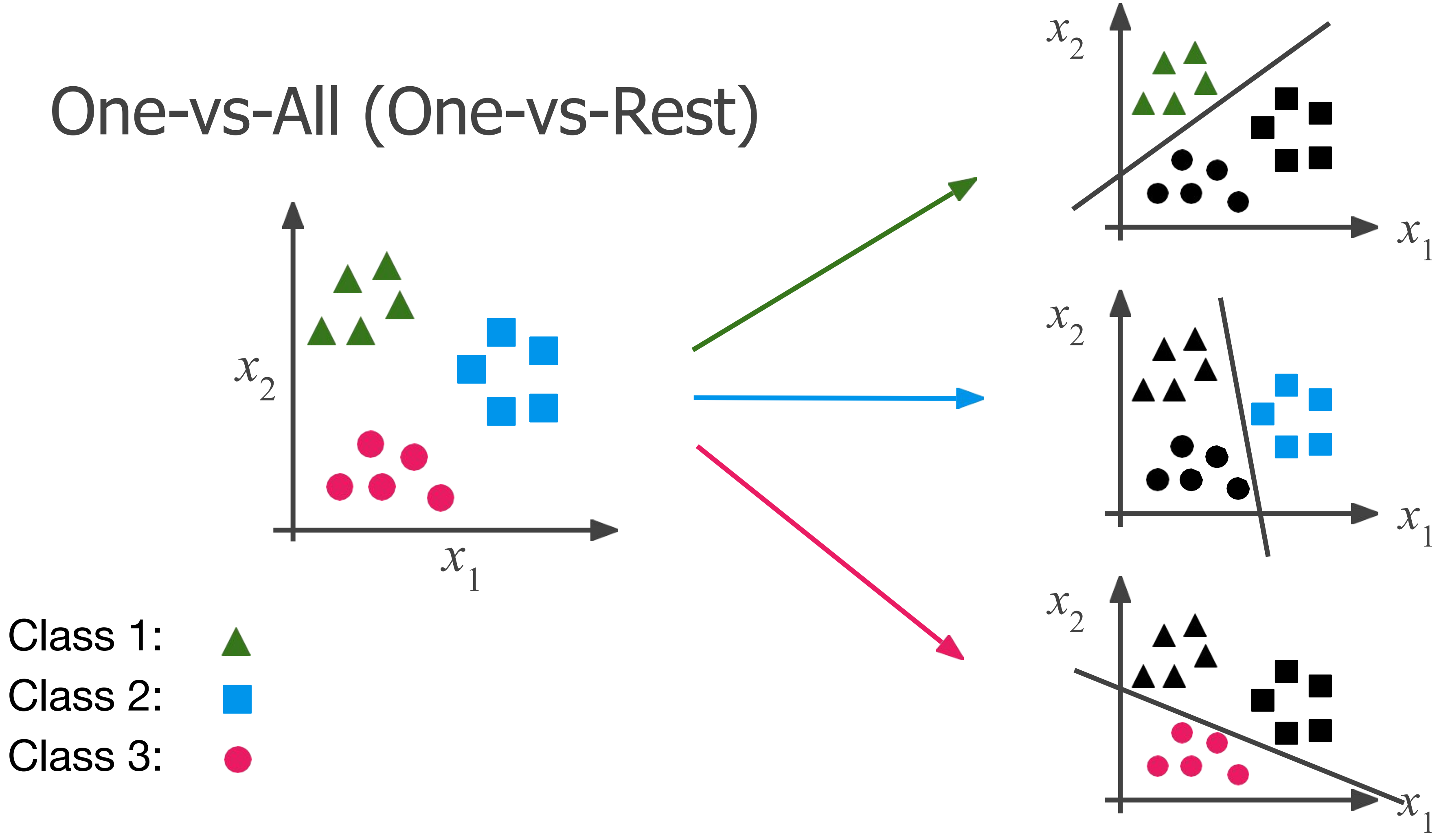


Class 1:

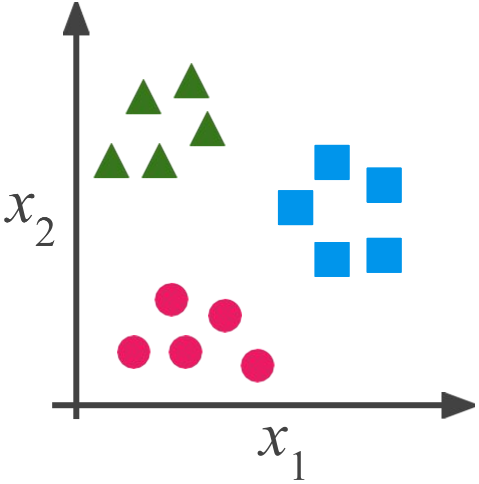
Class 2:

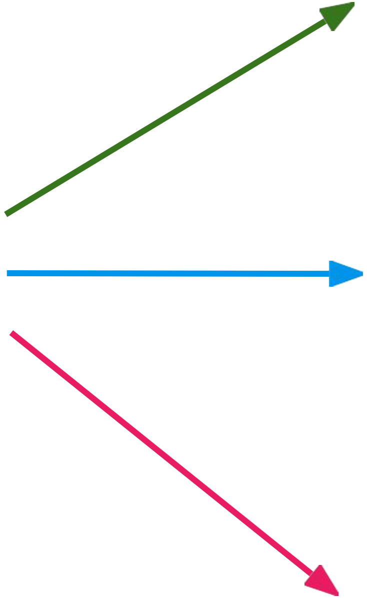
Class 3:

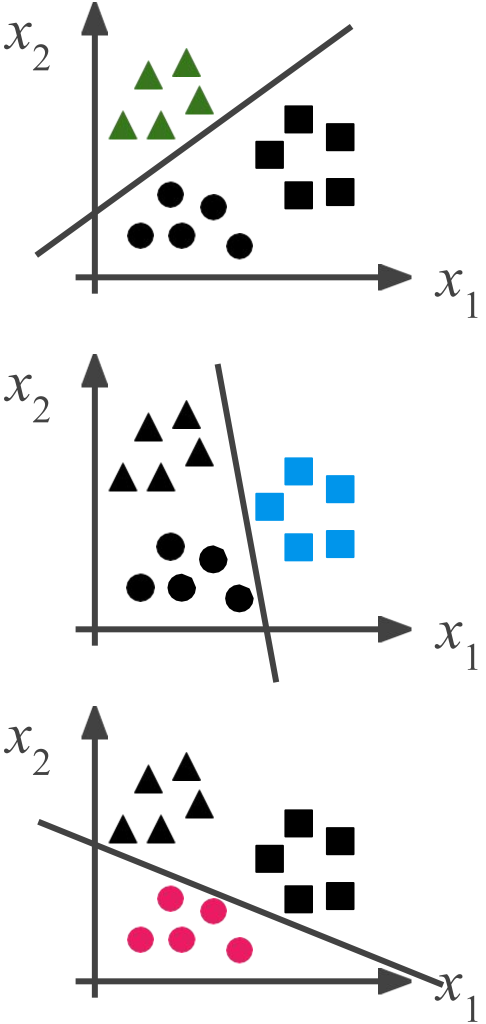
One-vs-All (One-vs-Rest)



One-vs-All (One-vs-Rest)







Class 1:

Class 2:

Class 3:

$$h_\theta^{(i)}(x) = P(y = i \mid x;\theta) \quad (i=1,2,3)$$

One-vs-All (One-vs-Rest)

Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

One a new input x, to make a prediction, pick the class i that maximizes

$$\max_i \; h_\theta^{(i)}(x)$$

References

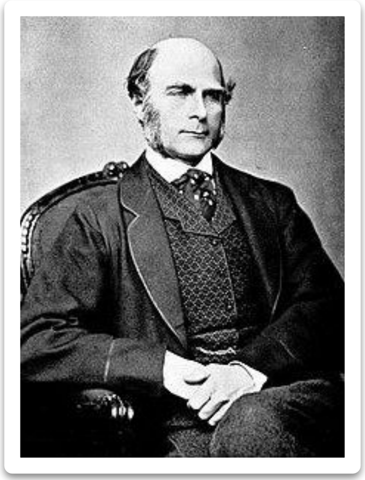
Machine Learning Books

- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Chap. 4
- Pattern Recognition and Machine Learning, Chap. 4

Machine Learning Courses

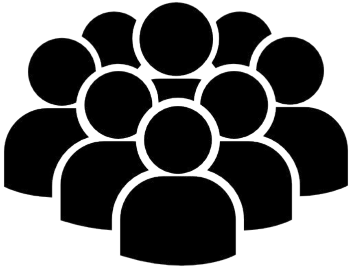
- https://www.coursera.org/learn/machine-learning, Week 3
- Logistic Regression — The Math of Intelligence (Week 2): https://youtu.be/D8alok2P468
- http://cs229.stanford.edu/notes/cs229-notes1.pdf

Ensemble Learning



Francis Galton
(1822-1909)

Animal's weight?



~800 people

1,197 kg

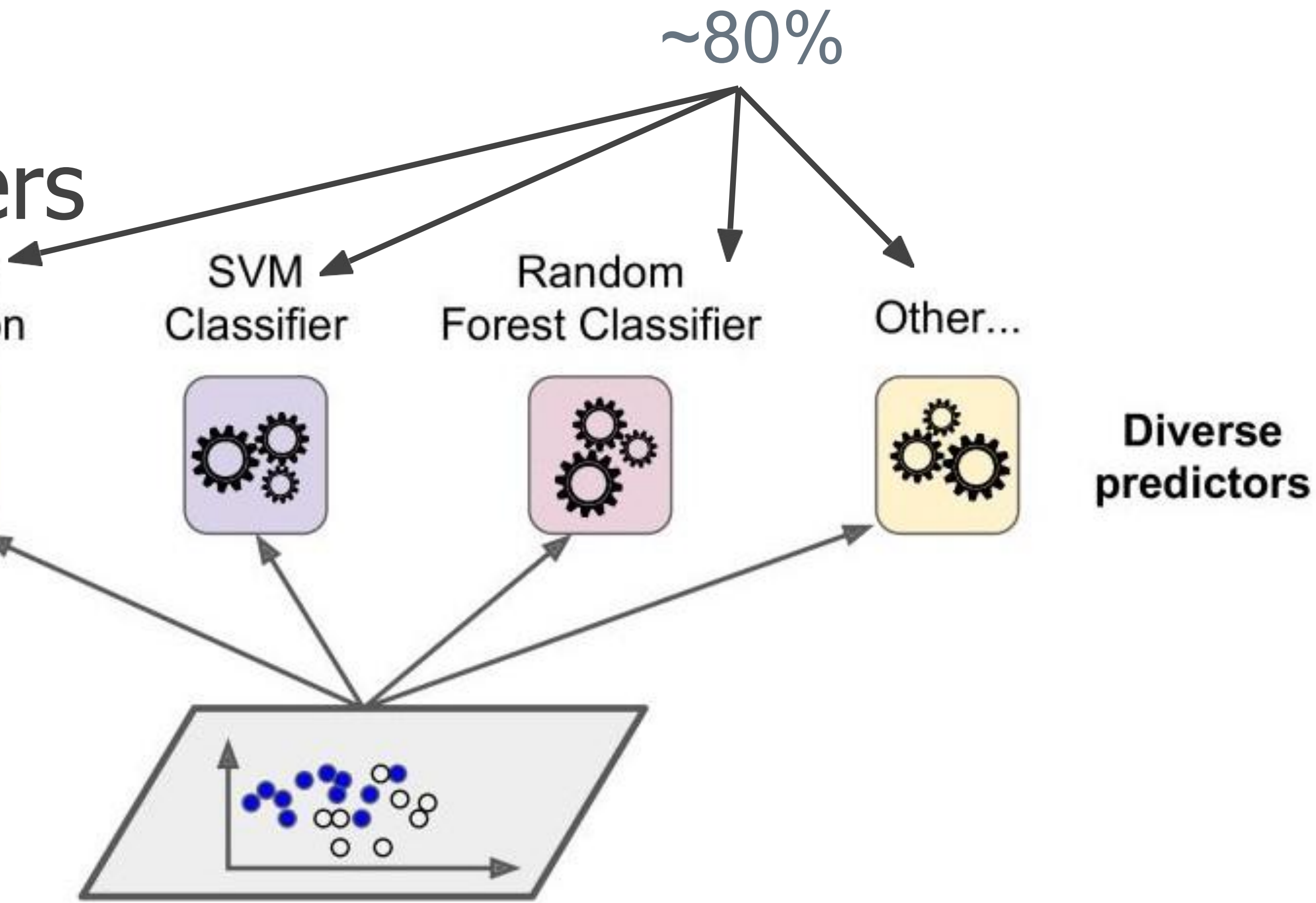
1,207 kg

Wisdom of the Crowd

Ensemble Learning

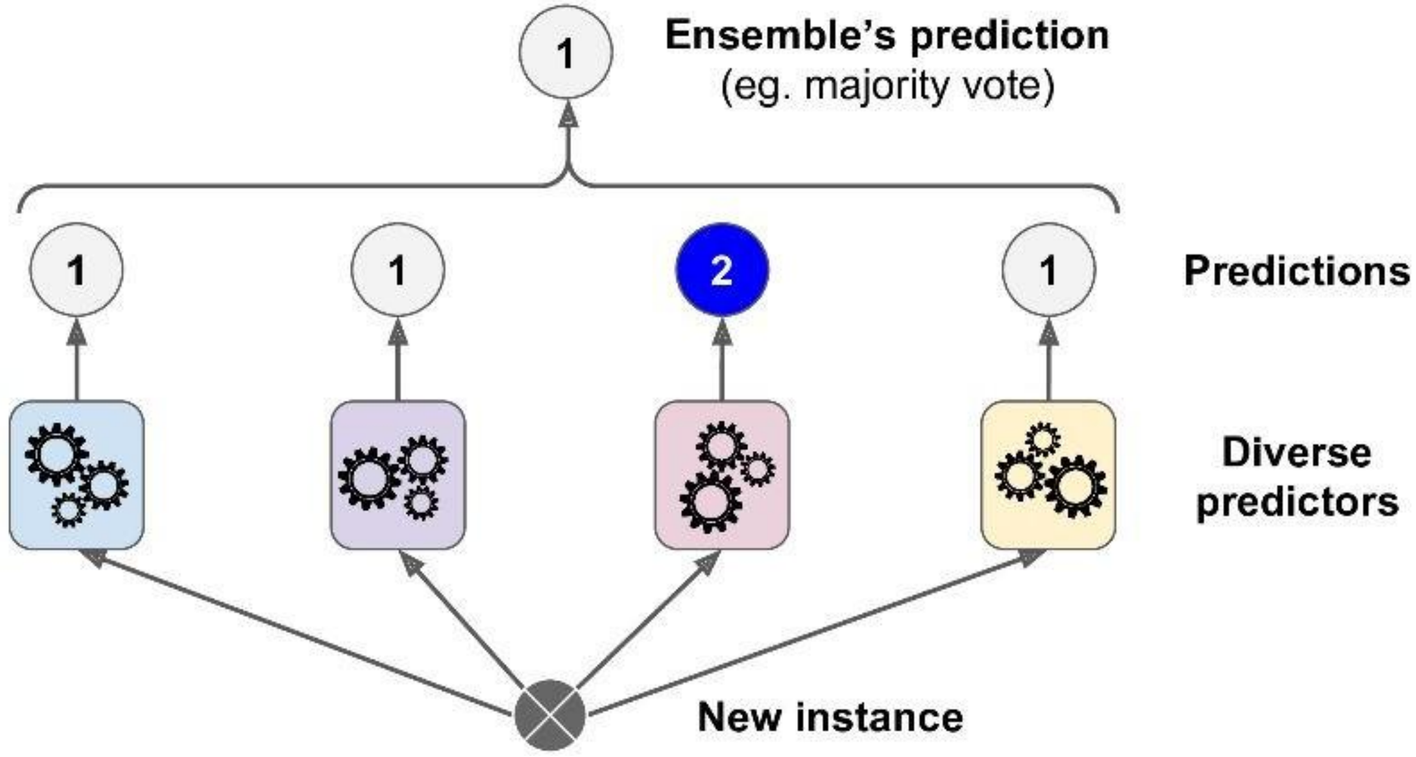
- Multiple learning algorithms **to obtain better predictive performance** than could be obtained from any learning algorithms individually.

Voting Classifiers



Voting Classifiers

Hard/Soft voting classifier



Voting Classifiers

- Voting classifier **often achieves a higher accuracy than the best classifier** in the ensemble.

Voting Classifiers

- Voting classifier **often achieves a higher accuracy than the best classifier** in the ensemble.

- Even if each classifier is a **weak learner**, the ensemble can still be a **strong learner**, provided there are a sufficient number of weak learners and they are sufficiently diverse.

Voting Classifiers

- Ensemble methods work best when the predictors are as **independent** from one another as possible.

Voting Classifiers

- Ensemble methods work best when the predictors are as **independent** from one another as possible.

- One way to get diverse classifiers is to train them using **very different algorithms**: this increases the chance that they will make very different types of errors, improving the ensemble's accuracy.

Voting Classifiers

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
        estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
                    voting='hard'

    )
voting_clf.fit(X_train, y_train)
```

Voting Classifiers

```
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.904
```

```
from sklearn.ensemble import RandomForestClassif
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()
voting_clf = VotingClassifier(
        estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
                voting='hard'
    )
voting_clf.fit(X_train, y_train)
```

Ensemble Methods

- **Bagging (and Pasting)**
- Boosting
- Stacking

Bagging and Pasting

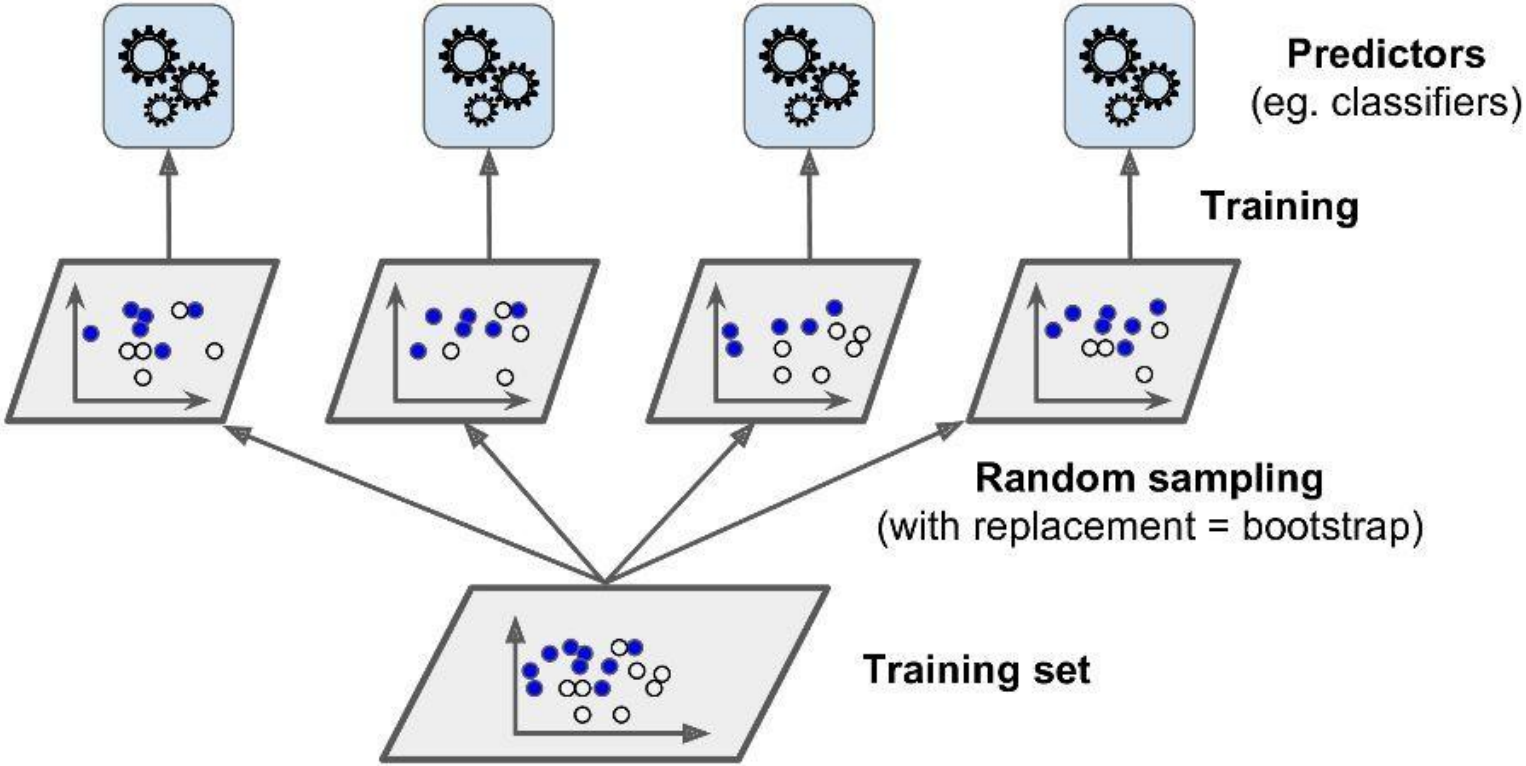
- Use **the same training algorithm** for every predictor, but to train them on **different random subsets** of the training set.

"Bagging Predictors", L. Breiman (1996): http://goo.gl/o42tml

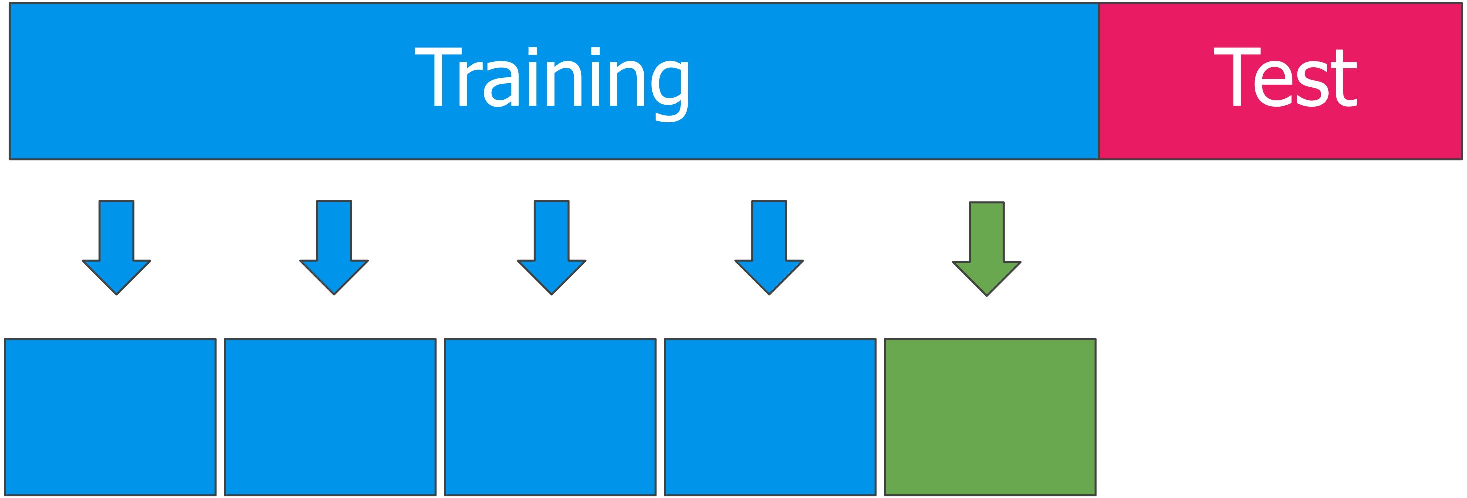
Bagging and Pasting

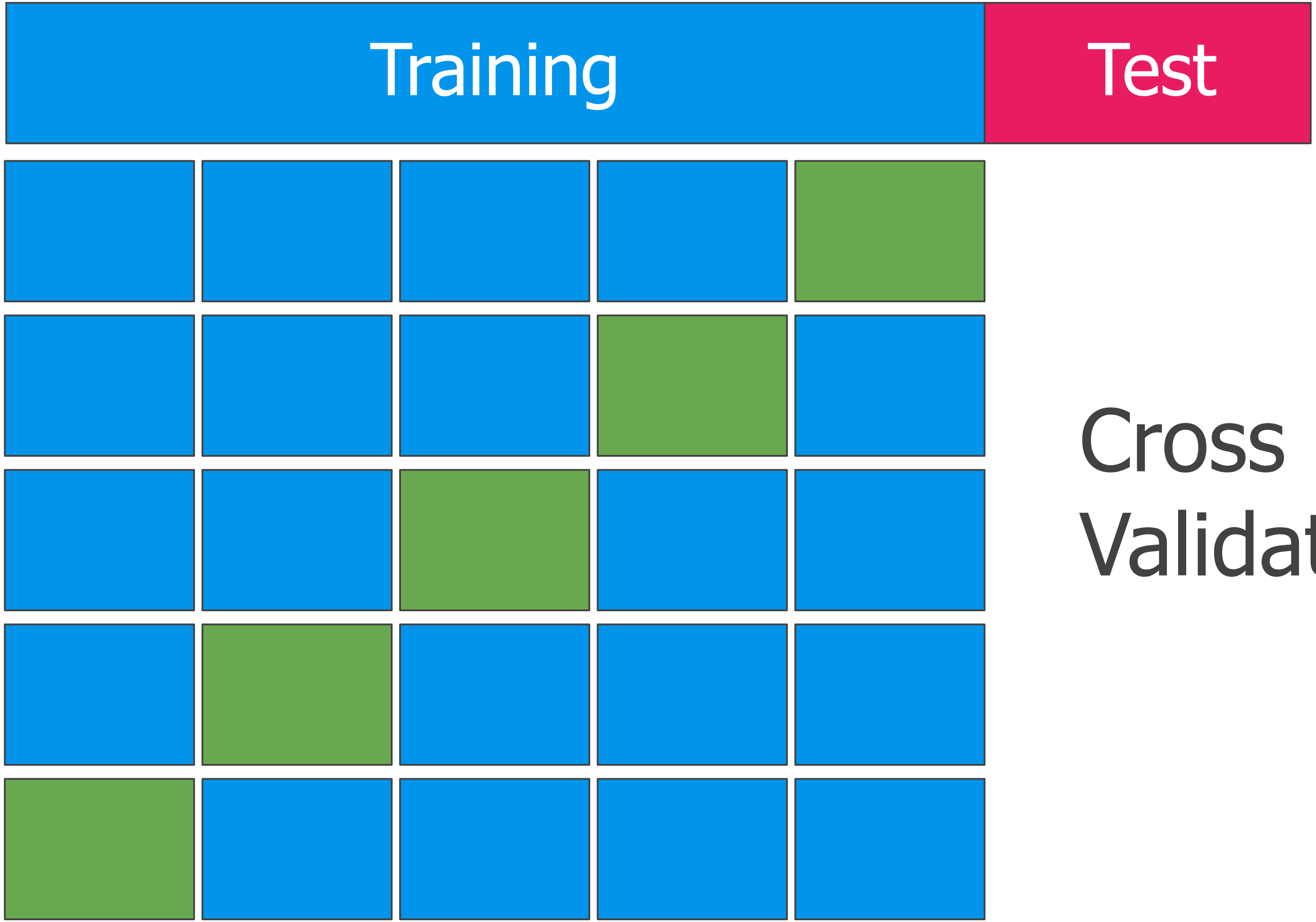
- Use **the same training algorithm** for every predictor, but to train them on **different random subsets** of the training set.
- **Bagging** (short for Bootstrap Aggregating): sampling is performed **with** replacement.
- **Pasting**: sampling is performed **without** replacement.

Bagging and Pasting

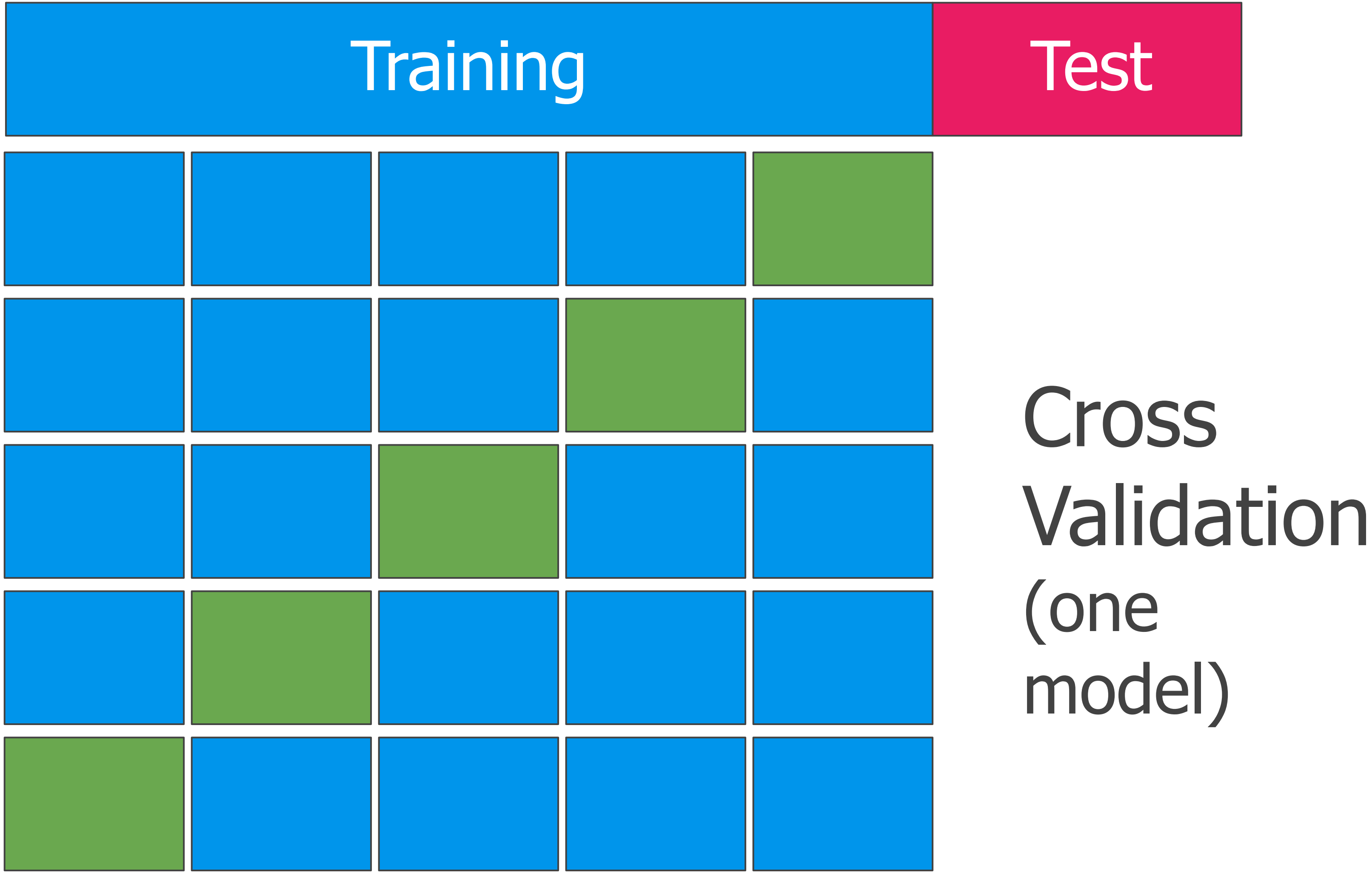


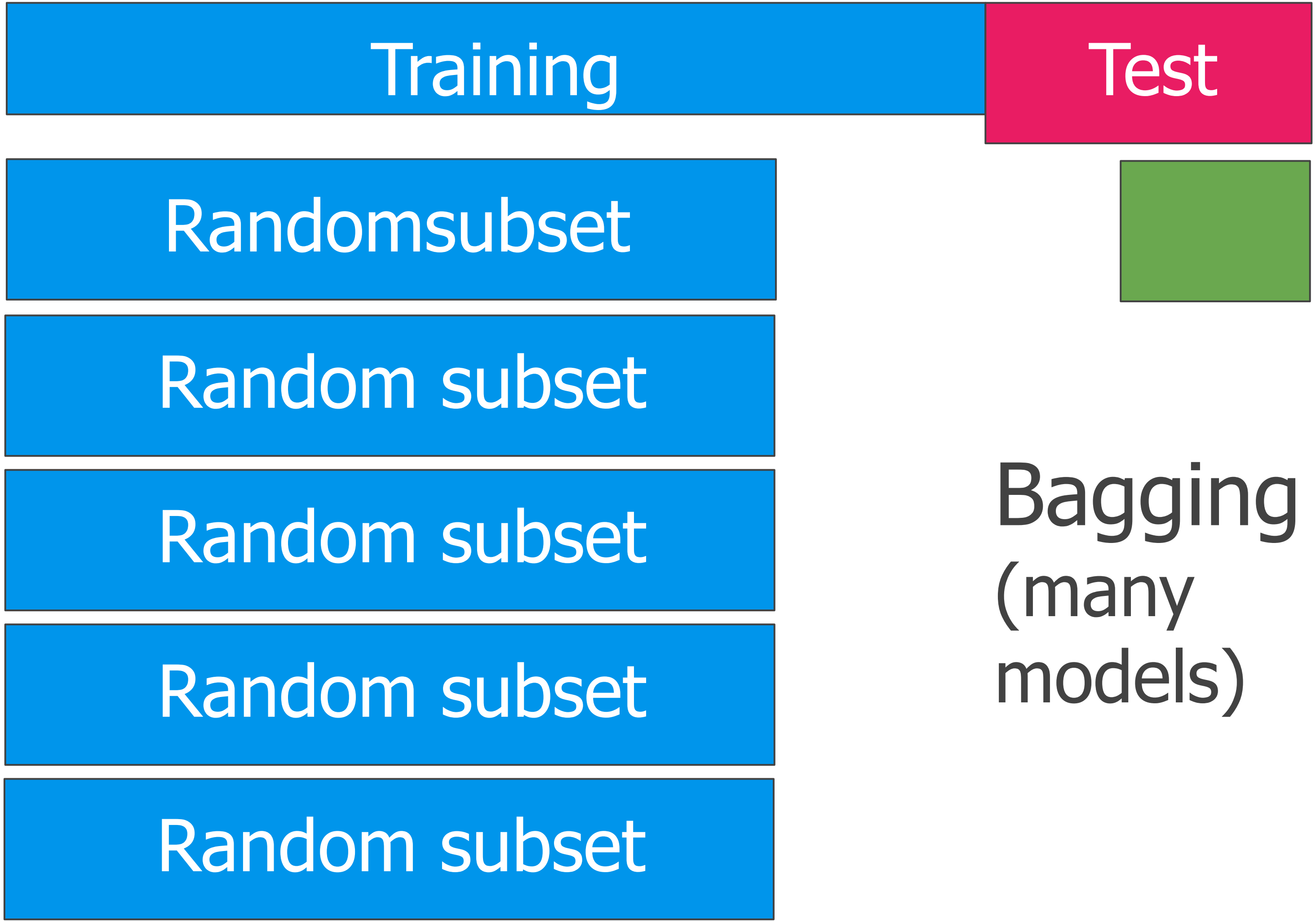
Cross Validation



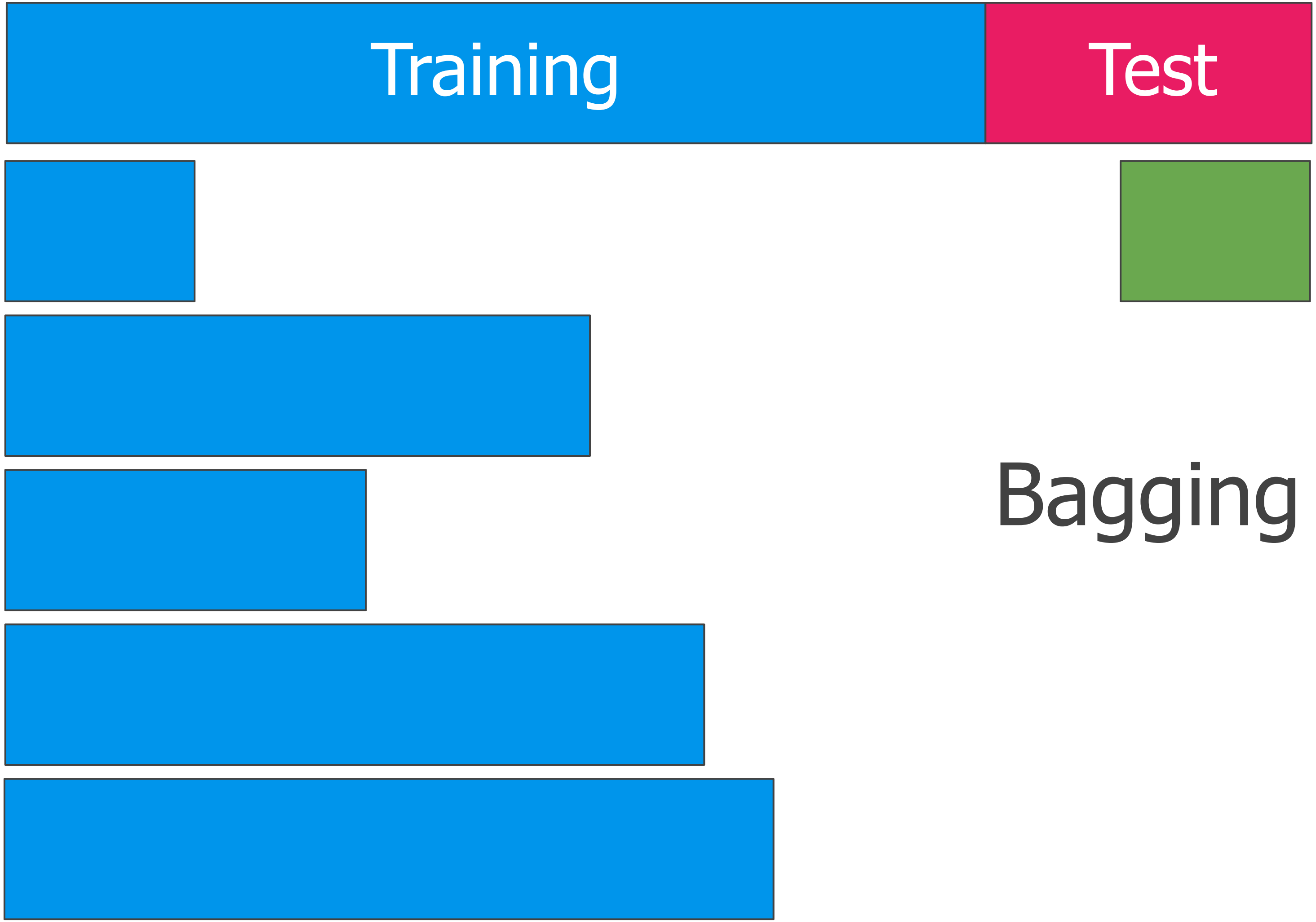


Cross Validation


Training
Test
Cross
Validation
(one
model)



Bagging
(many
models)


Training
Test
Bagging

Bagging and Pasting

- Once all predictors are trained, the ensemble can make a prediction for a new instance by simply **aggregating the predictions of all predictors**.

- **Bagging and Pasting scale very well.**

Bagging and Pasting

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
bag_clf = BaggingClassifier(
        DecisionTreeClassifier(),
        n_estimators=500, max_samples=100,
        bootstrap=True, n_jobs=-1

    )
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

Bagging and Pasting

- **Random Patches Ensemble** method: sampling **both** training instances and **features**.
- This is particularly useful when dealing with high-dimensional inputs.

"Ensembles on Random Patches", G. Louppe and P. Geurts (2012)

Ensemble Methods

- Bagging (and Pasting)
- **Boosting**
- Stacking

Slides largely based on Prof. Sandra Avila's Machine Learning Course

Boosting

- The general idea of most boosting methods is **to train predictors sequentially**, each trying to correct its predecessor.

Boosting

- The general idea of most boosting methods is **to train predictors sequentially**, each trying to correct its predecessor.

- Most popular: AdaBoost and Gradient Boost.

AdaBoost [Freund and Schapire, 1997]

- One way for a new predictor to correct its predecessor is to pay a bit **more attention** to the training instances that **the predecessor underfitted**.

“A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”, Y. Freund and R. Schapire (1997): http://goo.gl/OIduRW

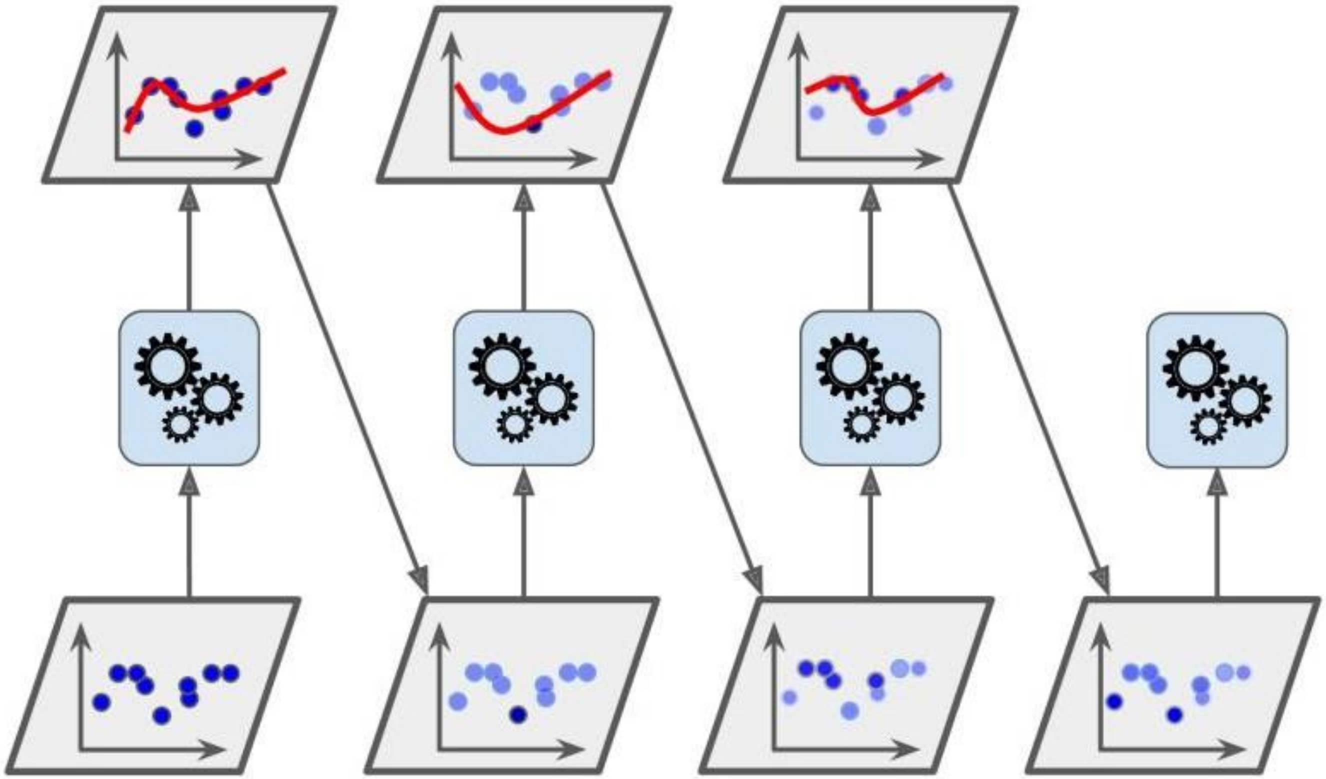
AdaBoost [Freund and Schapire, 1997]

- One way for a new predictor to correct its predecessor is to pay a bit **more attention** to the training instances that **the predecessor underfitted**.

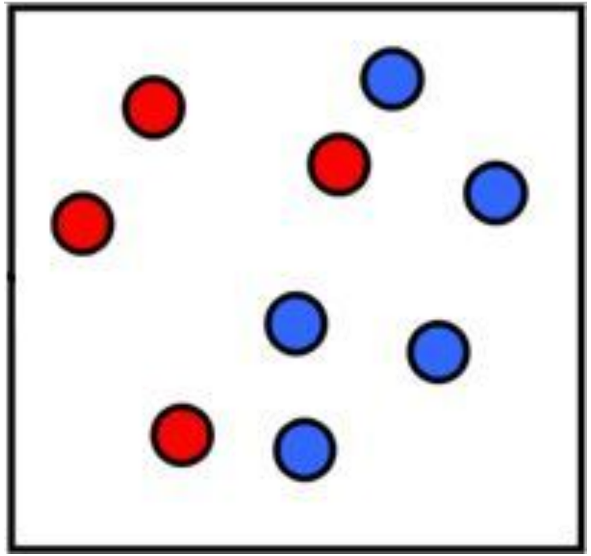
- This results in new predictors focusing more and more on **the hard cases**.

“A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”, Y. Freund and R. Schapire (1997): http://goo.gl/OIduRW

AdaBoost

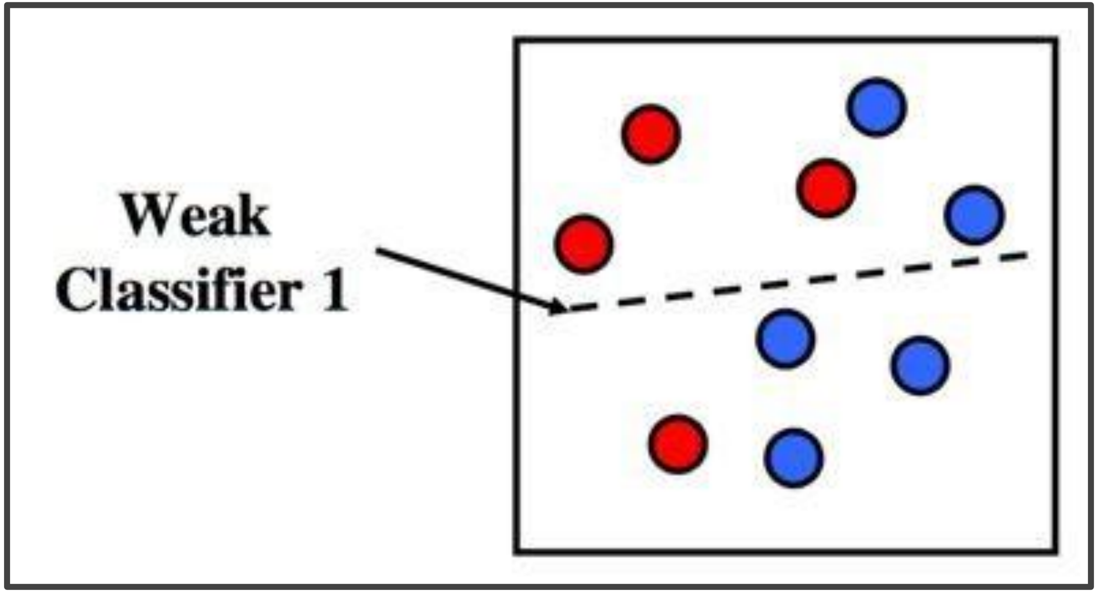


AdaBoost

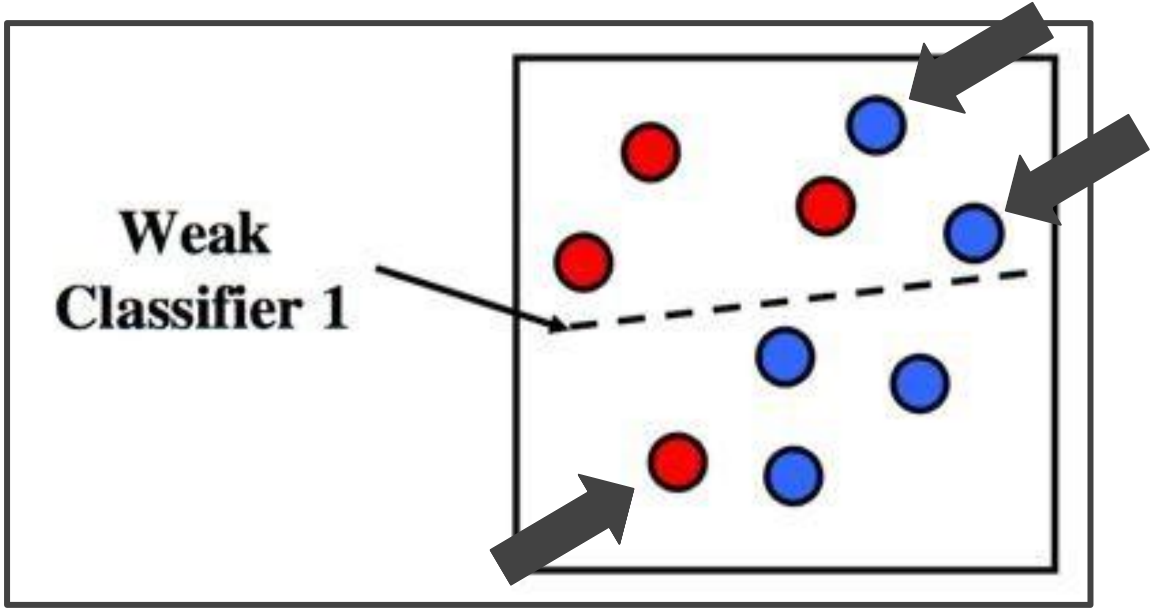


Slides largely based on Prof. Sandra Avila's Machine Learning Course

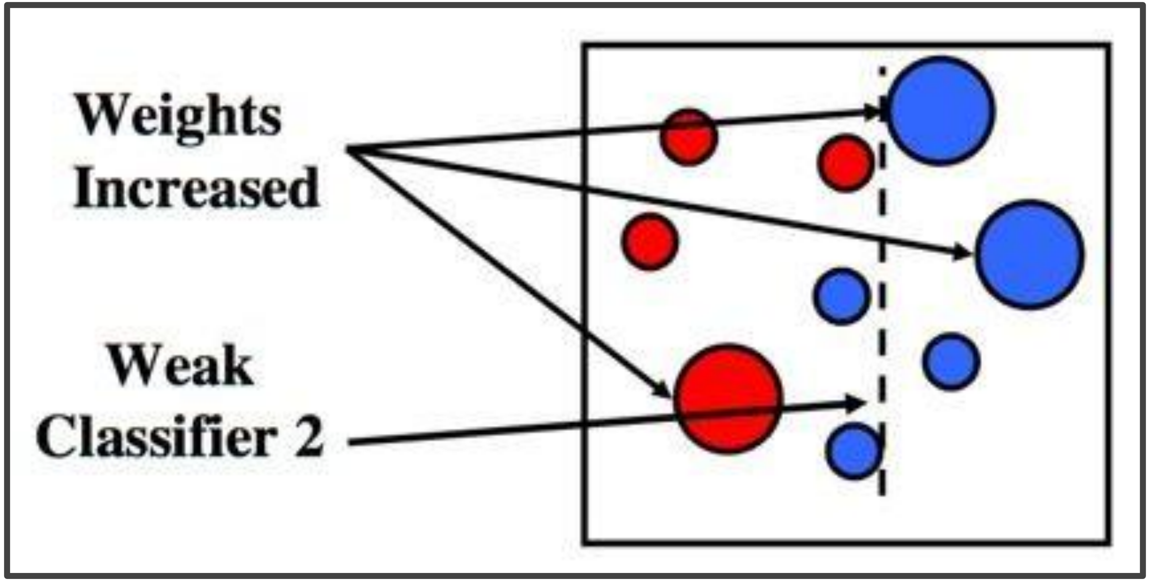
AdaBoost



AdaBoost

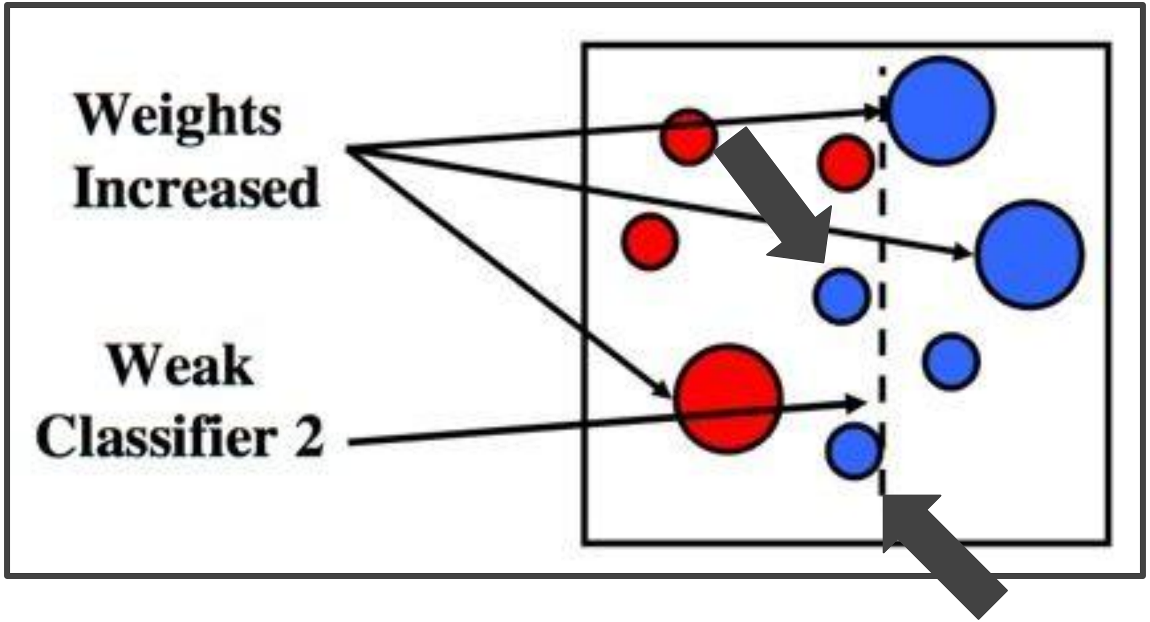


AdaBoost



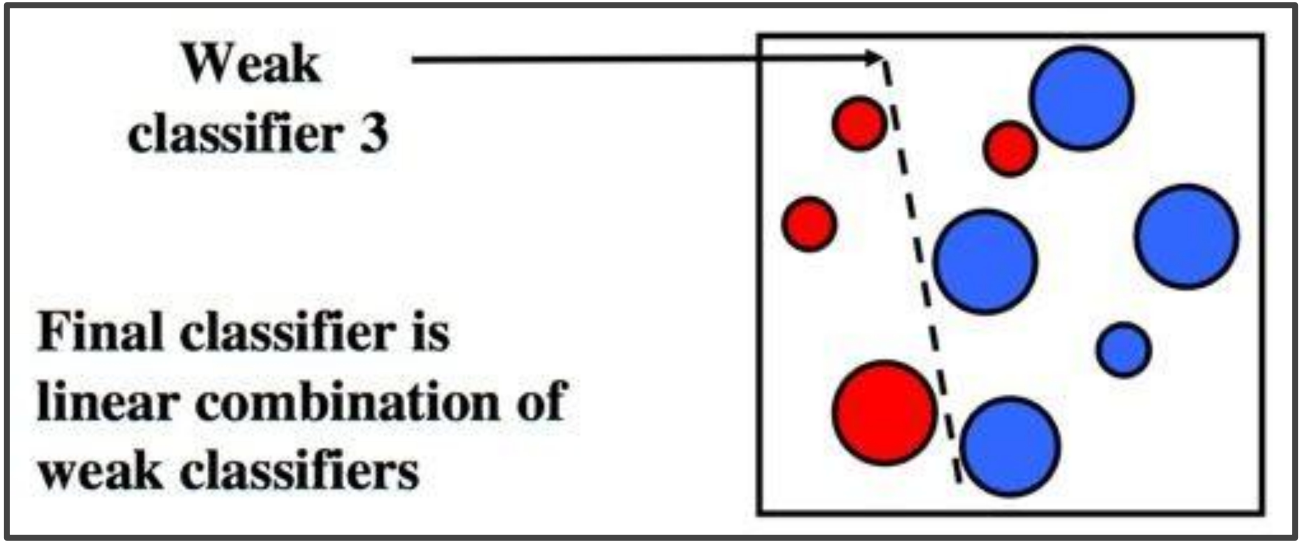
Slides largely based on Prof. Sandra Avila's Machine Learning Course

AdaBoost



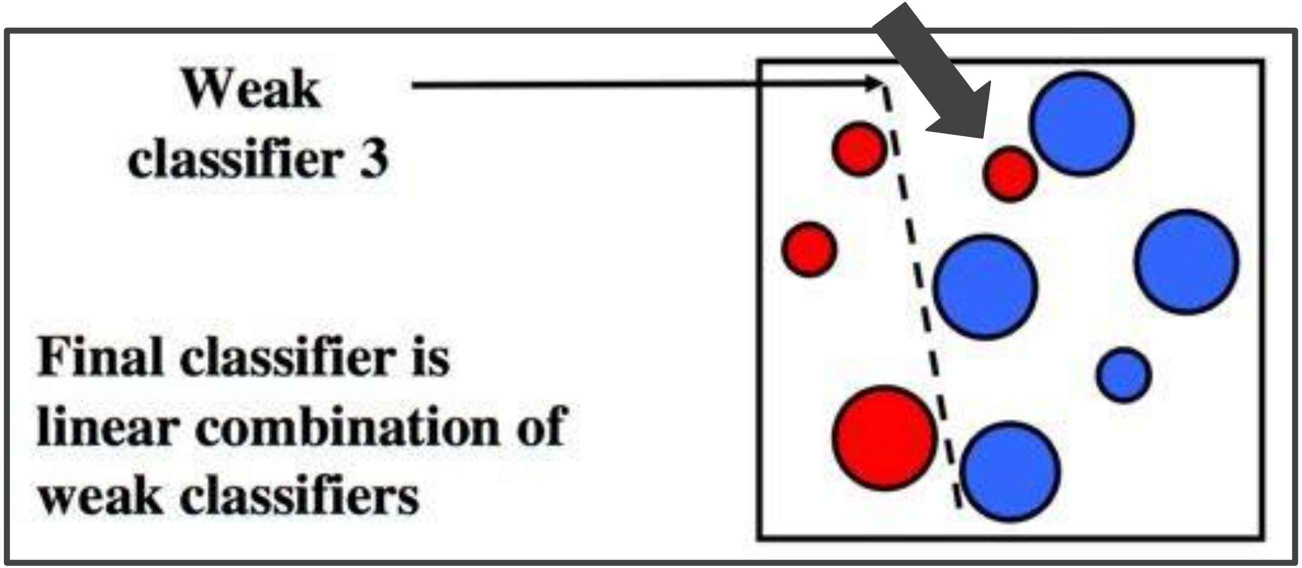
Slides largely based on Prof. Sandra Avila's Machine Learning Course

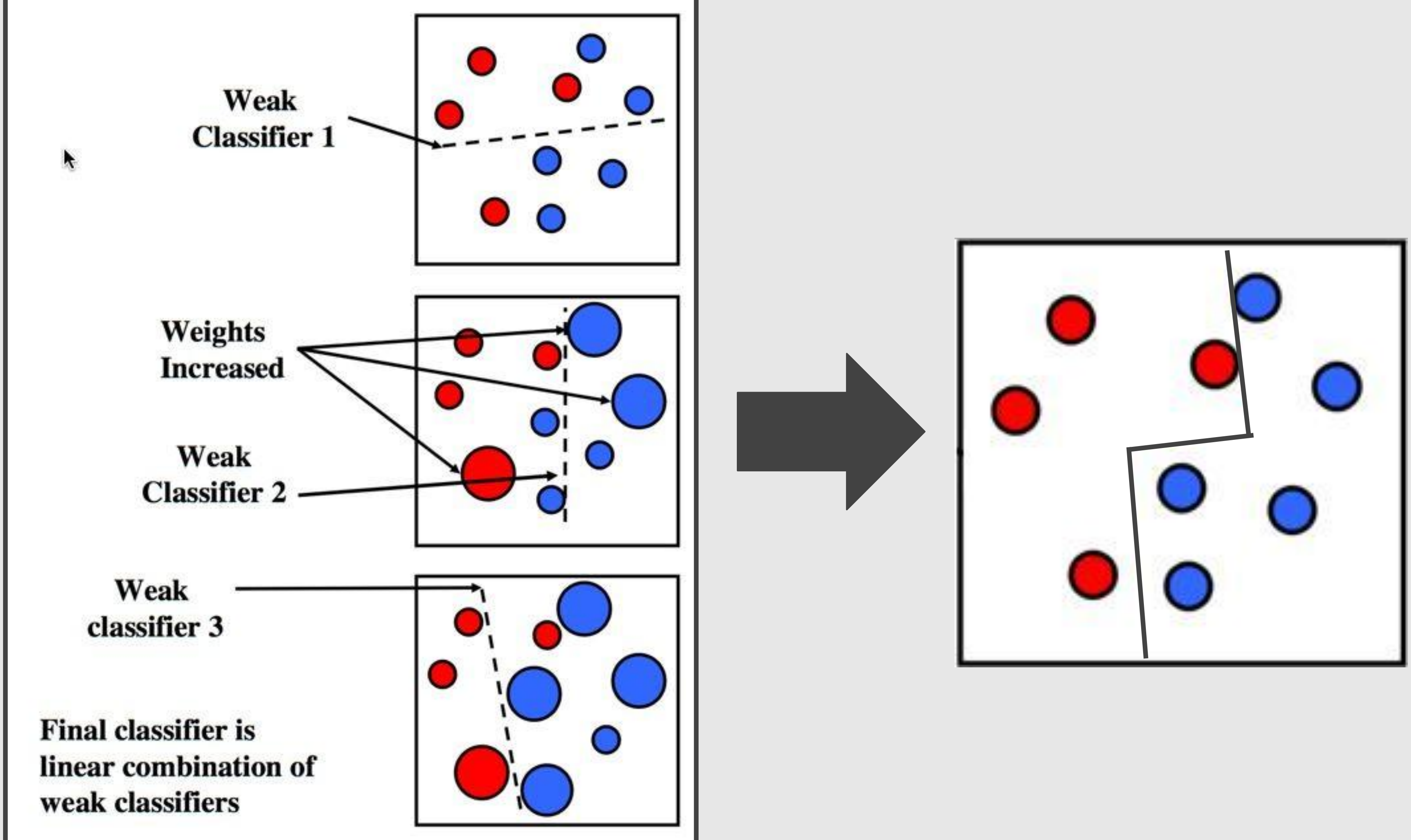
AdaBoost



Weak
classifier 3
Final classifier is
linear combination of
weak classifiers

AdaBoost


Weak
classifier 3
Final classifier is
linear combination of
weak classifiers


Weak
Classifier 1
Weights
Increased
Weak
Classifier 2
Weak
classifier 3
Final classifier is
linear combination of
weak classifiers

AdaBoost

1. Assign every observation, x_i, an initial weight value, $w_i = \frac{1}{n}$, where n is the total number of observations.
2. Train a "weak" model. (most often a decision tree)
3. For each observation:
    - 3.1. If predicted incorrectly, w_i is increased
    - 3.2. If predicted correctly, w_i is decreased
4. Train a new weak model where observations with greater weights are given more priority.
5. Repeat steps 3 and 4 until observations perfectly predicted or a preset number of trees are trained.

Chris Albon

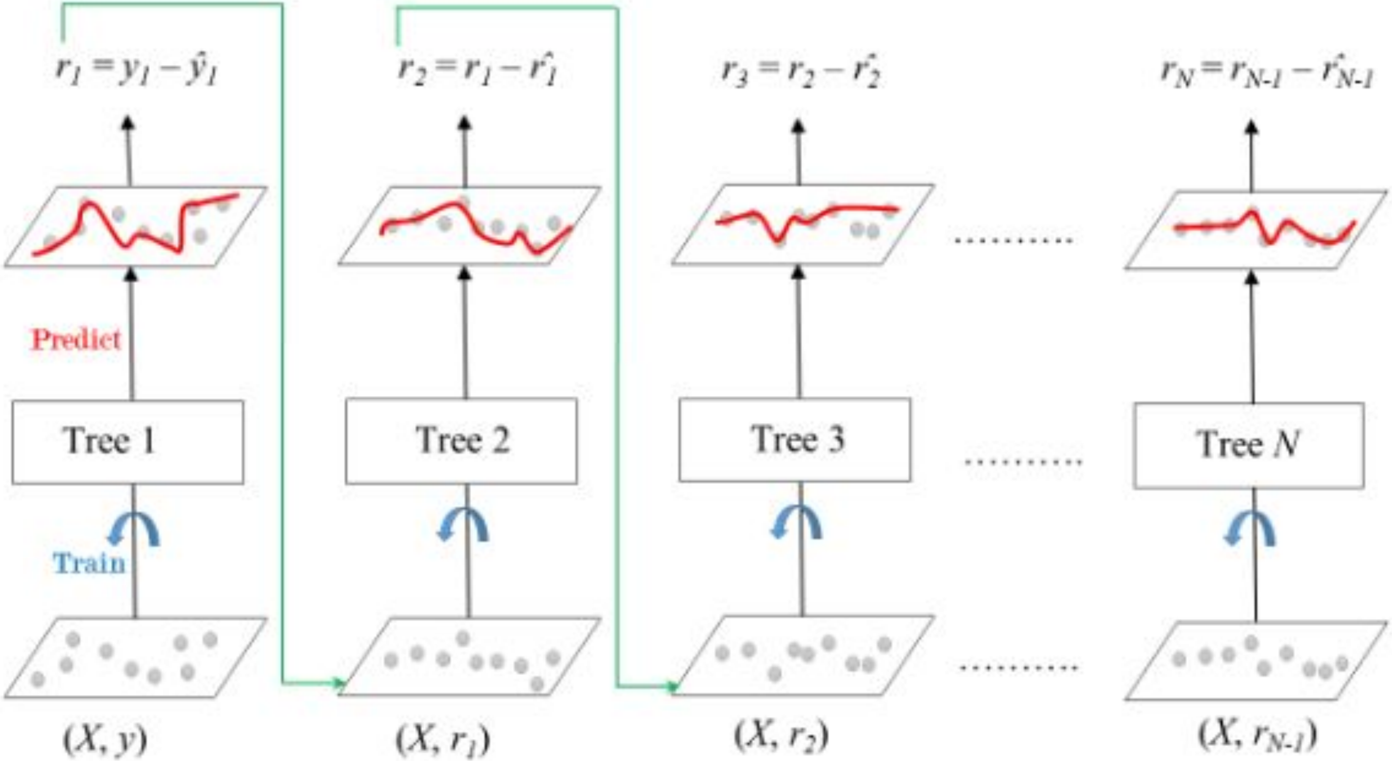
Gradient Boosting [Breiman, 1997]

- Instead of tweaking the instance weights at every iteration like AdaBoost does, this method fit the new predictor to the **residual errors** made by the previous predictor.

Gradient Boosting [Breiman, 1997]

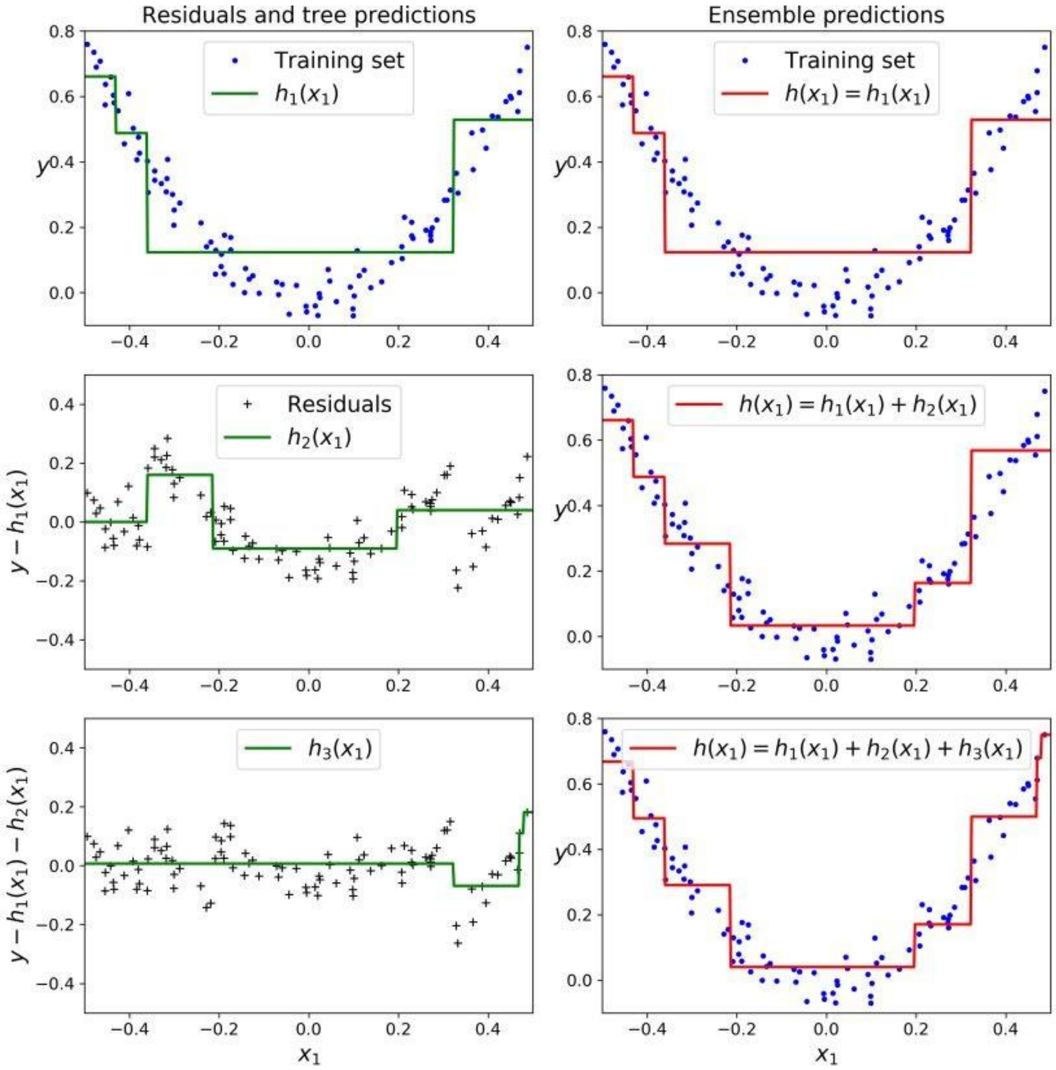
- Instead of tweaking the instance weights at every iteration like AdaBoost does, this method fit the new predictor to the **residual errors** made by the previous predictor.

- Instead of training on a newly sample distribution, the weak learner **trains on the remaining errors**.



y(pred) = y1 + (eta * r1) + (eta * r2) + ... + (eta * rN)

Slides largely based on Prof. Sandra Avila's Machine Learning Course



Residuals and tree predictions
Ensemble predictions
Training set
$h_1(x_1)$
$h(x_1) = h_1(x_1)$
Residuals
$h_2(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1)$
$h_3(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1) + h_3(x_1)$
y
$y - h_1(x_1)$
$y - h_1(x_1) - h_2(x_1)$
x_1

Gradient Boosting [Breiman, 1997]

1. Fit a simple linear regressor or decision tree on data
 [call x as input and y as output]
2. Calculate error residuals. Actual target value, minus predicted target value
 [e1 = y - y_predicted1]
3. Fit a new model on error residuals as target variable with same input variables
 [call it e1_predicted]
4. Add the predicted residuals to the previous predictions
 [y_predicted2 = y_predicted1 + e1_predicted]
5. Fit another model on residuals that is still left, i.e. **[e2 = y - y_predicted2]** and repeat steps 2 to 5 until it starts overfitting or the sum of residuals become constant.

Gradient Boosting [Breiman, 1997]

- XGboost [Chen and Guestrin, 2016]:

  Extreme Gradient Boosting

  https://github.com/tqchen/xgboost

  It aims at being extremely fast, scalable and portable.

Ensemble Methods

- Bagging (and Pasting)
- Boosting
- **Stacking**

Stacking [Wolpert, 1992]

- Stacking (short for Stacked Generalization)

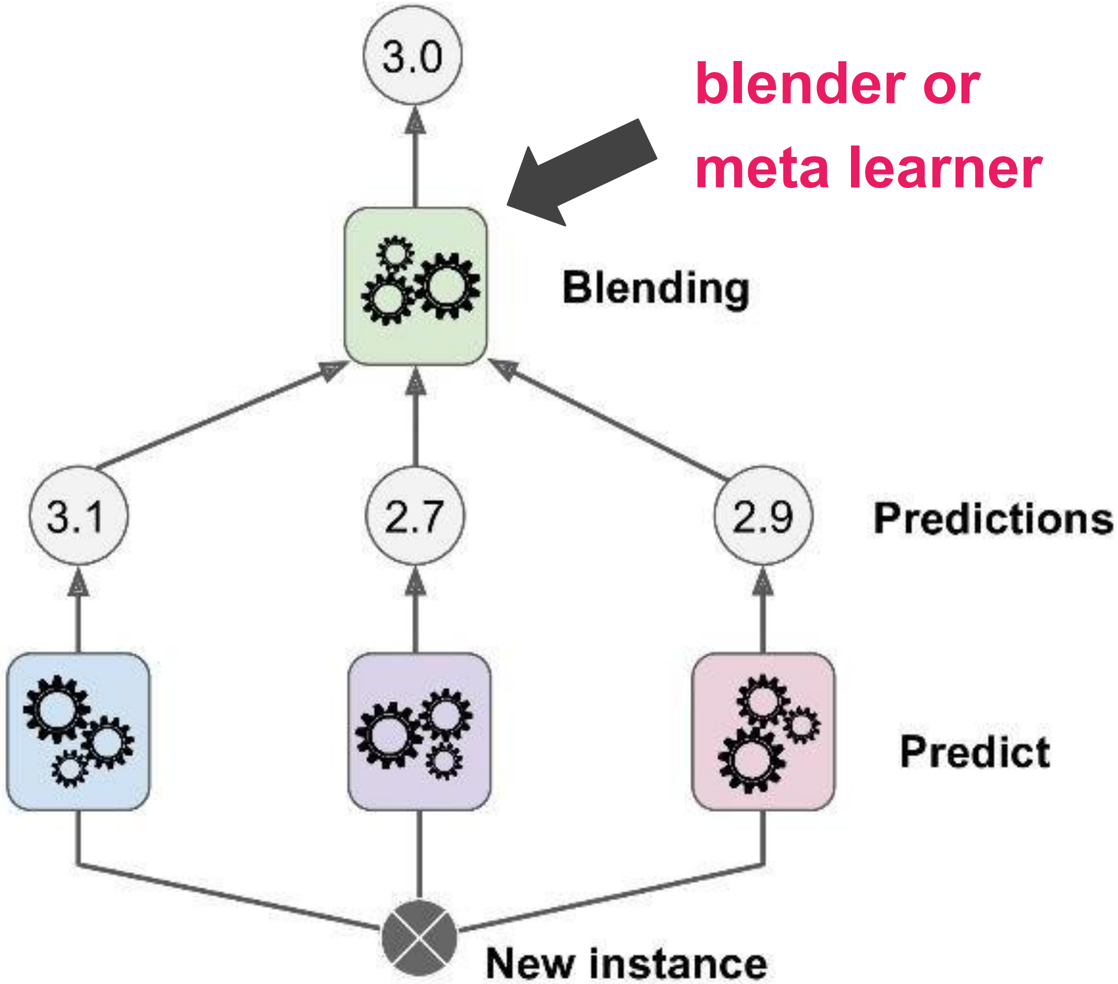
“Stacked Generalization”, D. Wolpert (1992): http://goo.gl/9I2NBw

Stacking [Wolpert, 1992]

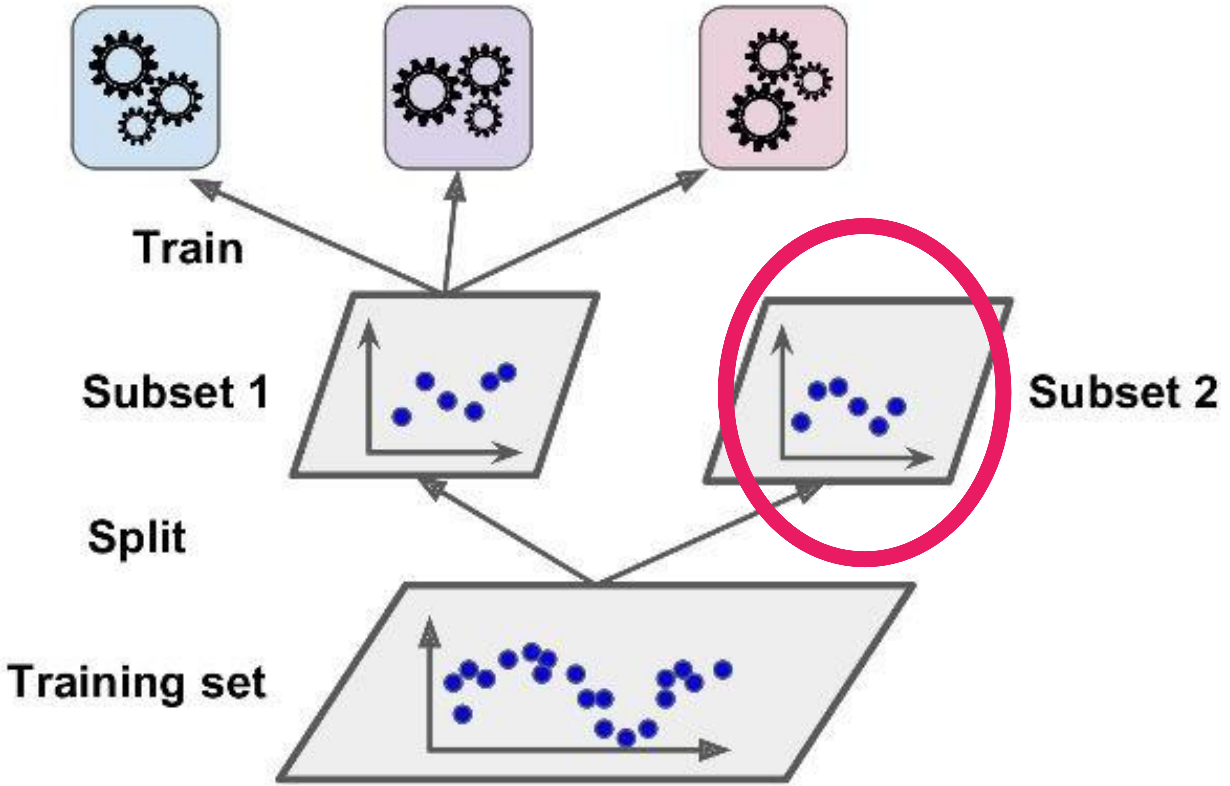
- Stacking (short for Stacked Generalization)
- Instead of using trivial functions (such as hard voting) to aggregate the predictions of all predictors in an ensemble, we **train a model to perform this aggregation**.

“Stacked Generalization”, D. Wolpert (1992): http://goo.gl/9I2NBw

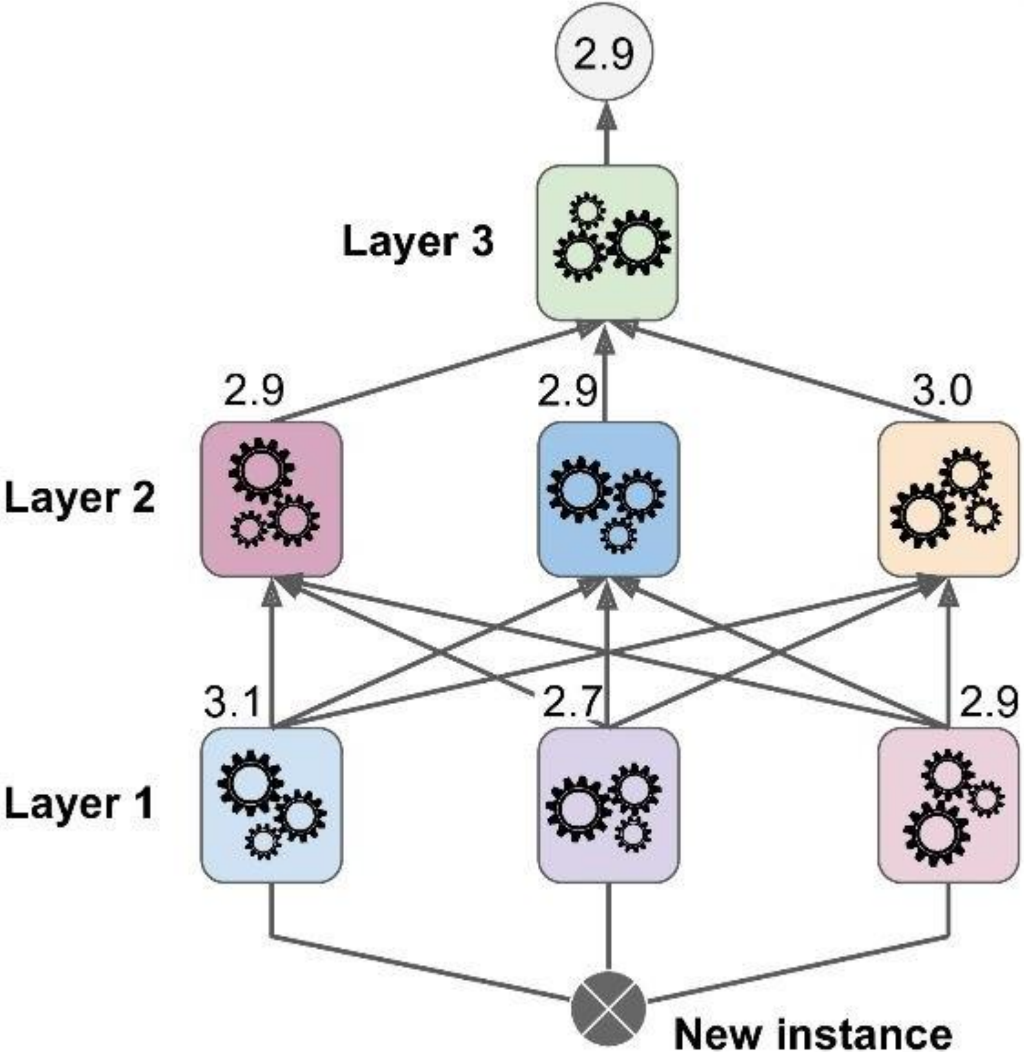
Stacking


3.0
blender or
meta learner
Blending
3.1
2.7
2.9
Predictions
Predict
New instance

To train the blender, a common approach is to use a **hold-out set**.



Multi-layer Stacking Ensemble



Stacking [Wolpert, 1992]

- Scikit-Learn does not support stacking directly. =(

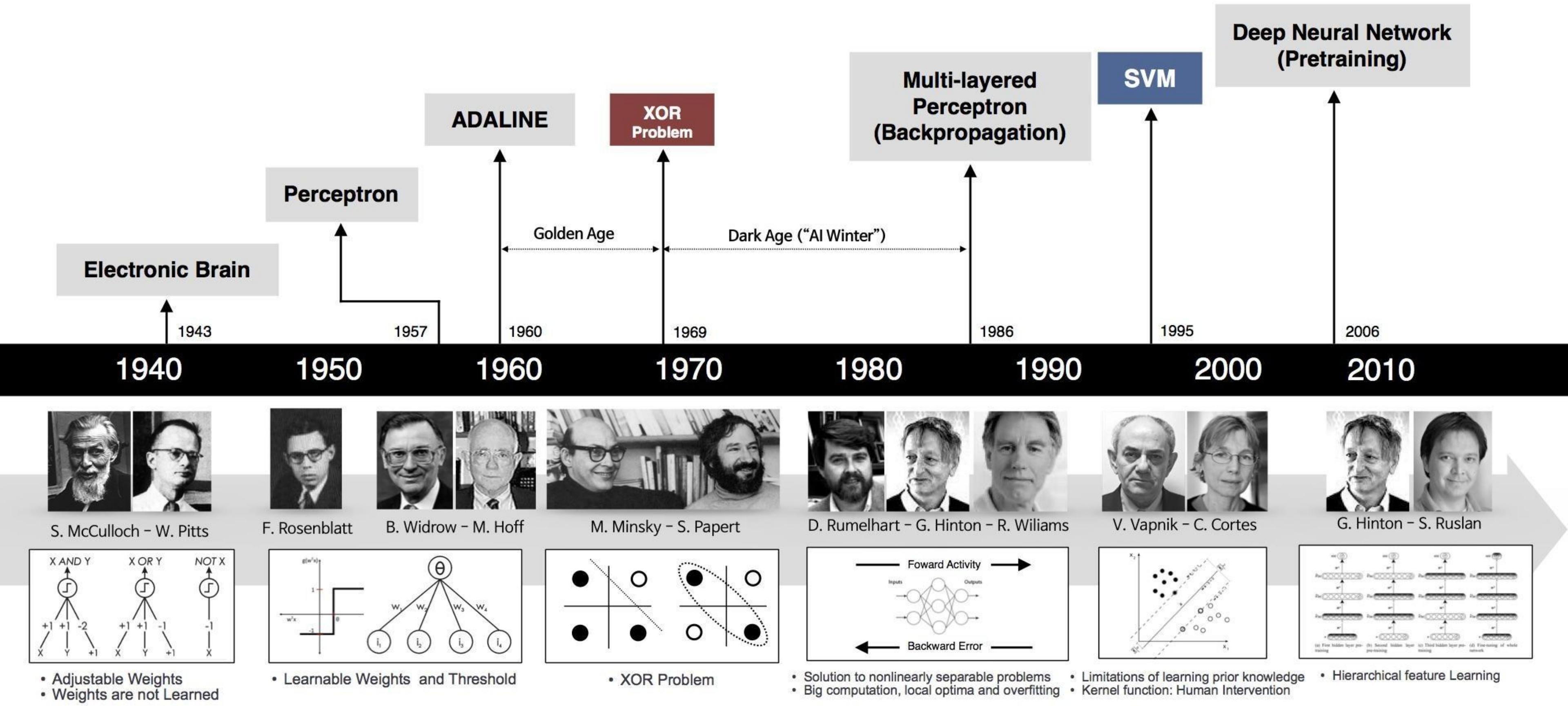
“Stacked Generalization”, D. Wolpert (1992): http://goo.gl/9I2NBw

References

Machine Learning Books

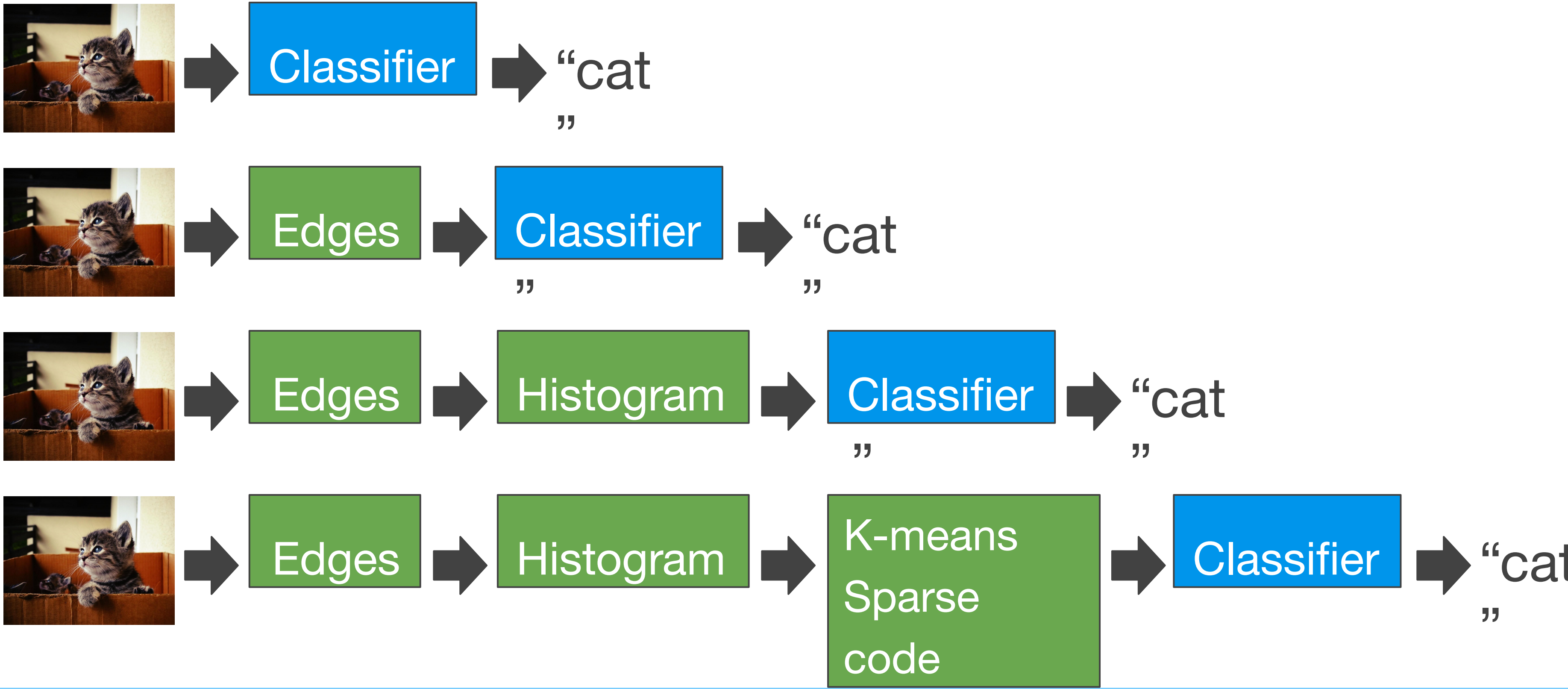
- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Chap. 6 & 7
- Pattern Recognition and Machine Learning, Chap. 14
- Pattern Classification, Chap 8 & 9 (Sec. 9.5)
- "Scikit Learn Ensemble Learning, Bootstrap Aggregating (Bagging) and Boosting" https://youtu.be/X3Wbfb4M33w

Support Vector Machines (SVMs)



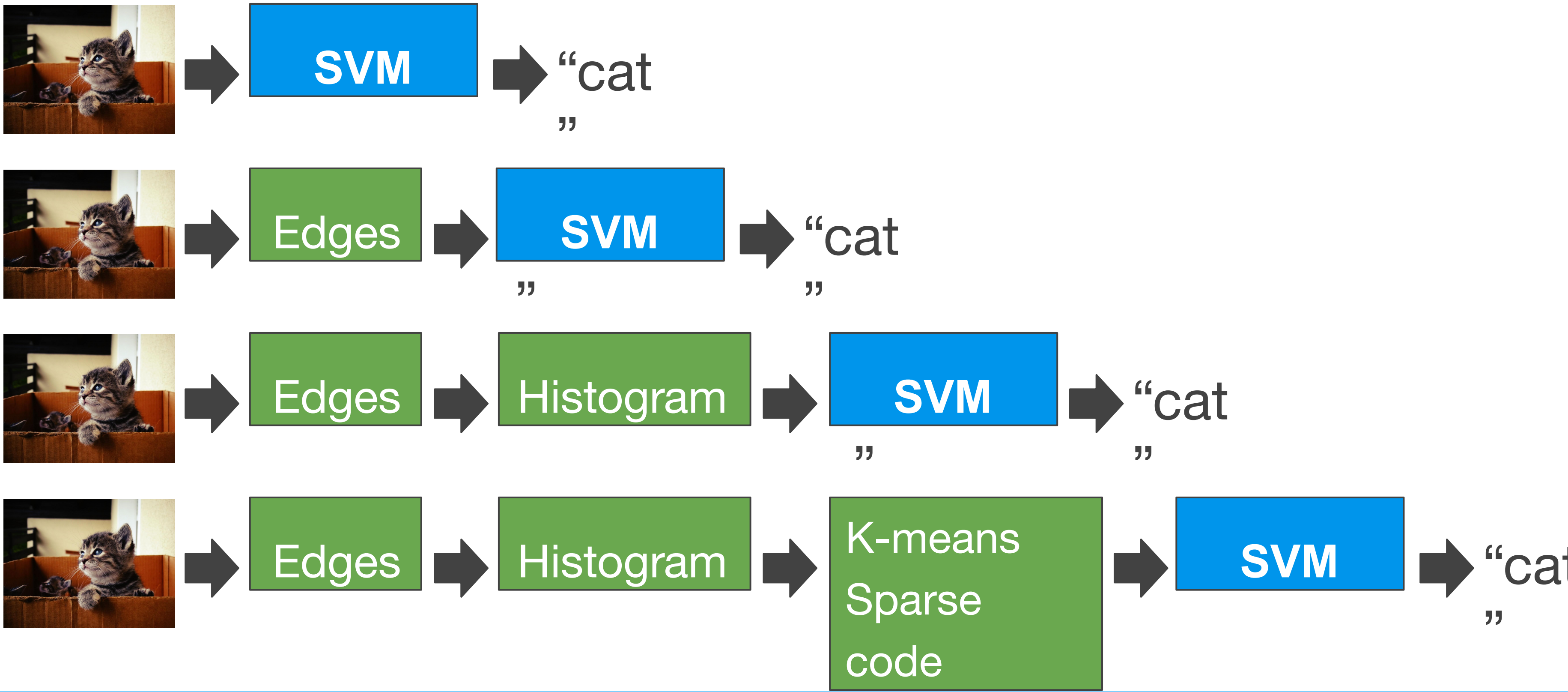
Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch – W. Pitts
F. Rosenblatt
B. Widrow – M. Hoff
M. Minsky – S. Papert
D. Rumelhart – G. Hinton – R. Wiliams
V. Vapnik – C. Cortes
G. Hinton – S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

Traditional Recognition



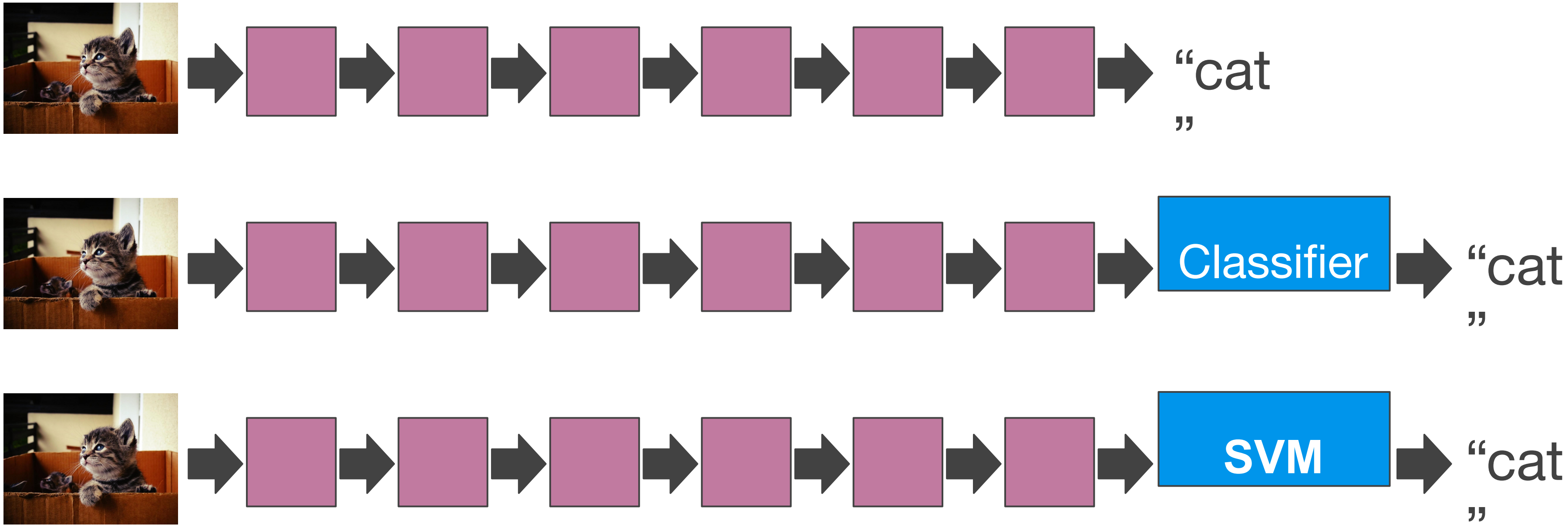
Slides largely based on Prof. Sandra Avila's Machine Learning Course

Traditional Recognition



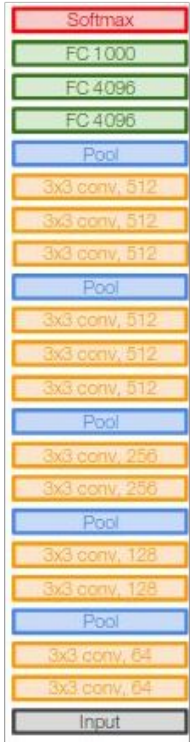
Slides largely based on Prof. Sandra Avila's Machine Learning Course

Deep Learning



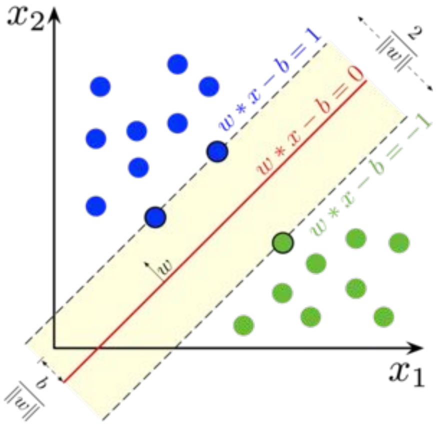
Slides largely based on Prof. Sandra Avila’s Machine Learning Course

Transfer Learning with CNNs



[0.01 0.8 1 0.5 … 0.3 0.07 0 0.4 0.6 0 0]
4096-d

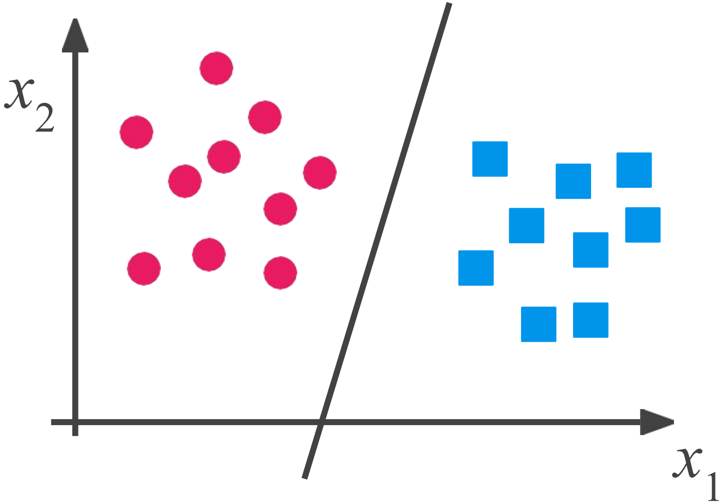
Train a classifier (e.g., SVM)



VGG as **Feature Extractor**

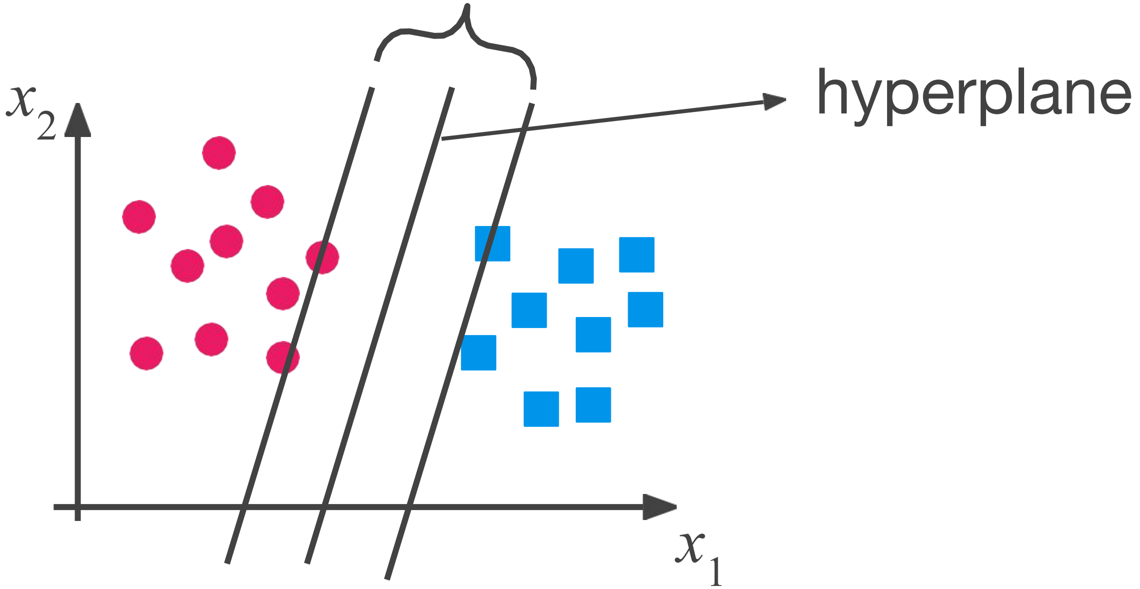
Support Vector Machine (SVM)
[Vapnik and Chervonenkis, 1964; Vapnik, 1982; Vapnik, 1995]

Idea of separating data with **a large "gap"**.



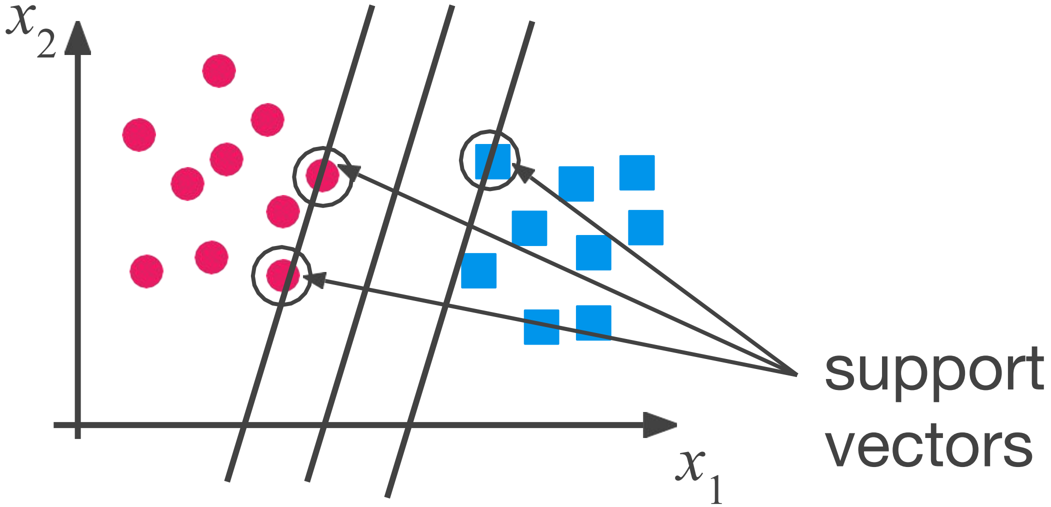
Support Vector Machine (SVM)
[Vapnik and Chervonenkis, 1964; Vapnik, 1982; Vapnik, 1995]

Idea of separating data with **a large "gap"**.



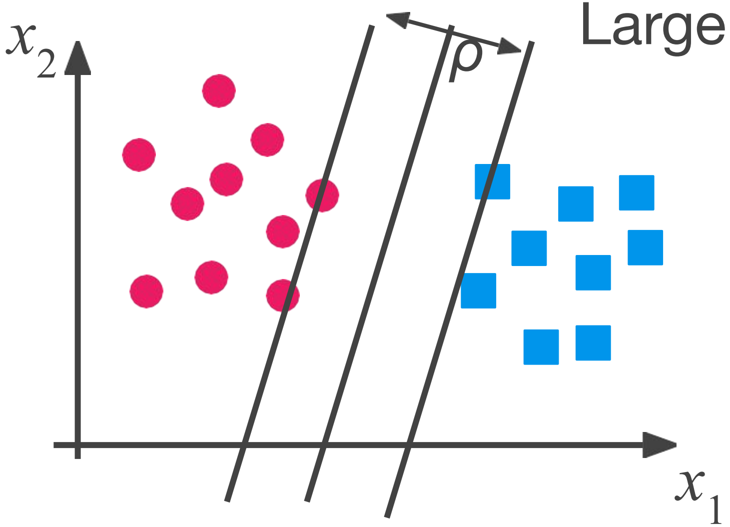
Support Vector Machine (SVM)
[Vapnik and Chervonenkis, 1964; Vapnik, 1982; Vapnik, 1995]

Examples closest to the hyperplane are support vectors.



Support Vector Machine (SVM)
[Vapnik and Chervonenkis, 1964; Vapnik, 1982; Vapnik, 1995]

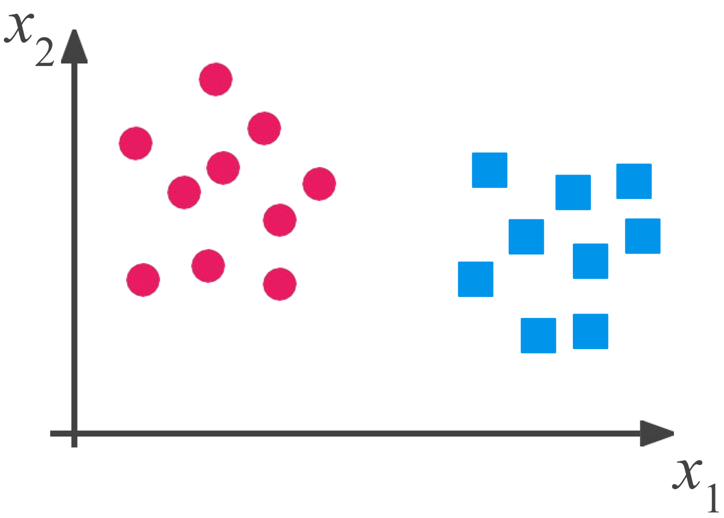
Margin ρ of the separator is the distance between support vectors.



How does SVM work?

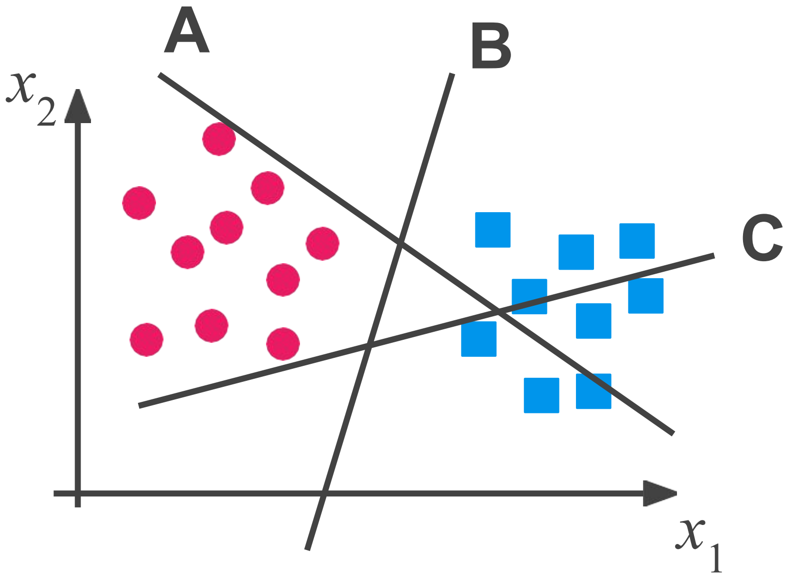
How can we identify the right hyperplane?

Scenario 1



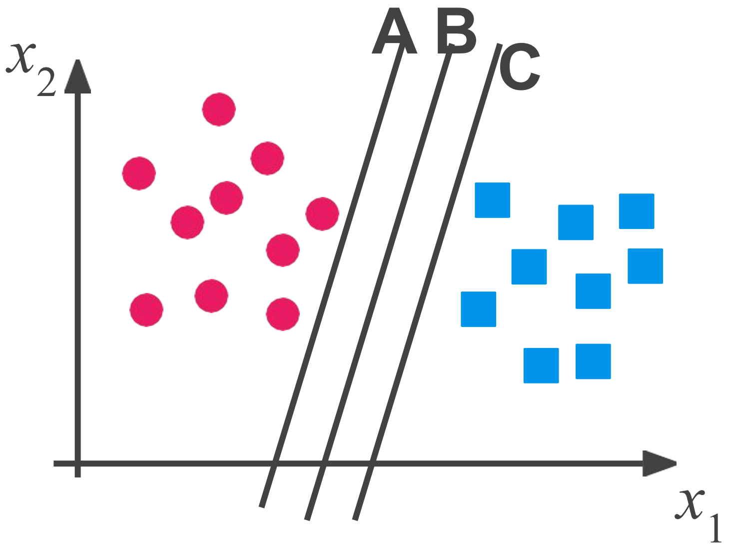
How can we identify the right hyperplane?

Scenario 1



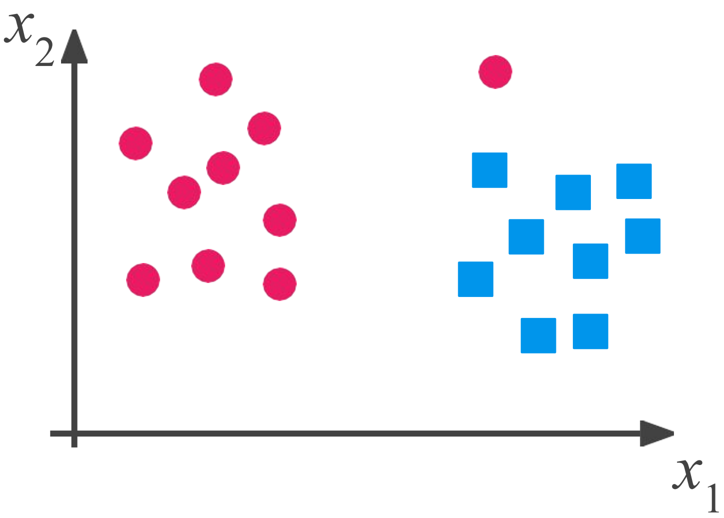
How can we identify the right hyperplane?

Scenario 2



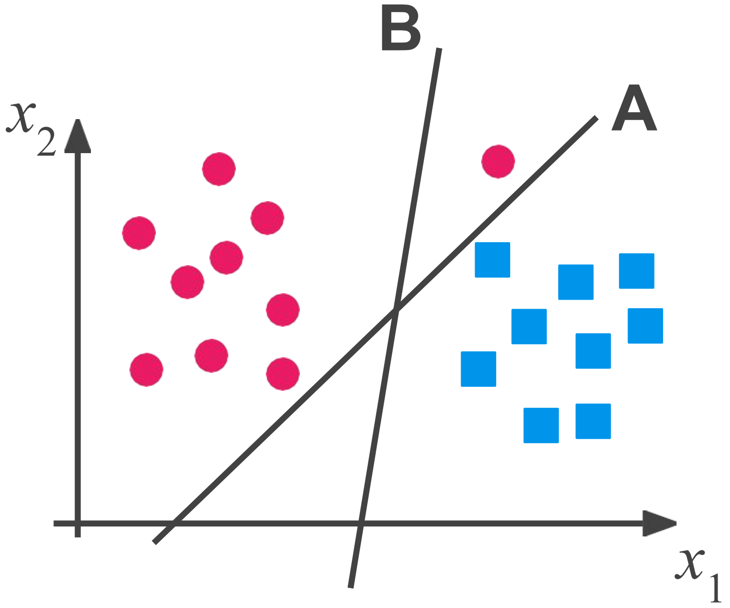
How can we identify the right hyperplane?

Scenario 3



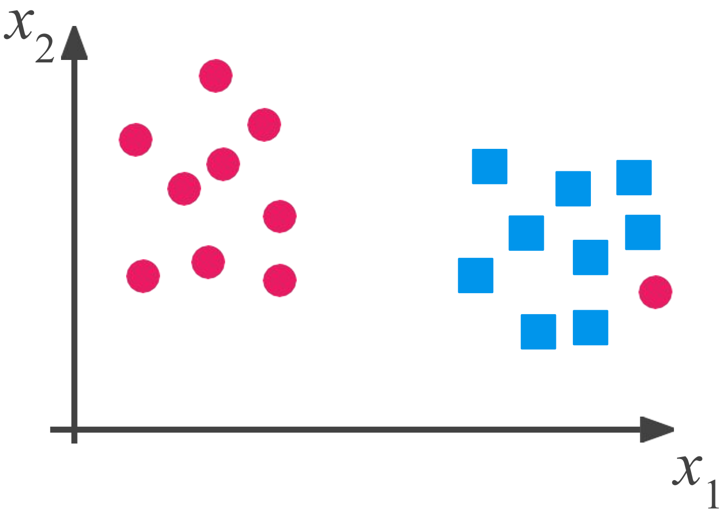
How can we identify the right hyperplane?

Scenario 3



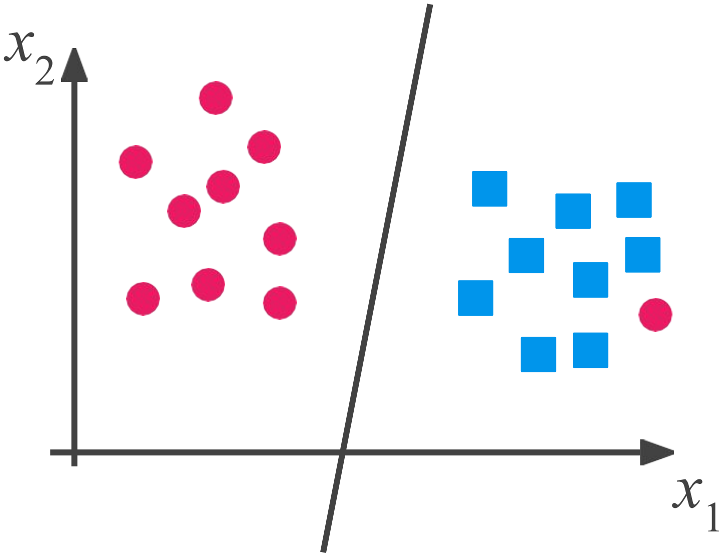
How can we identify the right hyperplane?

Scenario 4



How can we identify the right hyperplane?

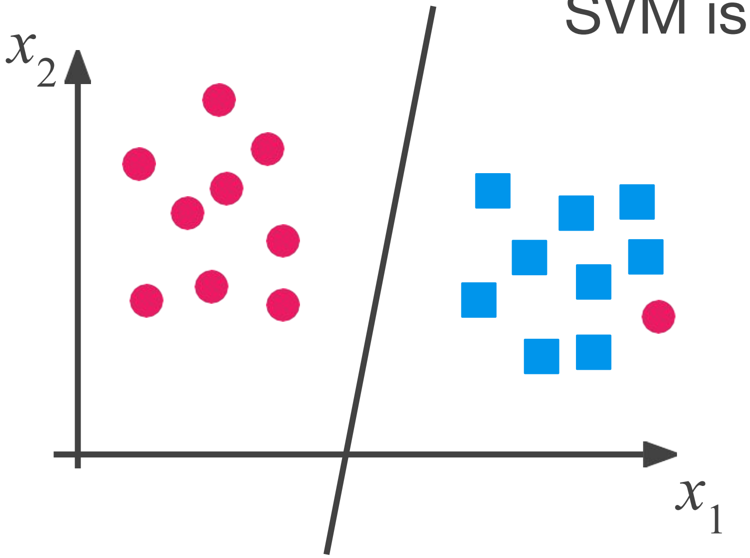
Scenario 4



How can we identify the right hyperplane?

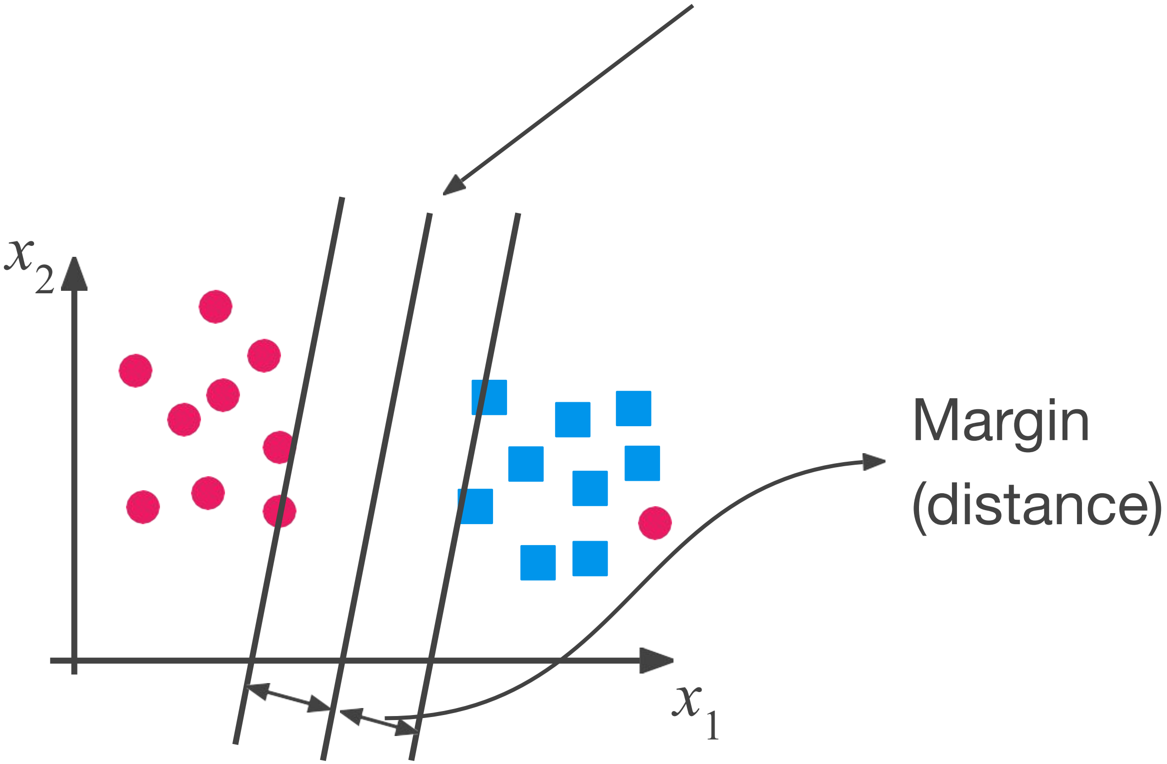
Scenario 4

SVM is robust to outliers



How can we identify the right hyperplane?

Scenario 4



SVM: Notation

We will be considering a **linear classifier for a binary classification** problem with labels y and features x.

SVM: Notation

We will be considering a **linear classifier for a binary classification** problem with labels y and features x.

- Class labels: $y \in \{-1,1\}$ (instead of $\{0,1\}$)
- Parameters: w, b (instead of vector θ)

SVM: Notation

We will be considering a **linear classifier for a binary classification** problem with labels y and features x.

- Class labels: $y \in \{-1,1\}$ (instead of $\{0,1\}$)
- Parameters: w, b (instead of vector θ)
- Classifier: $h_{w,b}(x) = g(w^T x + b)$
  - $g(z) = 1$ if $z \geq 0$, and $g(z) = -1$ otherwise

SVM: The Optimal Hyperplane

Given a training example $(x^{(i)}, y^{(i)})$, we define the margin of (w, b) with respect to the training example:

$$y^{(i)}(w^T x + b) \geq 1, i = \{1, ..., m\}.$$

SVM: The Optimal Hyperplane

Let $P(x^{(1)}, y^{(1)})$ be a point and l be a line defined by $ax + by + c = 0$. The distance d from P to l is defined by:

$$d(l,P) = \frac{|ax^{(1)} + by^{(1)} + c|}{\sqrt{a^2 + b^2}}$$

SVM: The Optimal Hyperplane

Let $P(x^{(1)}, y^{(1)})$ be a point and l be a line defined by $ax + by + c = 0$. The distance d from P to l is defined by:

$$d(l,P) = \frac{|ax^{(1)} + by^{(1)} + c|}{\sqrt{a^2 + b^2}}$$

$$d(w,b,x) = \frac{|w^T x + b|}{||w||}$$

SVM: The Optimal Hyperplane

$$d(w,b,x) = \frac{|w^T x + b|}{||w||}$$

$$\min_{w,b} \tfrac{1}{2}||w||^2$$

$$\text{s.t. } y^{(i)}(w^T x + b) \geq 1,\ i = \{1, ..., m\}$$

SVM: The Optimal Hyperplane

$$d(w,b,x) = \frac{|w^T x + b|}{||w||}$$

http://cs229.stanford.edu/notes2019fall/cs229-notes3.pdf

$$\min_{w,b} \ \tfrac{1}{2}||w||^2$$

$$\text{s.t. } y^{(i)}(w^T x + b) \geq 1, \ i = \{1, \ldots, m\}$$

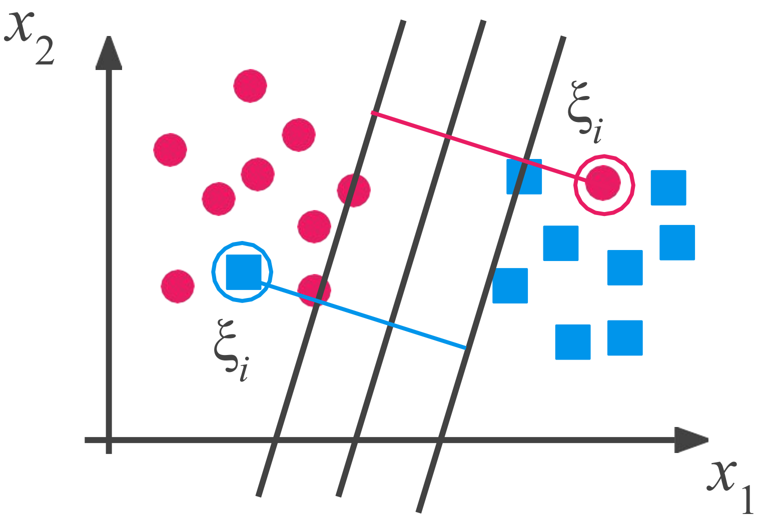
Need to optimize a quadratic function subject to linear

Soft Margin Classification

What if the training set is not linearly separable?

Soft Margin Classification

Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called **soft**.



Soft Margin Classification

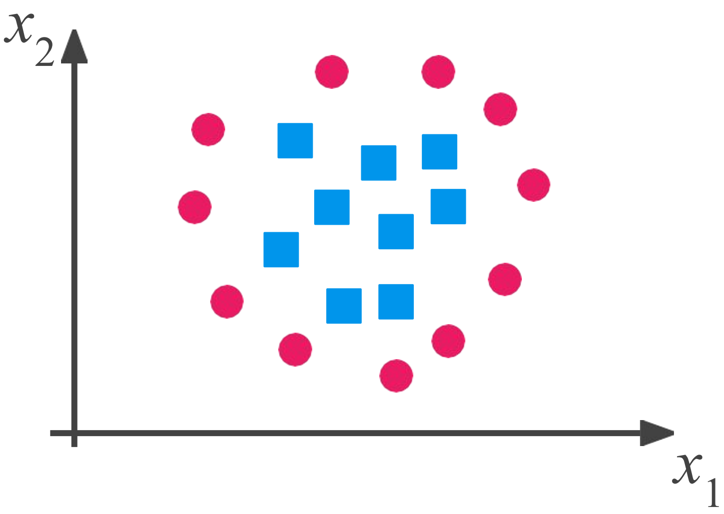
Modified formulation incorporates slack variables:

$$\boxed{\begin{array}{c} \min_{w,b,\xi} \ \tfrac{1}{2}||w||^2 + C\,\Sigma\xi_i \\ \text{s.t. } y_i(w^T x + b) \geq 1 - \xi_i,\ \xi_i \geq 0,\ i = \{1, \ldots, m\} \end{array}}$$

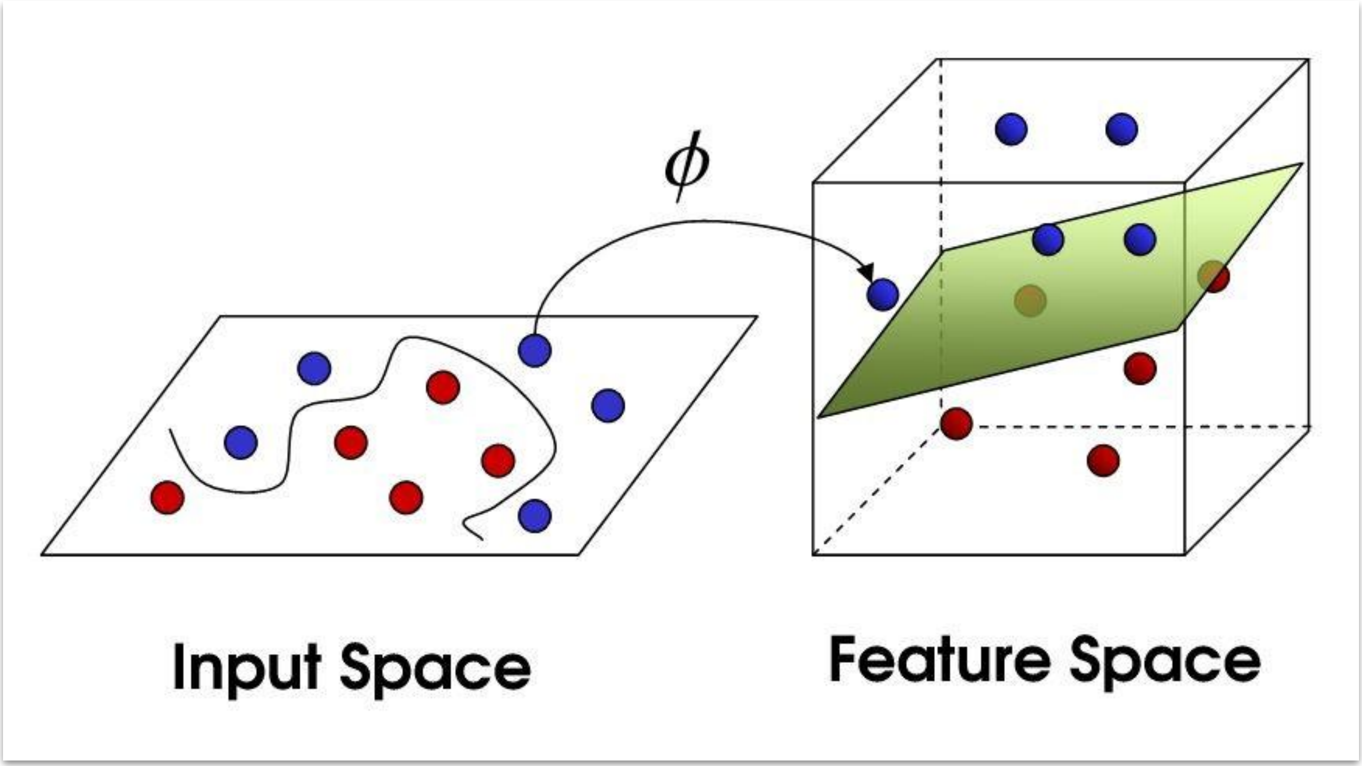
Parameter C can be viewed as a way to control overfitting:
it “trades off” the relative importance of maximizing the margin and fitting the training data.

How can we identify the right hyperplane?

Scenario 5

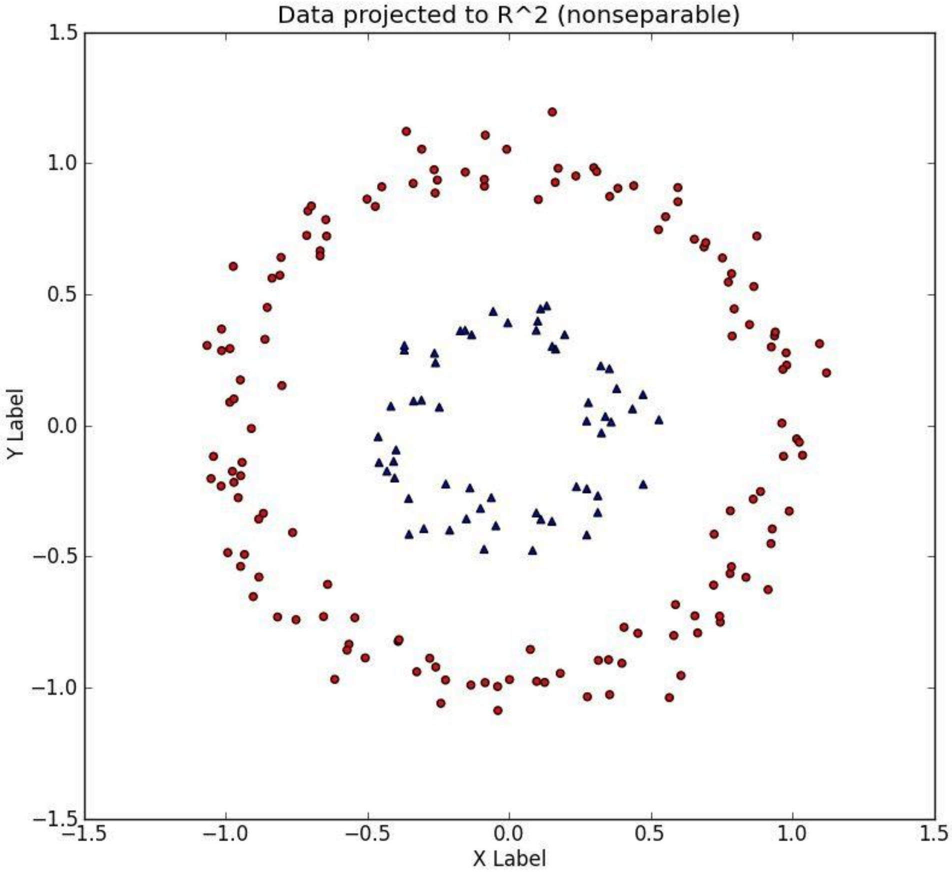


Kernel Trick

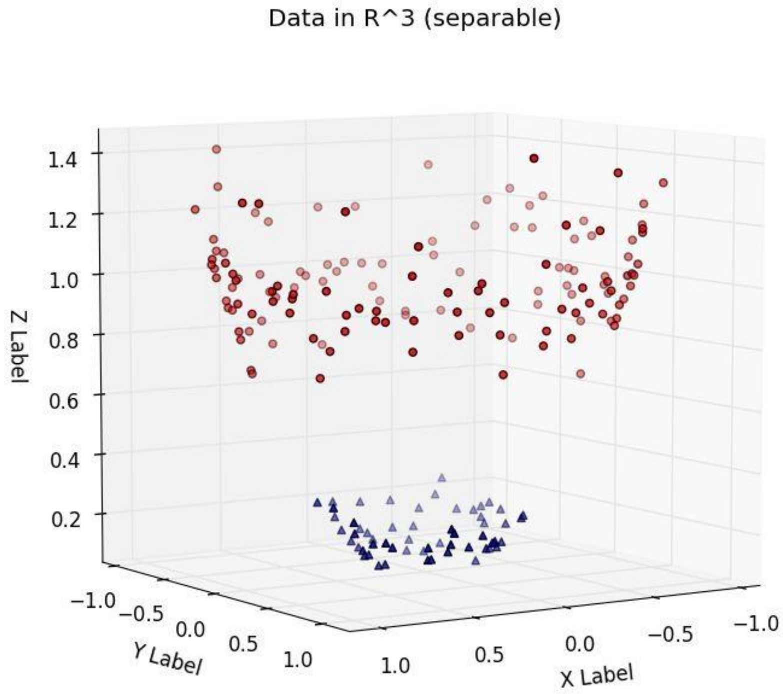


Kernel Trick

- Linear SVM: $x_i \cdot x_j$
- Nonlinear SVM: $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$, feature mapping ϕ
- Kernel matrix $K_{ij} = K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j) = \phi(x_j) \cdot \phi(x_i) = K_{ji}$
- Radial Basis Function (RBF) kernel
- Gaussian kernel
- Polynomial kernel
- Chi-square kernel, histogram intersection kernel, string kernel, ...



Data projected to R^2 (nonseparable)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Y Label
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
X Label



Data in R^3 (separable)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
Z Label
−1.0
−0.5
0.0
0.5
1.0
Y Label
1.0
0.5
0.0
−0.5
−1.0
X Label

Important Parameters

Important parameters having higher impact on model performance, “kernel”, “gamma” and “C”.

Important Parameters

Important parameters having higher impact on model performance, “kernel”, “gamma” and “C”.

C: Penalty parameter C of the error term. It also controls the trade off between smooth decision boundary and classifying the training points correctly.

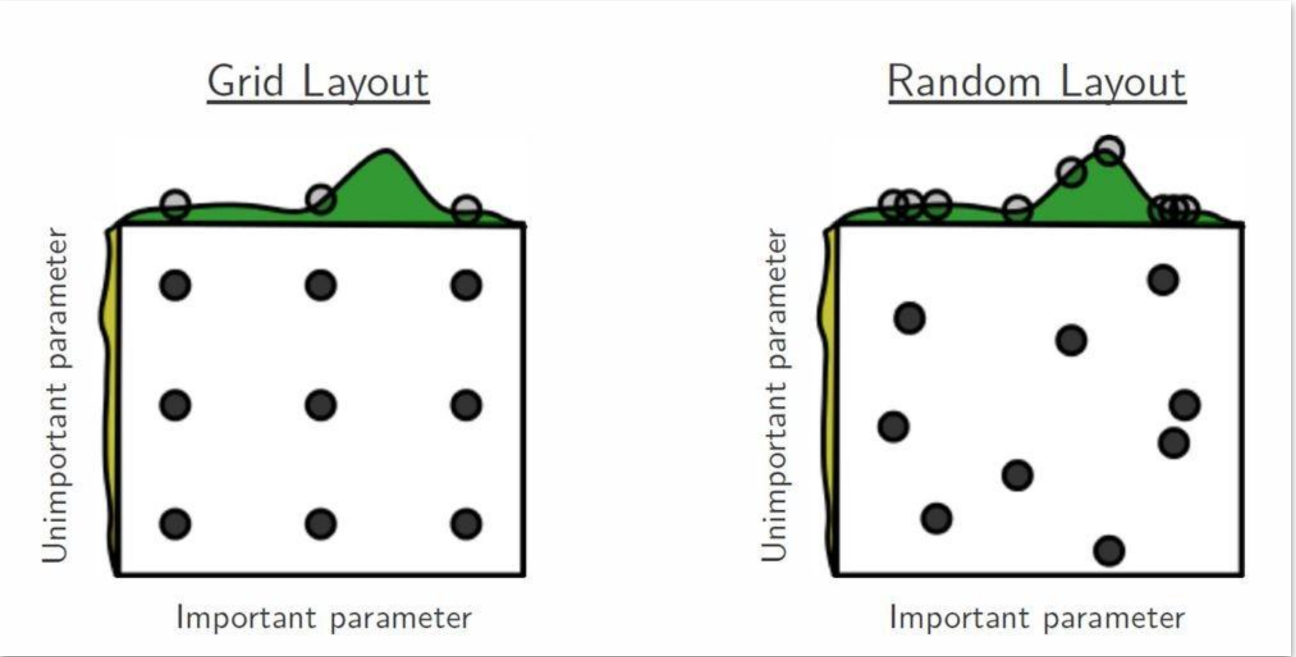
Important Parameters

Important parameters having higher impact on model performance, “kernel”, “gamma” and “C”.

C: Penalty parameter C of the error term. It also controls the trade off between smooth decision boundary and classifying the training points correctly.

The parameters can be tuned using grid-search.

Grid Search

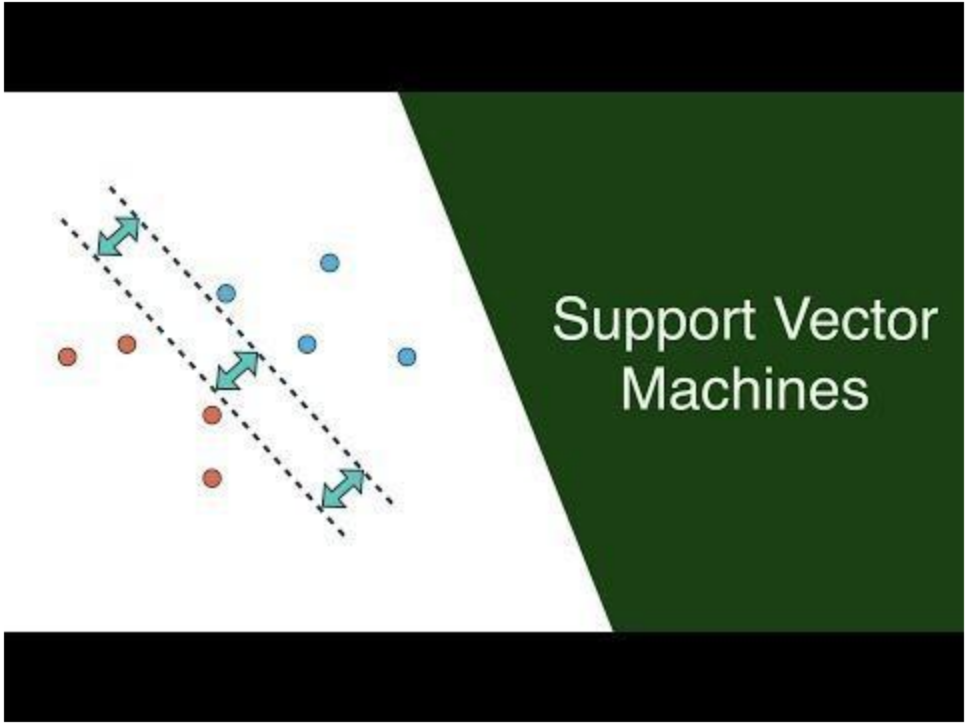


"Random Search for Hyper-Parameter Optimization": http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Libraries

- Scikit-learn: https://scikit-learn.org/stable/modules/svm.html
- LIBSVM: https://www.csie.ntu.edu.tw/~cjlin/libsvm
- LIBLINEAR: https://www.csie.ntu.edu.tw/~cjlin/liblinear
- PmSVM: https://sites.google.com/site/wujx2001/home/power-mean-svm

Support Vector Machines (SVMs): A friendly introduction



https://www.youtube.com/watch?v=Lpr__X8zuE8

Slides largely based on Prof. Sandra Avila's Machine Learning Course

References

Machine Learning Books

- Hands-On Machine Learning with Scikit-Learn and TensorFlow
- Pattern Recognition and Machine Learning, Chap. 6 & 7

Machine Learning Courses

- https://www.coursera.org/learn/machine-learning, Week 7
- http://cs229.stanford.edu/notes2019fall/cs229-notes3.pdf

Slides largely based on Prof. Sandra Avila's Machine Learning Course

Decision Trees

Decision Tree & Random Forest

- **Decision Trees** are versatile Machine Learning algorithms that can perform both classification and regression tasks, and even multi-output tasks.

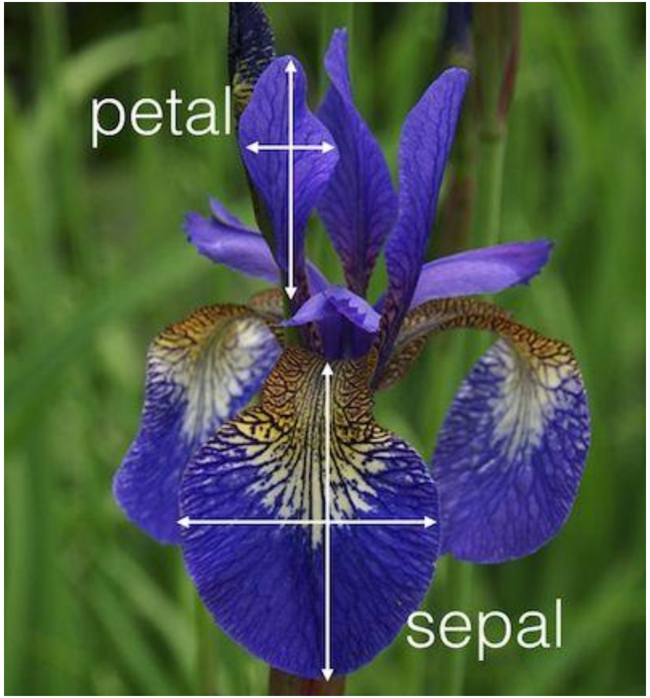
Decision Tree & Random Forest

- Decision Trees are versatile Machine Learning algorithms that can perform both classification and regression tasks, and even multi-output tasks.

- **Random Forest is an ensemble of Decision Trees**, generally trained using the Bagging method (or sometimes Pasting).

Slides largely based on Prof. Sandra Avila's Machine Learning Course

Decision Tree: Iris Dataset



http://sebastianraschka.com/Articles/2014_python_lda.html

150 iris flowers from three different species.

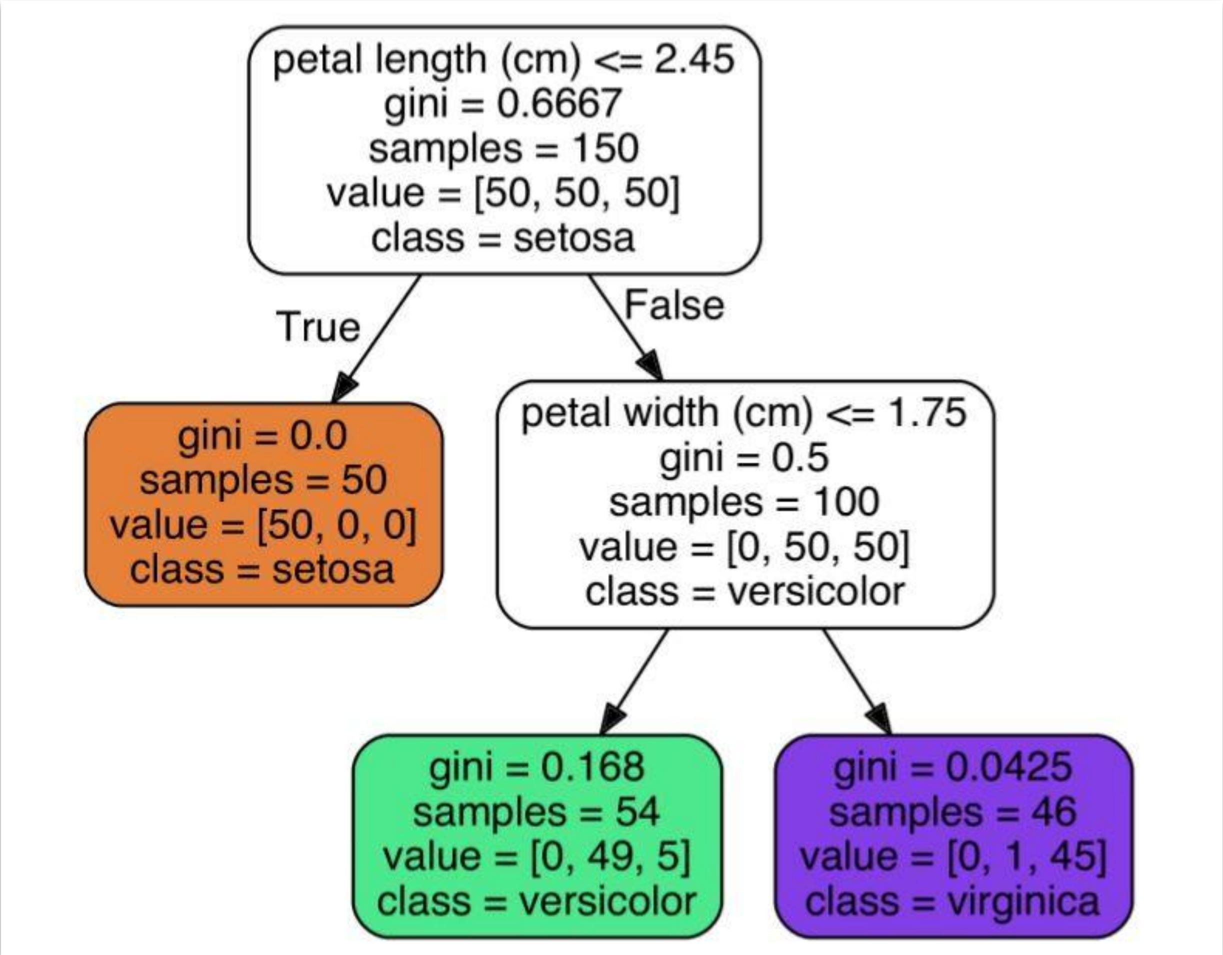
The three classes in the Iris dataset:

1. Iris-setosa (n=50)
2. Iris-versicolor (n=50)
3. Iris-virginica (n=50)

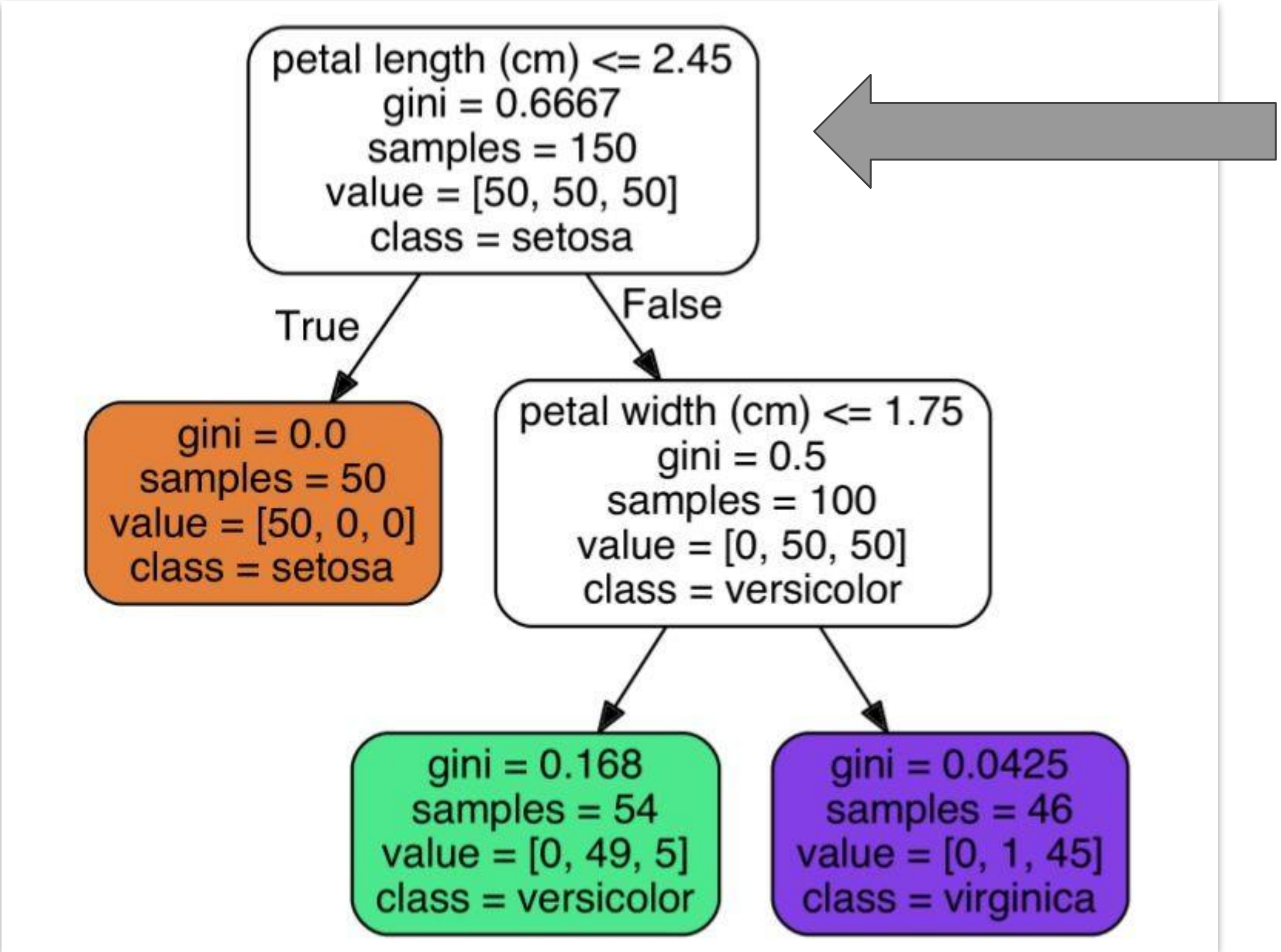
The four features of the Iris dataset:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

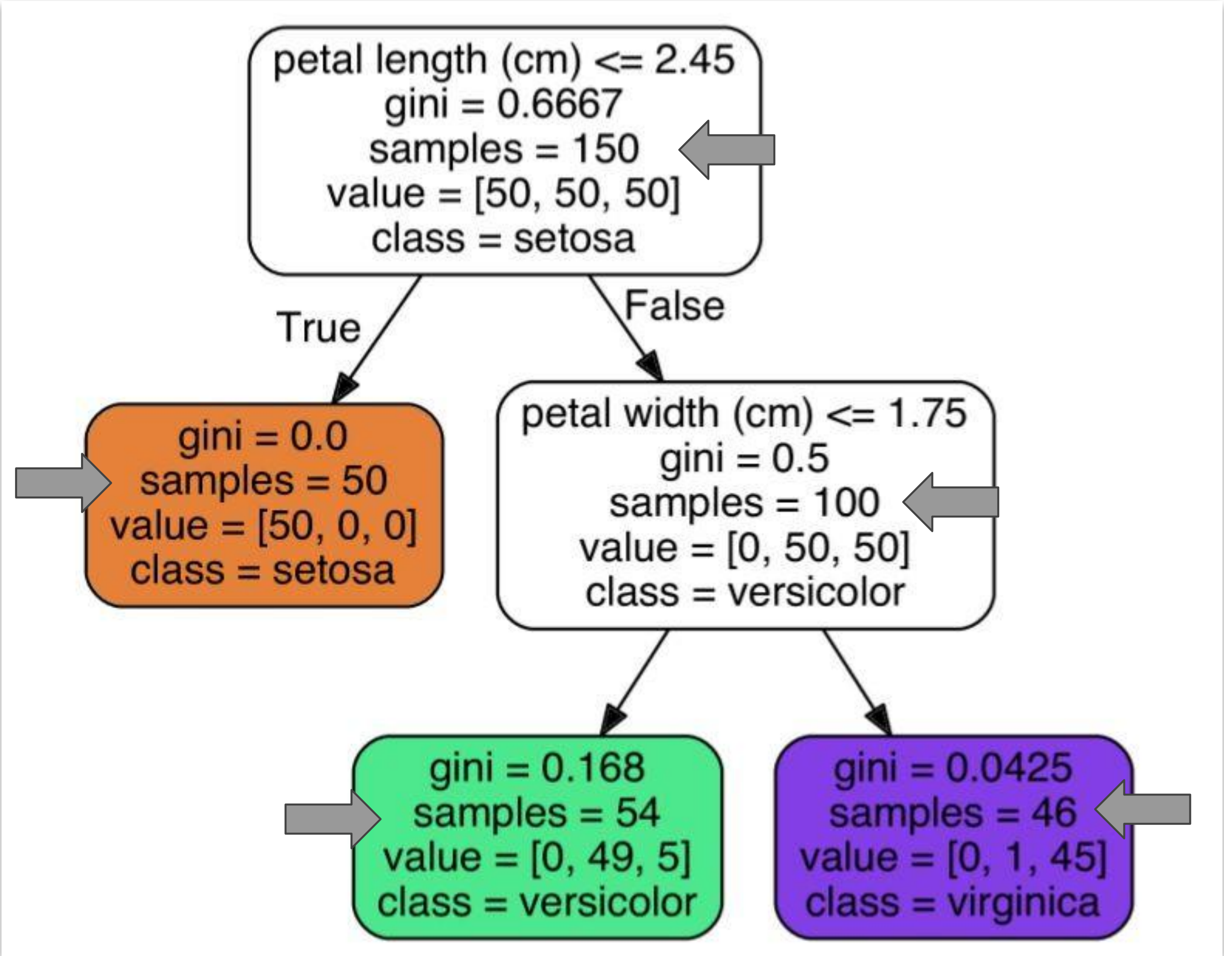
Slides largely based on Prof. Sandra Avila's Machine Learning Course


petal length (cm) <= 2.45
gini = 0.6667
samples = 150
value = [50, 50, 50]
class = setosa
True
False
gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa
petal width (cm) <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor
gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor
gini = 0.0425
samples = 46
value = [0, 1, 45]
class = virginica

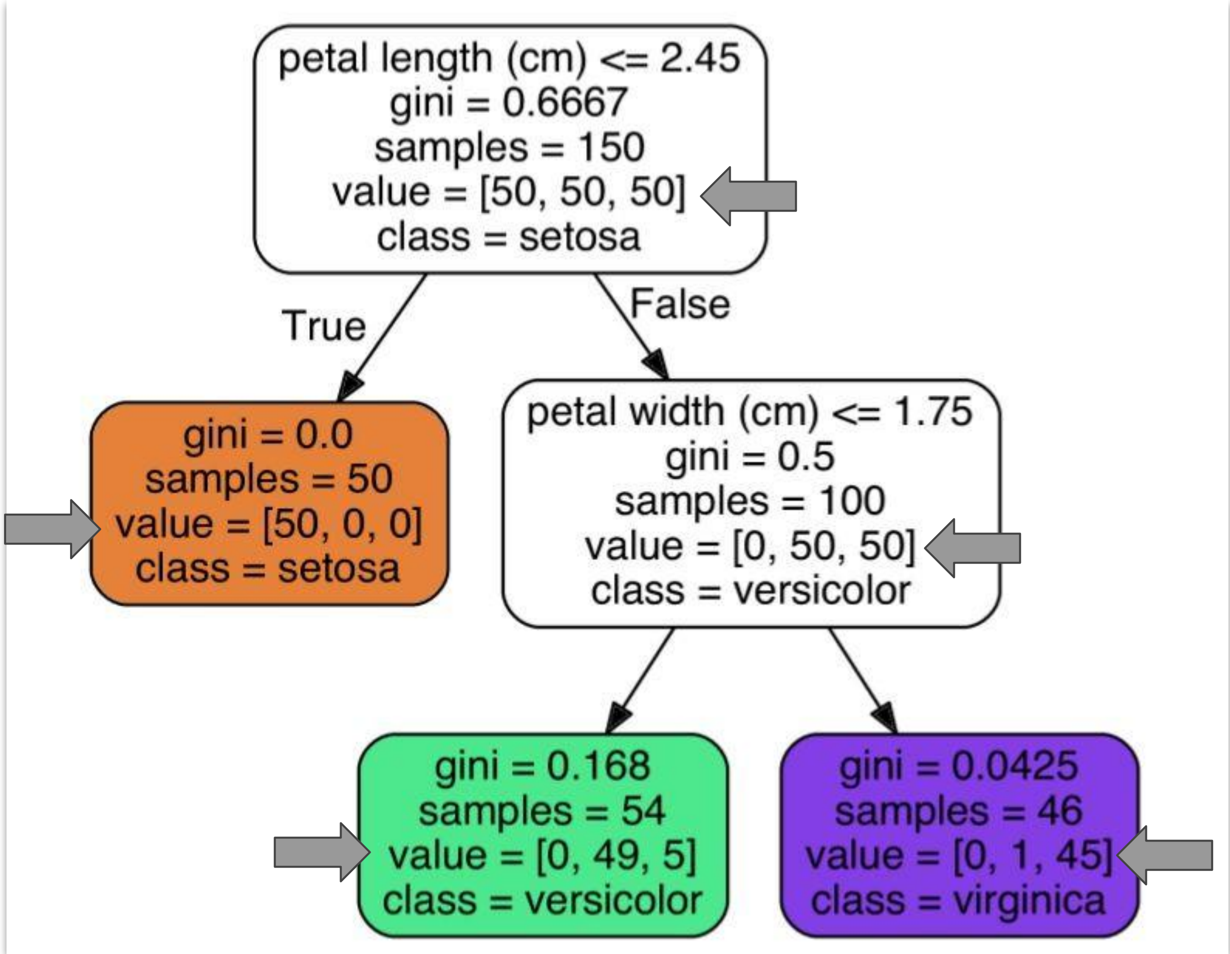
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target
tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X, y)

#You can visualize the trained Decision Tree by first using the export_graphviz() method
to output a graph definition file called iris_tree.dot:
from sklearn.tree import export_graphviz

export_graphviz(
     tree_clf,
     out_file="iris_tree.dot",
     feature_names=iris.feature_names[2:],
     class_names=iris.target_names,
     rounded=True,
     filled=True
)

$ dot -Tpng iris_tree.dot -o iris_tree.png#convert this .dot file to a variety of
formats such as PDF or PNG using the dot command-line tool from the graphviz package
```



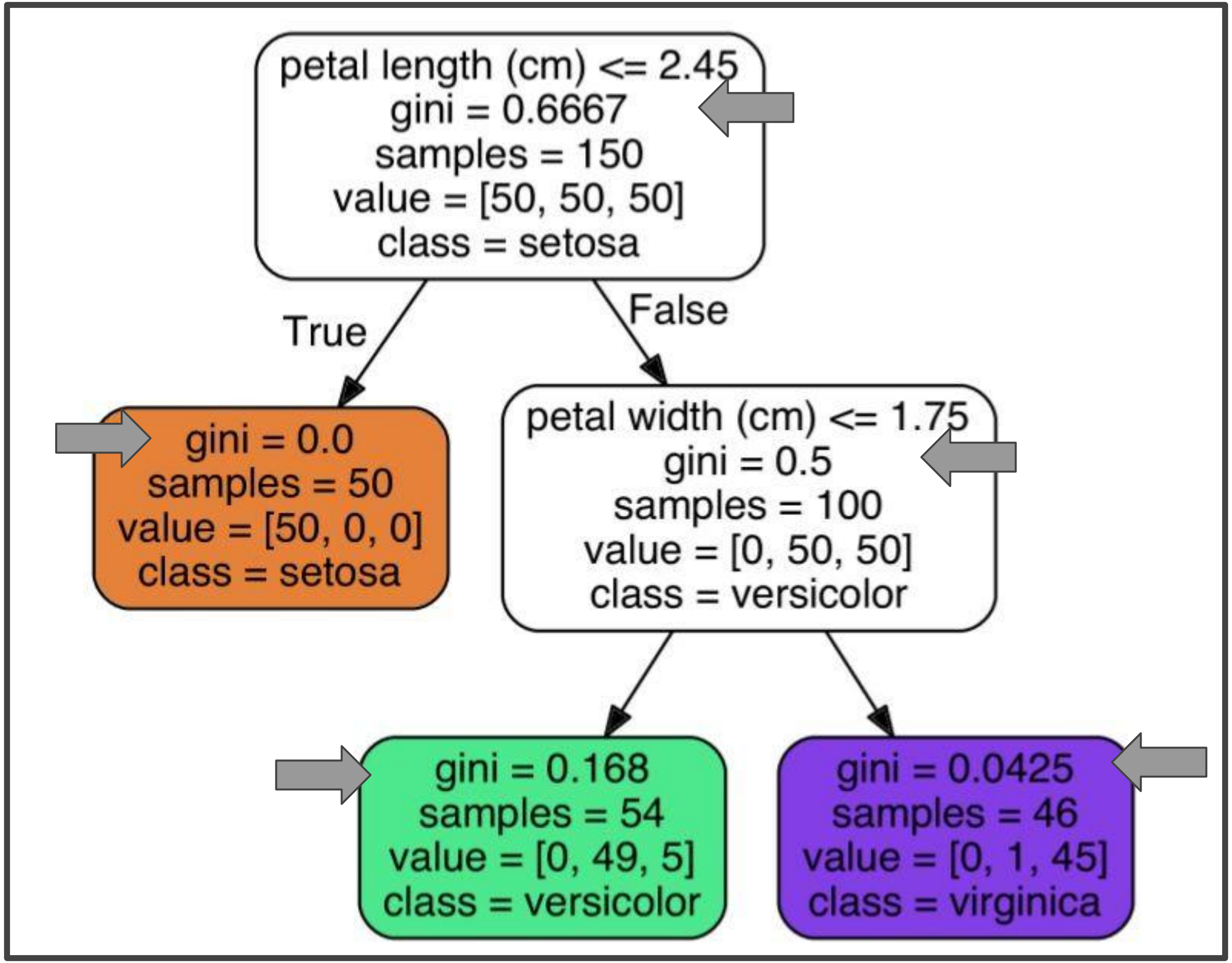
This node asks whether the flower's petal length is smaller than 2.45 cm



A **node's samples** attribute counts how many training instances it applies to.

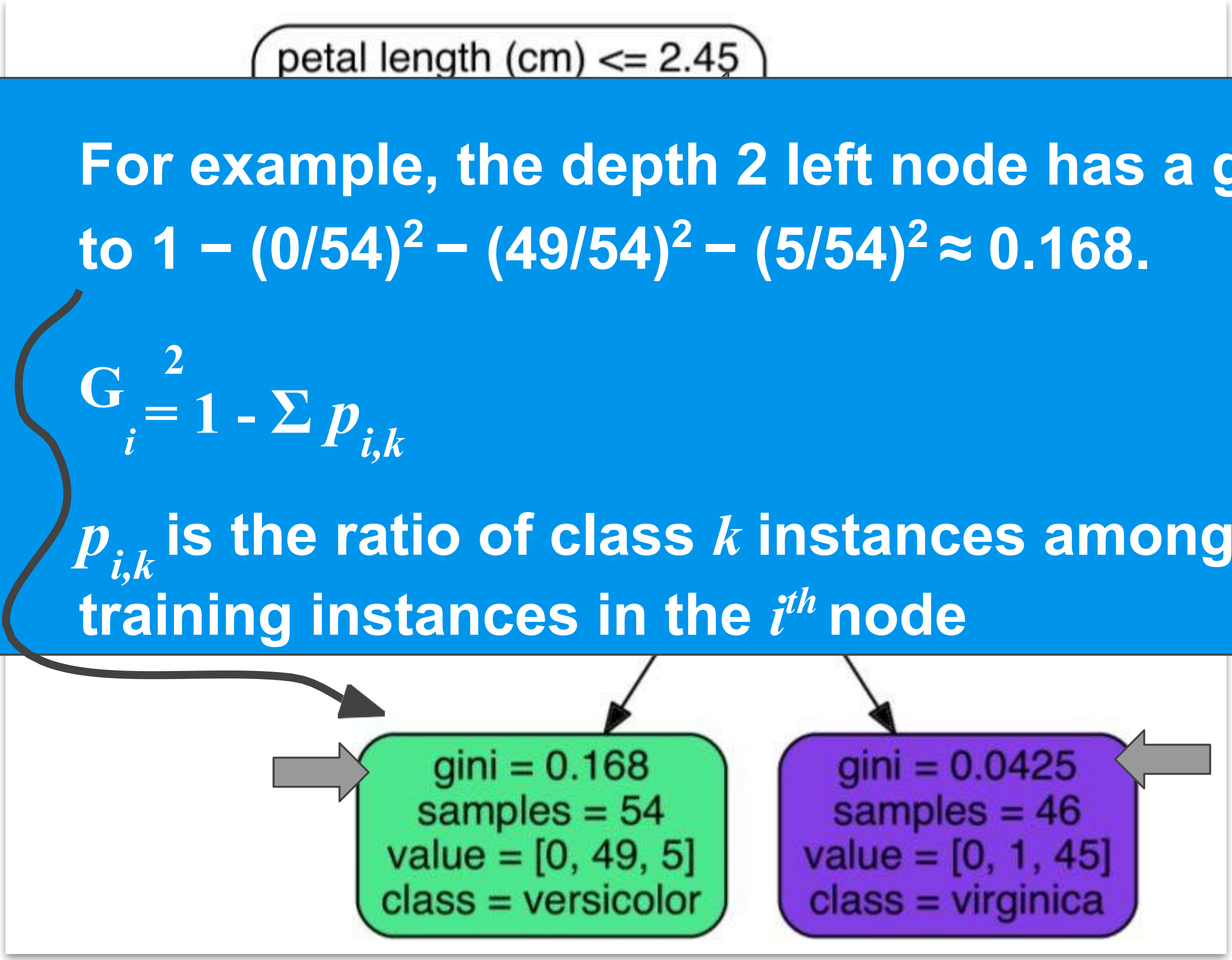


A **node's value** attribute tells you how many training instances of each class this node applies to.



A **node's gini** attribute measures its impurity.

"pure" (gini=0): all training instances belong to the same class.



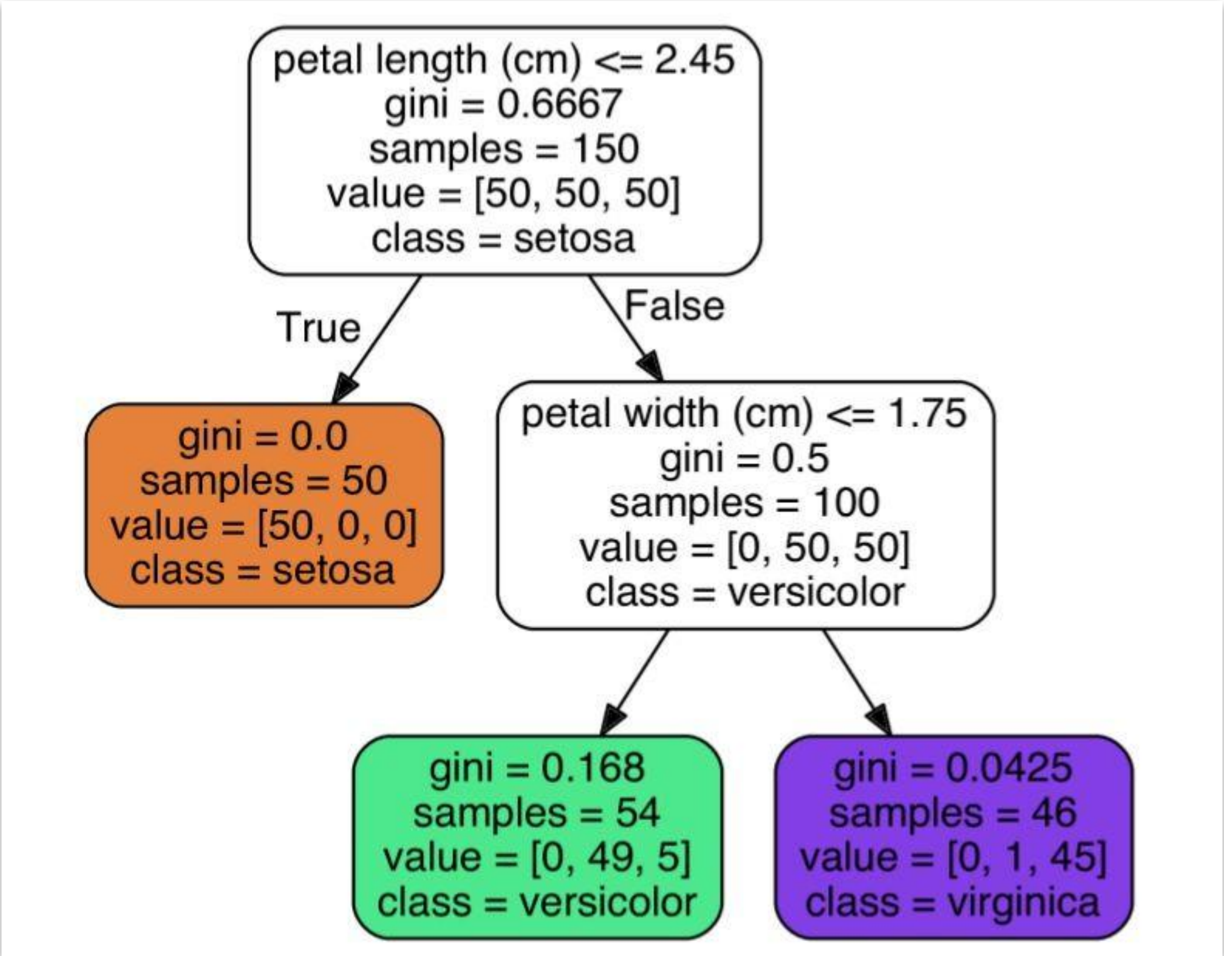
For example, the depth 2 left node has a gini score equal to $1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$.

$$G_i = 1 - \Sigma p_{i,k}^2$$

$p_{i,k}$ **is the ratio of class k instances among the training instances in the i^{th} node**

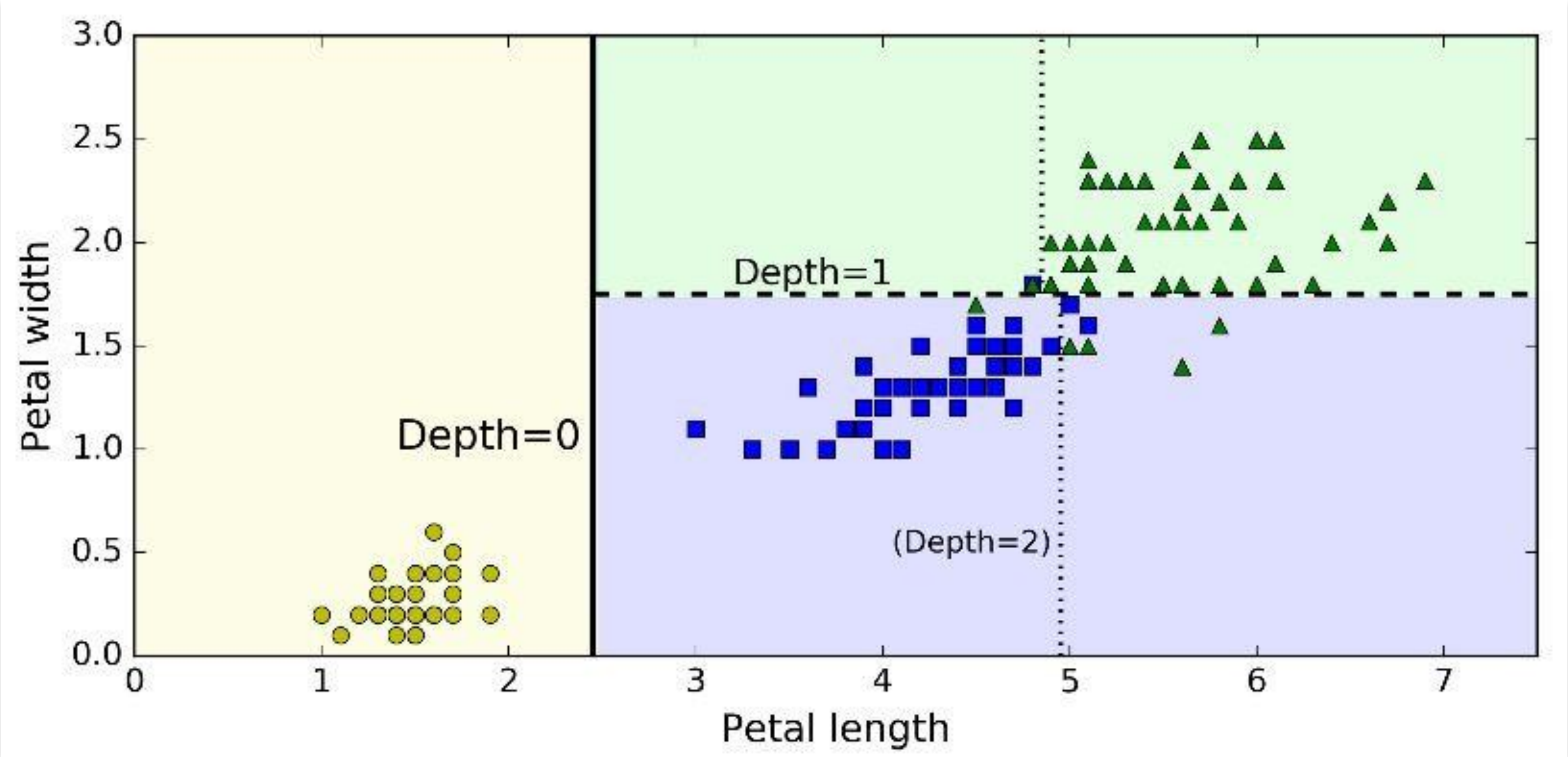
s

belong to the same class.



A **node's gini** attribute measures its impurity.

"pure" (gini=0): all training instances belong to the same class.


Depth=0
Depth=1
(Depth=2)
Petal width
Petal length
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5
6
7

Model Interpretation

- White Box vs Black Box
- Decision Trees are fairly intuitive and their decisions are easy to interpret (white box models).
- Random Forests or neural networks are generally considered black box models.

The CART Algorithm (Scikit-Learn default)

- Classification And Regression Tree (CART) algorithm.

The CART Algorithm (Scikit-Learn default)

- Classification And Regression Tree (CART) algorithm.
- The idea is really quite simple: the algorithm first splits the training set in two subsets using a single feature k and a threshold t_k (e.g. “petal length ≤ 2.45 cm”).

The CART Algorithm (Scikit-Learn default)

- Classification And Regression Tree (CART) algorithm.
- The idea is really quite simple: the algorithm first splits the training set in two subsets using a single feature k and a threshold t_k (e.g. “petal length ≤ 2.45 cm”).
- How does it choose k and t_k?

The CART Algorithm (Scikit-Learn default)

- Classification And Regression Tree (CART) algorithm.

- The idea is really quite simple: the algorithm first splits the training set in two subsets using a single feature k and a threshold t_k (e.g. “petal length ≤ 2.45 cm”).

- How does it choose k and t_k?

  It searches for the pair (k, t_k) that produces the purest subsets (weighted by their size).

The CART Algorithm (Scikit-Learn default)

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

$$\text{where } \begin{cases} G_{\text{left/right}} & \text{measures the impurity of the left/right subset,} \\ m_{\text{left/right}} & \text{is the number of instances in the left/right subset.} \end{cases}$$

CART cost function for classification

It stops recursing once it reaches the maximum depth (hyperparameter), or if it cannot find a split that will reduce impurity

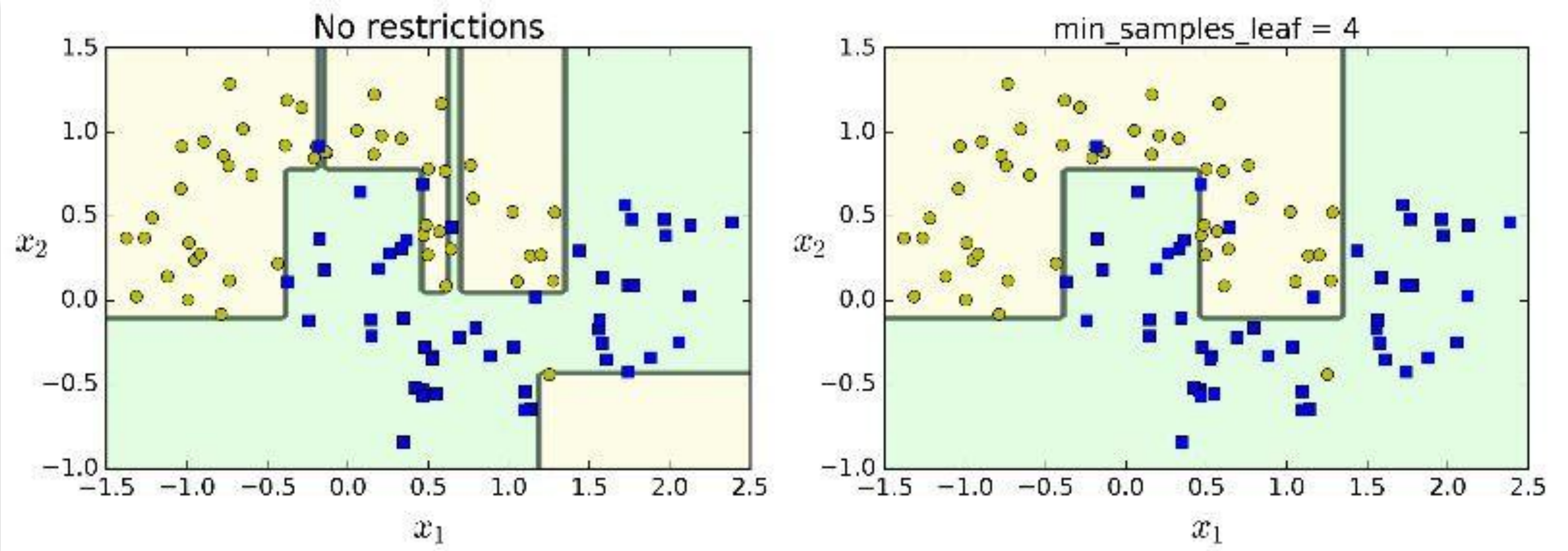
The CART Algorithm (Scikit-Learn default)

- CART algorithm is a greedy algorithm.
- Training: $O(n \times m \log(m))$, m samples, n features.
- Overall prediction: $O(\log2\ (m))$.

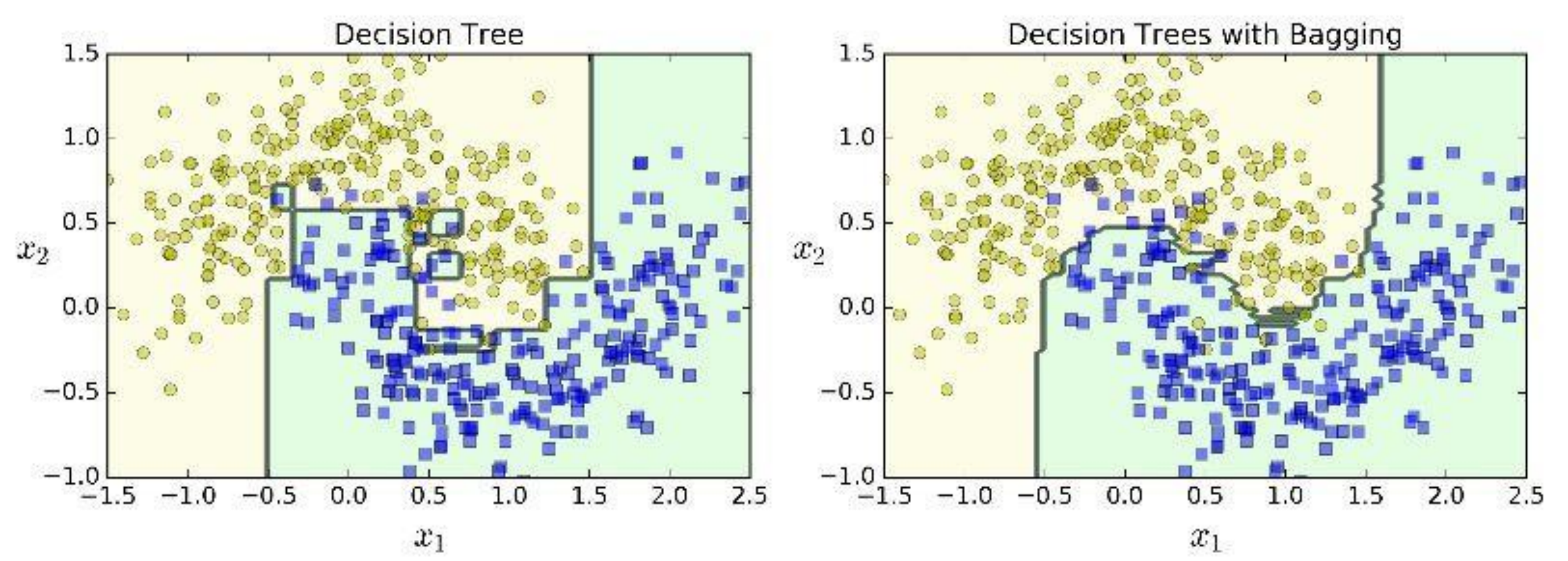
Regularization (in Scikit-Learn)

- `max_depth hyperparameter`: the default value is `None`, which means unlimited.
- `min_samples_split`: the minimum number of samples a node must have before it can be split.
- `min_samples_leaf`: the minimum number of samples a leaf node must have.

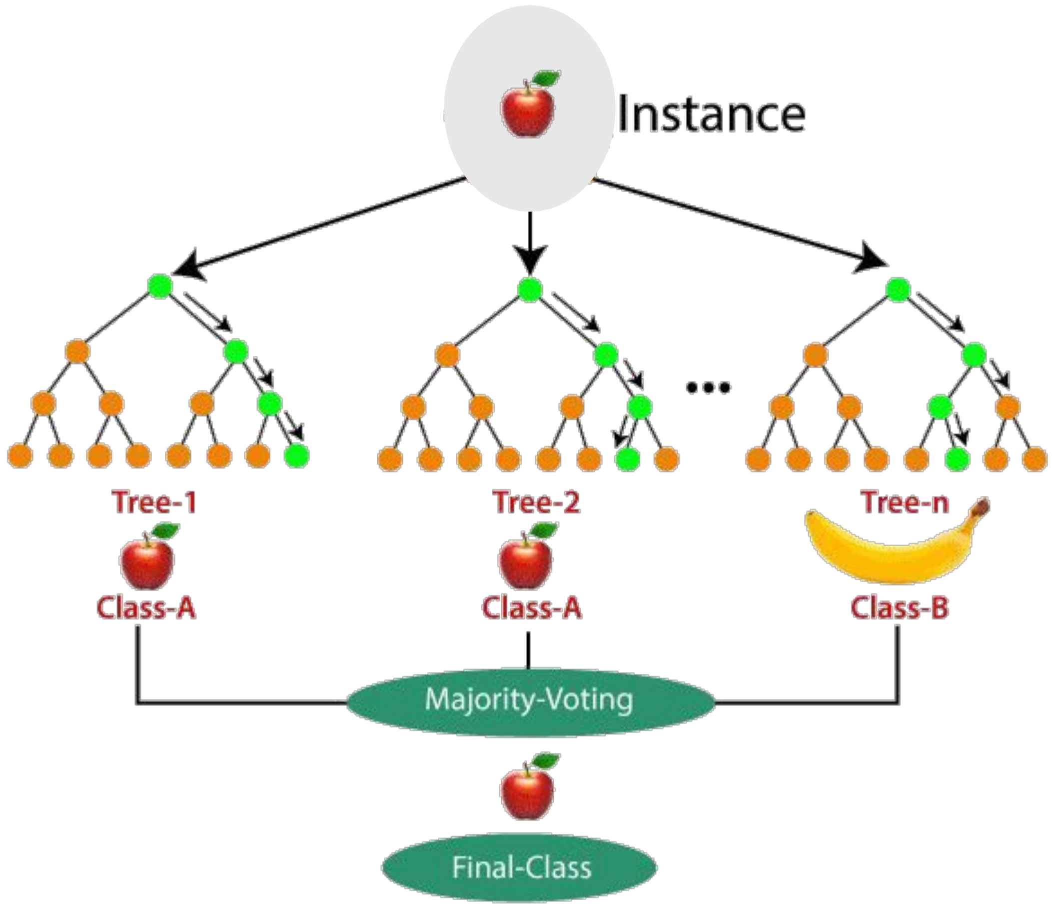
Regularization (in Scikit-Learn)



Regularization (in Scikit-Learn)



Random Forest


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

Random Forest [Ho, 1995]

- Random Forest is an ensemble of Decision Trees, generally trained using the Bagging method.

"Random Decision Forests", Tin Kam Ho (1995): http://goo.gl/zVOGQ1

Random Forest [Ho, 1995]

- Random Forest is an ensemble of Decision Trees, generally trained using the Bagging method.

- **Extra randomness when growing trees**:
    - Instead of searching for the very best feature when splitting a node, it searches for the best feature among a **random subset of features**.

"Random Decision Forests", Tin Kam Ho (1995): http://goo.gl/zVOGQ1

Random Forest [Ho, 1995]

1. Assume number of cases in the training set is *N*. Then, sample of these *N* cases is taken at random but with replacement.

Random Forest [Ho, 1995]

2. If there are *M* input variables, a number $m<M$ is specified such that at each node, *m* variables are selected at random out of the *M*.

   The best split on these *m* is used to split the node. The value of *m* is held constant while we grow the forest.

“Random Decision Forests”, Tin Kam Ho (1995): http://goo.gl/zVOGQ1

Slides largely based on Prof. Sandra Avila’s Machine Learning Course

Random Forest [Ho, 1995]

3. Each tree is grown to the largest extent possible and there is no pruning.

4. Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression).

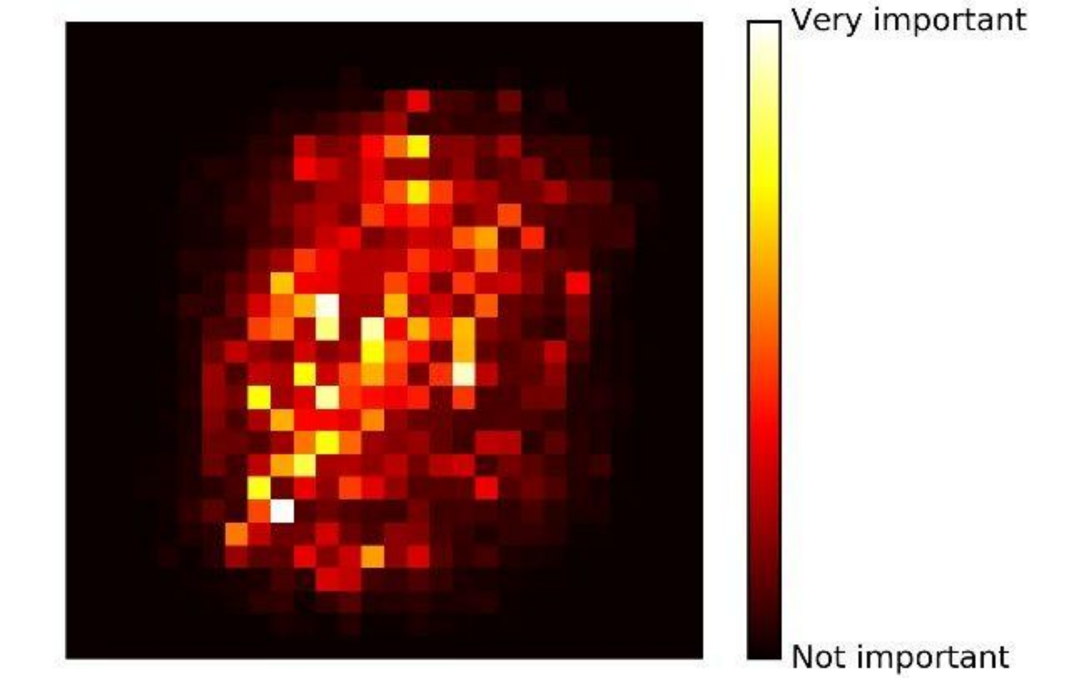
"Random Decision Forests", Tin Kam Ho (1995): http://goo.gl/zVOGQ1

Random Forest: Feature Importance

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier iris =
load_iris()
rnd_clf = RandomForestClassifier(n_estimators= 500, n_jobs= -1)
rnd_clf.fit(iris[ "data"], iris["target"])

for name, score in zip(iris["feature_names" ],
rnd_clf.feature_importances_):
    print(name, score)
```

```
sepal          length          (cm)
0.112492250999 sepal width (cm)
0.0231192882825  petal  length
(cm) 0.441030464364 petal width
(cm) 0.423357996355
```

Random Forest: Feature Importance



MNIST pixel importance (according to a Random Forest classifier)

Forward Thinking: Building Deep Random Forests

Kevin Miller, Chris Hettinger, Jeffrey Humpherys, Tyler Jarvis, and David K
Department of Mathematics
Brigham Young University
Provo, Utah 84602
millerk5@byu.edu, hettinger@math.byu.edu, jeffh@math.byu.e
jarvis@math.byu.edu, david.kartchner@math.byu.edu

Abstract

Distributed Deep Forest and its Application to Automatic Detection of Cash-out Fraud

Ya-Lin Zhang†, Jun Zhou‡, Wenhao Zheng†, Ji Feng†, Longfei Li‡, Ziqi Liu‡, Ming Li†, Zhiqiang Zhang‡, Chaochao Chen‡, Xiaolong Li‡, Zhi-Hua Zhou†
†National Key Lab for Novel Software Technology, Nanjing University, China
†{zhangyl, zhengwh, fengj, lim, zhouzh}@lamda.nju.edu.cn
‡Ant Financial Services Group, China
‡{jun.zhoujun, longyao.llf, ziqiliu, lingyao.zzq, chaochao.ccc, xl.li}@antfin.com

Training Big Random Forests with Little Resources

Fabian Gieseke
Department of Computer Science
University of Copenhagen
Copenhagen, Denmark
fabian.gieseke@di.ku.dk

Christia
Department of Co
University of
Copenhagen
igel@di.

ABSTRACT
Without access to large compute clusters, building random forests on large datasets is still a challenging problem. This is, in particular, the case if fully-grown trees are desired. We propose a simple yet effective framework that allows to efficiently construct ensembles of huge trees for hundreds of millions or even billions of training instances using a cheap desktop computer with commodity hardware. The basic idea is to consider a multi-level construction scheme, which builds top trees for small random subsets of the available data and which subsequently distributes all training instances to the top trees' leaves for further processing. While being conceptually simple, the overall efficiency crucially depends on the particular implementation of the different phases. The practical merits of our

ensembles in a parallel or distri
dividual compute nodes (e.g., by
node). While this can significa
such frameworks naturally requ
ing environments. Further, the e
might cause problems in case t
large to fit into the main memo
In this work, we propose a
scheme for building random fore
scale. The main idea is to build
phases: Starting with a top tree b
the data, one subsequently distr
leaves of that tree. For each leaf

Deep Forest: Towards an Alternative to Deep Neural Networks*

Zhi-Hua Zhou and **Ji Feng**
National Key Lab for Novel Software Technology, Nanjing University, Nanjing 210023, China
{zhouzh, fengj}@lamda.nju.edu.cn

Abstract
In this paper, we propose gcForest, a decision tree

ample, even when several authors all use convolutional neural networks [LeCun *et al.*, 1998; Krizhenvsky *et al.*, 2012; Simonyan and Zisserman, 2014], they are actually using dif-

References

Machine Learning Books

- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Chap. 6 & 7
- Pattern Recognition and Machine Learning, Chap. 14
- Pattern Classification, Chap 8 & 9 (Sec. 9.5)

- https://towardsdatascience.com/random-forest-in-python-24d0893d51c0