

# Supervised Learning: Classification

---

DS 8015

# OUTLINE

---

- 1 Logistic Regression
- 2 KNN
- 3 Decision Trees
- 4 Python Implementations
- 5 Cross-validation

# CLASSIFICATION: DEFINITION

---

- Given a collection of records (**training set**)
  - ⇒ Each record contains a set of **attributes**, one of the attributes is the class.
    - Find a model for class attribute as a function of the values of other attributes.
- Goal: previously unseen records should be assigned a class as accurately as possible.
- A test set is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

## APPLICATION — DIRECT MARKETING

---

- Goal: Reduce cost of mailing by targeting a set of consumers likely to buy a new cell-phone product.
- Approach:
  - Use the data for a similar product introduced before.
  - We know which customers decided to buy and which decided otherwise. These {buy, don't buy} decisions form the class attribute.
  - Collect various demographic, lifestyle, and company-interaction related information about all such customers.
    - ⇒ Type of business, where they stay, how much they earn, etc.
  - Use this information as input attributes to learn a classifier model.

## APPLICATION — FRAUD DETECTION

---

- Goal: Predict fraudulent cases in credit card transactions.
- Approach:
  - Use credit card transactions and the information on its account-holder as attributes.  
⇒ When does a customer buy, what does he buy, how often he pays on time, etc
  - Label past transactions as fraud or fair transactions. This forms the class attribute.
  - Learn a model for the class of the transactions.
  - Use this model to detect fraud by observing credit card transactions on an account.

## APPLICATION — CUSTOMER ATTRITION/CHURN

---

- Goal: To predict whether a customer is likely to be lost to a competitor.
- Approach:
  - Use detailed record of transactions with each of the past and present customers, to find attributes.
    - ⇒ How often the customer calls, where he calls, what time-of-the day he calls most, his financial status, marital status, etc.
  - Label the customers as loyal or disloyal.
  - Find a model for loyalty.

## EXAMPLE — CREDIT APPROVAL DECISIONS - 1

- We will analyze the Credit Approval Decisions data to predict how to classify new elements.
  - Categorical variable of interest: Decision (whether to approve or reject a credit application)
  - Predictor variables: shown in columns A-E

	A	B	C	D	E	F
1	Credit Approval Decisions					
2						
3	Homeowner	Credit Score	Years of Credit History	Revolving Balance	Revolving Utilization	Decision
4	Y	725	20	\$ 11,320	25%	Approve
5	Y	573	9	\$ 7,200	70%	Reject
6	Y	677	11	\$ 20,000	55%	Approve
7	N	625	15	\$ 12,800	65%	Reject
8	N	527	12	\$ 5,700	75%	Reject
9	Y	795	22	\$ 9,000	12%	Approve
10	N	733	7	\$ 35,200	20%	Approve
11	N	620	5	\$ 22,800	62%	Reject
12	Y	591	17	\$ 16,500	50%	Reject
13	Y	660	24	\$ 9,200	35%	Approve

## EXAMPLE — CREDIT APPROVAL DECISIONS - 2

### □ Modified Credit Approval Decisions

The categorical variables are coded as numeric:

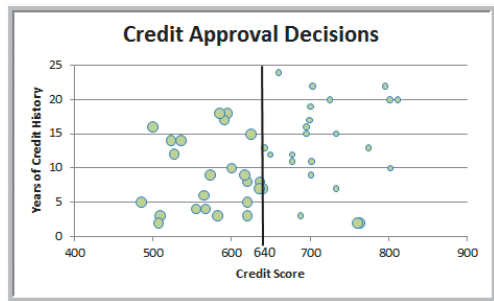
- Homeowner - 0 if No, 1 if Yes
- Decision - 0 if Reject, 1 if Approve

	A	B	C	D	E	F
1	Credit Approval Decisions					
2						
3	Homeowner	Credit Score	Years of Credit History	Revolving Balance	Revolving Utilization	Decision
4	1	725	20	\$ 11,320	25%	1
5	1	573	9	\$ 7,200	70%	0
6	1	677	11	\$ 20,000	55%	1
7	0	625	15	\$ 12,800	65%	0
8	0	527	12	\$ 5,700	75%	0
9	1	795	22	\$ 9,000	12%	1
10	0	733	7	\$ 35,200	20%	1
11	0	620	5	\$ 22,800	62%	0
12	1	591	17	\$ 16,500	50%	0
13	1	660	24	\$ 9,200	35%	1



## EXAMPLE — CREDIT APPROVAL DECISIONS - 3

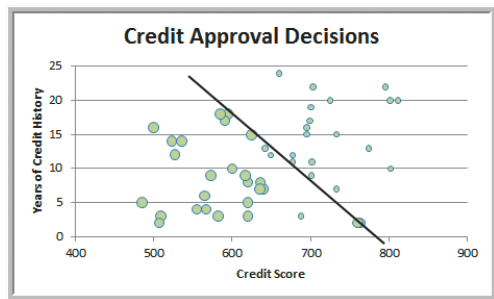
- Let large bubbles correspond to rejected applications
- Classification rule: Reject if  
credit score  $\leq 640$



⇒ 2 misclassifications out of 50 → 4%

## EXAMPLE — CREDIT APPROVAL DECISIONS - 4

- Classification rule: Reject if  $0.095(\text{credit score}) + (\text{years of credit history}) \leq 74.66$

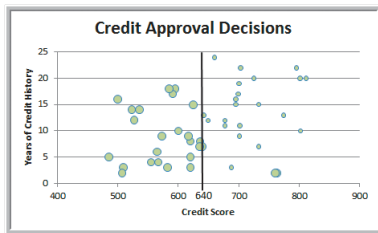


⇒ 3 misclassifications out of 50 → 6%

## EXAMPLE — CREDIT APPROVAL DECISIONS - 5

### □ Classification Matrix for Credit-Approval Classification Rules

Actual Classification	Predicted Classification	
	Decision = 1	Decision = 0
Decision = 1	23	2
Decision = 0	0	25



- Off-diagonal elements are the misclassifications
- Probability of a misclassification = 4%

# Logistic Regression

# LOGR FOR CLASSIFICATION - 1

---

- The Logistic Regression (LogR) model builds a regression model to predict the probability that a given data entry belongs to the category numbered as “1”.
- Just like Linear regression assumes that the data follows a linear function, Logistic regression models the data using the sigmoid/logistic function:

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

- Logistic regression becomes a classification technique only when a decision threshold is brought into the picture. The setting of the threshold value is a very important aspect of Logistic regression and is dependent on the classification problem itself.
- We will focus on binary classification where the target variable can have two possible types: “0” or “1”.
  - ⇒ LogR is also applicable to other types of classification problems (i.e., multinomial and ordinal classification).

## LOGR FOR CLASSIFICATION - 2

- The dataset has ' $p$ ' feature variables and ' $n$ ' observations. The feature matrix is represented as:

$$X = \begin{bmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n2} & \dots & x_{np} \end{bmatrix}$$

where  $x_{ij}$  denotes the values of  $j^{th}$  feature for  $i^{th}$  observation.

- The  $i^{th}$  observation,  $x_i$ , can be represented as:

$$x_i = [1, x_{i1}, \dots, x_{ip}]^T$$

- $h(x_i)$  represents the predicted response for  $i^{th}$  observation (i.e.  $x_i$ ).  $h(x_i)$  is also referred to as hypothesis.

## LOGR FOR CLASSIFICATION - 3

- The hypothesis used for prediction in linear regression:

$$h(x_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}$$

where  $\beta_0, \beta_1, \dots, \beta_p$  are the regression coefficients.

$$\beta = [\beta_0, \beta_1, \dots, \beta_p]^\top \Rightarrow h(x_i) = \beta^\top x_i$$

- A hypothesis for classification:

$$h(x_i) = g(\beta^\top x_i) = \frac{1}{1 + e^{-\beta^\top x_i}}$$

which ensures that  $0 \leq h(x_i) \leq 1$ .

- Conditional probabilities for two labels for  $i^{th}$  observation

$$P(y_i = 1 | x_i; \beta) = h(x_i)$$

$$P(y_i = 0 | x_i; \beta) = 1 - h(x_i)$$

$$\Rightarrow P(y_i | x_i; \beta) = (h(x_i))^{y_i} (1 - h(x_i))^{1-y_i}$$

## LOGR FOR CLASSIFICATION - 4

- A cost function similar to linear regression's (i.e. minimize total squared error) would not be appropriate as it would be non-convex for logistic regression.
- A cost function for logistic regression:

$$J(\beta) = \begin{cases} -\log(h(x)) & \text{if } y = 1 \\ -\log(1 - h(x)) & \text{if } y = 0 \end{cases}$$

which can be combined to a single function as

$$J(\beta) = \sum_{i=1}^n -y_i \log(h(x_i)) - (1 - y_i) \log(1 - h(x_i))$$

⇒ Find  $\beta$  that minimizes  $J(\beta)$ .



## LOGR FOR CLASSIFICATION - 5

- We take partial derivatives of  $J(\beta)$  to derive the *stochastic gradient descent rule*:

$$\frac{\partial J(\beta)}{\partial \beta_j} = (h(x) - y)x_j$$

where  $y$  and  $h(x)$  represent the response vector and predicted response vector.

### Stochastic Gradient Descent for LogR

Input:  $\mathbf{X}$ ,  $num\_iter$

Initialize weights  $\beta = (\beta_0, \beta_1, \dots, \beta_p)$

for  $k = 1 : num\_iter$

    set  $\alpha = 1/k$       (Learning rate)

    update weight vector for  $j$ th feature

$$\beta_j \leftarrow \beta_j - \alpha \sum_{i=1}^n (h(x_i) - y_i)x_{ij}$$

end for

return weights  $\beta$

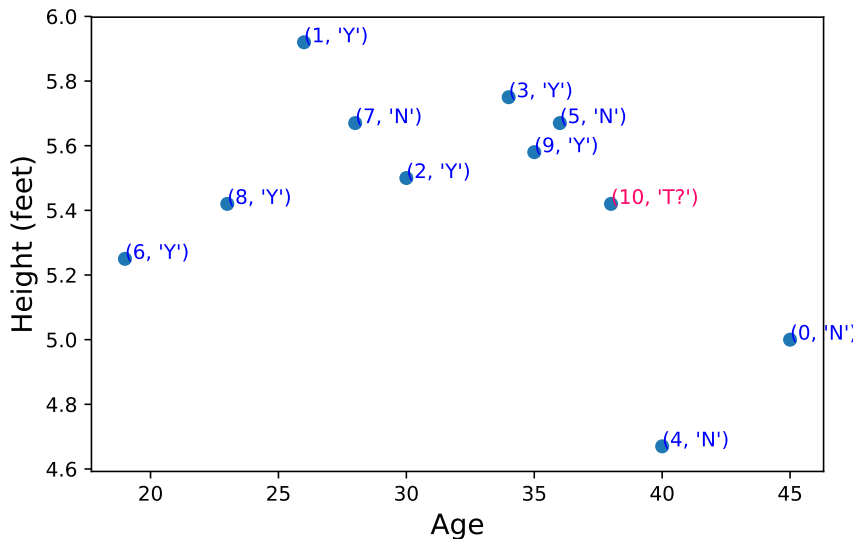
# KNN

# KNN FOR CLASSIFICATION

---

- KNN is a non-parametric, lazy learning algorithm.
  - A non-parametric method does not make any assumptions on the underlying data distribution.
  - A lazy algorithm does not use the training data points to do any generalization. In other words, there is no explicit training phase or it is very minimal.
- KNNs purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point.
- KNN algorithm is based on feature similarity: How closely out-of-sample features resemble our training set determines how we classify a given data point.  
⇒ Euclidean, Manhattan or Hamming Distance can be used to measure similarity.
- For the classification task, KNN algorithm classifies an object/instance by a majority vote of its neighbors, with the object being assigned to the class most common among its  $k$  nearest neighbors.

# KNN FOR CLASSIFICATION: EX - 1



# KNN FOR CLASSIFICATION: EX - 2

ID	Age	Height	Thin	distance
10	38	5.42	T?	0.0
5	36	5.67	N	2.02
4	40	4.67	N	2.14
9	35	5.58	Y	3.0
3	34	5.75	Y	4.01
0	45	5.0	N	7.01
2	30	5.5	Y	8.0
7	28	5.67	N	10.0
1	26	5.92	Y	12.01
8	23	5.42	Y	15.0
6	19	5.25	Y	19.0

Euclidean distance

ID	Age	Height	Thin	distance
10	0.79	-0.06	T?	0.0
9	0.38	0.41	Y	0.62
5	0.52	0.67	N	0.78
2	-0.29	0.17	Y	1.1
3	0.25	0.91	Y	1.11
7	-0.56	0.67	N	1.53
0	1.73	-1.29	N	1.55
8	-1.24	-0.06	Y	2.03
1	-0.83	1.41	Y	2.19
4	1.06	-2.26	N	2.22
6	-1.78	-0.56	Y	2.62

Euclidean distance over normalized data

# Decision Trees

# DECISION TREES (DT) FOR CLASSIFICATION

- Build a tree with a set of hierarchical decisions which eventually give a final result, i.e a classification/regression prediction.
  - ⇒ The decisions will be selected such that the tree is as small as possible while aiming for high classification/regression accuracy.
- A decision tree has two kinds of nodes
  1. Each leaf node has a class label, determined by majority vote of training examples reaching that leaf.
  2. Each internal node is a question on features. It branches out according to the answers.
- Decision Tree models are created using two steps: Induction and pruning.
  - **Induction** is where we actually build the tree, i.e set all of the hierarchical decision boundaries based on our data.
  - Because of the nature of training DTs they can be prone to overfitting.
    - ⇒ **Pruning** is the process of removing the unnecessary structure from a decision tree, effectively reducing the complexity to combat overfitting with the added bonus of making it even easier to interpret.

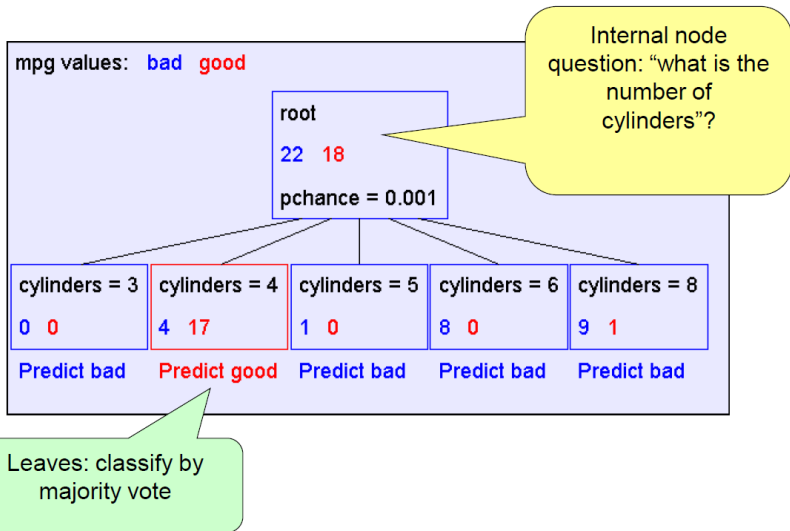
## DT: MPG EX-1



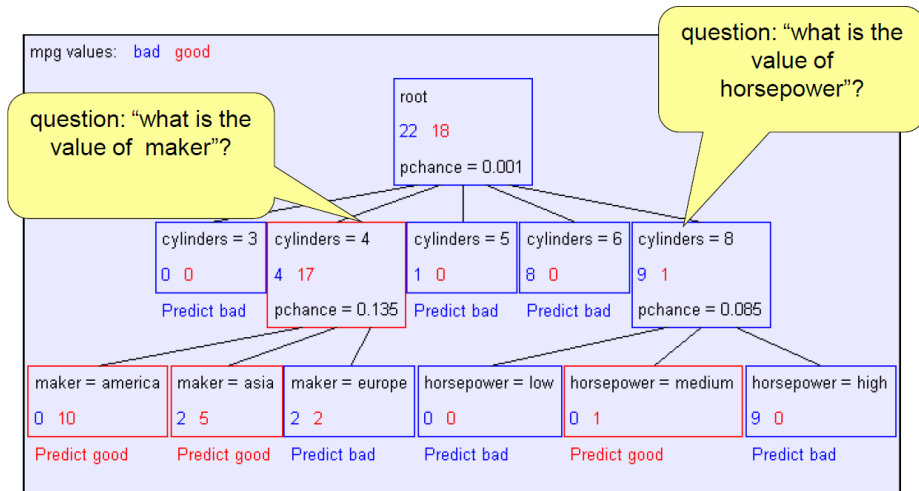
mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
good	4	low	low	low	high	75to78	asia
bad	6	medium	medium	medium	medium	70to74	america
bad	4	medium	medium	medium	low	75to78	europa
bad	8	high	high	high	low	70to74	america
bad	6	medium	medium	medium	medium	70to74	america
bad	4	low	medium	low	medium	70to74	asia
bad	4	low	medium	low	low	70to74	asia
bad	8	high	high	high	low	75to78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
bad	8	high	high	high	low	70to74	america
good	8	high	medium	high	high	79to83	america
bad	8	high	high	high	low	75to78	america
good	4	low	low	low	low	79to83	america
bad	6	medium	medium	medium	high	75to78	america
good	4	medium	low	low	low	79to83	america
good	4	low	low	medium	high	79to83	america
bad	8	high	high	high	low	70to74	america
good	4	low	medium	low	medium	75to78	europa
bad	5	medium	medium	medium	medium	75to78	europa



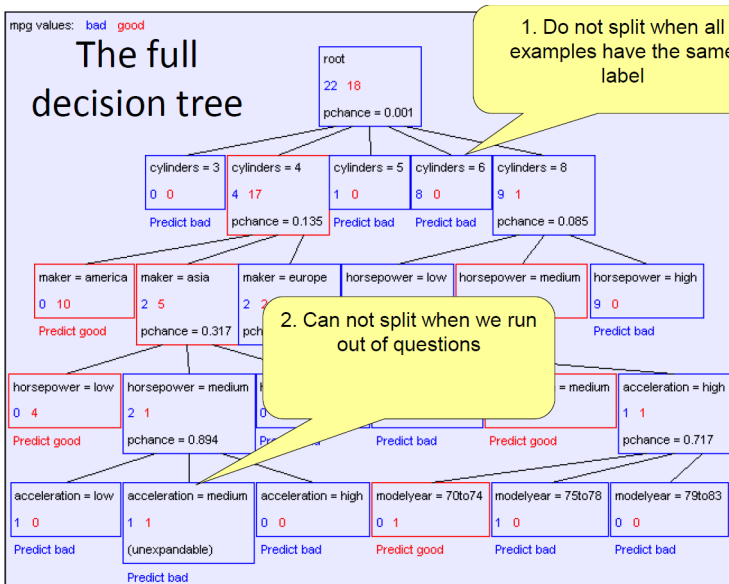
## DT: MPG EX-2



## DT: MPG EX-3



## DT: MPG EX-4



# DT ALGORITHM

---

```

buildtree(examples, questions, default)
'''
examples: a list of training examples
questions: a set of candidate questions, e.g.,
"what's the value of feature x_i"
default: default label prediction, e.g.,
over-all majority vote
'''

IF empty(examples) THEN return(default)
IF (examples have same label y) THEN return(y)
IF empty(questions) THEN return(majority vote in examples)
q = best_question(examples, questions)
Let there be n answers to q
  Create and return an internal node with n children
  The ith child is built by calling
    buildtree({example|q=ith answer}, questions\{q}, default)

```

## DT: THE BEST QUESTION

---

- What do we want: pure leaf nodes, i.e. all examples having (almost) the same  $y$ .
- A good question  $\Rightarrow$  a split that results in pure child nodes
- How do we measure the degree of purity induced by a question?

Here's one possibility:

**Mutual information = information gain**

$\Rightarrow$  A quantity from information theory

# DT: ENTROPY

- Entropy formula:

$$H(Y) = \sum_{i=1}^k -P(Y = y_i) \log_2(P(Y = y_i))$$

$$= \sum_{i=1}^k -p_i \log_2(p_i)$$

⇒ Interpretation: The number of yes/no questions (bits) needed on average to pin down the value of  $y$  in a random drawing.

- Conditional entropy:

$$H(Y|X = v) = \sum_{i=1}^k -P(Y = y_i|X = v) \log_2(P(Y = y_i|X = v))$$

$$H(Y|X) = \sum_{v: \text{ values of } X} P(X = v) H(Y|X = v)$$

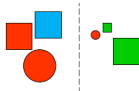
- $Y$  is label,  $X$  is a question (i.e. a feature),  $v$  is an answer to the question
- $P(Y|X = v)$  is the conditional probability

- Information gain (mutual info):  $I(Y; X) = H(Y) - H(Y|X)$

⇒ Choose question (feature)  $X$  which maximizes  $I(Y; X)$ .

## DT: ENTROPY EX

Example	Color	Shape	Size	Class
1	Red	Square	Big	+
2	Blue	Square	Big	+
3	Red	Circle	Big	+
4	Red	Circle	Small	-
5	Green	Square	Small	-
6	Green	Square	Big	-



- $H(class) = ?$
- $H(class|color) = ?$
- $I(class; color) = ?$

# DT: PRUNING TO AVOID OVERFITTING

- Pruning involves removing the branches that use features having low importance.
  - ⇒ Helps reducing complexity, thus increasing predictive power by reducing overfitting.
- The simplest method of pruning starts at leaves and removes each node with the most popular class in that leaf, this change is kept if it does not deteriorate accuracy.
- A common **approach for pruning**: a training and validation (tuning) set approach.
  - The available data are separated into two sets of examples:
    - ✓ a training set, which is used to form the learned hypothesis,
    - ✓ a separate validation set, which is used to evaluate the accuracy of this hypothesis, in particular, to evaluate the impact of pruning this hypothesis.
  - **Reduced-error pruning**: Consider each decision node to be candidates for pruning.
    - ⇒ Pruning a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node.
  - Nodes are removed only if the resulting pruned tree performs no worse than the original over the validation set.
    - ✓ Nodes are pruned iteratively, always choosing the node whose removal most increases the decision tree accuracy over the validation set.
    - ✓ Pruning of nodes continues until further pruning is harmful.



# DT: SUMMARY

---

- What if some (or all) of the features  $x_1, x_2, \dots, x_k$  are real-valued?
  - Idea 1: branch on each possible numerical value. (fragments the training data and prone to overfitting)
  - Idea 2: use questions in the form of  $(x_1 > t?)$ , where  $t$  is a threshold.
- Tree  $\rightarrow$  Rules
  - Each path, from the root to a leaf, corresponds to a rule where all of the decisions leading to the leaf define the antecedent to the rule, and the consequent is the classification at the leaf node.
  - For example, from the tree in the color/shape/size example, we could generate the rule: **if color=red and size=big then +**
- Decision trees are popular tools for data mining
  - Easy to understand/implement/use
  - Computationally cheap
  - Overfitting might happen

# Python Implementations

---

## PERFORMANCE METRICS FOR CLASSIFICATION

- Let TP, TN, FP, and FN refer to the number of true positive, true negative, false positive, and false negative classifications.

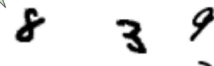



$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$\text{F1 score} = \left( \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \right) = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

# PERFORMANCE METRICS FOR CLASSIFICATION

		Predicted		
		Negative	Positive	
Actual	Negative	<div>TN</div> 	 <div>FP</div>	<div>Precision</div> <div>(e.g., 3 out of 4)</div>
	Positive	 <div>FN</div>	 <div>TP</div>	
<div>Recall</div> <div>(e.g., 3 out of 5)</div>				

# METHODS FOR CLASSIFICATION

---

- LogisticRegression
- KNeighborsClassifier
- LinearDiscriminantAnalysis
- QuadraticDiscriminantAnalysis
- GaussianProcessClassifier
- GaussianNB
- DecisionTreeClassifier
- RandomForestClassifier
- AdaBoostClassifier
- BaggingClassifier
- XGBClassifier
- VotingClassifier
- SVM
- MLPClassifier
- ...

See scikit-learn documentation [here!](#)

# CLASSIFICATION IMPLEMENTATION - 1

---

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, recall_score,
precision_score, classification_report

def evaluate_results(gtestLabels, gtestPred):
    d_accuracy = accuracy_score(gtestLabels, gtestPred)
    v_recall = recall_score(gtestLabels, gtestPred, average = None)
    v_prec = precision_score(gtestLabels, gtestPred, average = None)
    v_summaryReport = classification_report(gtestLabels, gTestPred,
        digits = 4)
    return d_accuracy, v_recall, v_prec
```

## CLASSIFICATION IMPLEMENTATION - 2

---

```
# LogisticRegression
```

```
clf = LogisticRegression()  
model = clf.fit(train_instances, train_labels)  
test_predictions = model.predict(test_instances)  
evaluate_results(test_labels, test_predictions)  
  
print('\n\n')  
print(clf.coef_)  
print(clf.intercept_)  
coefficients = pd.concat([pd.DataFrame(my_df.columns),  
                           pd.DataFrame(np.transpose(clf.coef_))], axis = 1)  
print(coefficients)
```

```
# KNeighborsClassifier
```

```
clf = KNeighborsClassifier(n_neighbors=2)  
model = clf.fit(train_instances, train_labels)  
test_predictions = model.predict(test_instances)  
evaluate_results(test_labels, test_predictions)
```

# CLASSIFICATION IMPLEMENTATION - 3

---

## # Decision Tree

```
clf = DecisionTreeClassifier()  
model = clf.fit(train_instances, train_labels)  
test_predictions = model.predict(test_instances)  
evaluate_results(test_labels, test_predictions)
```

## # Random Forest

```
clf = RandomForestClassifier()  
model = clf.fit(train_instances, train_labels)  
test_predictions = model.predict(test_instances)  
evaluate_results(test_labels, test_predictions)
```

## # XGBoost

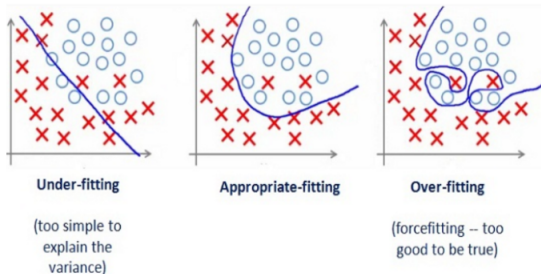
```
clf = XGBClassifier()  
model = clf.fit(train_instances, train_labels)  
test_predictions = model.predict(test_instances)  
evaluate_results(test_labels, test_predictions)
```



## Cross-validation

# OVERFITTING AND UNDERFITTING

- Two main reasons for the poor performance of a model:
  - the model is too simple to describe the target (underfitting — high bias)
  - the model is too complex to express the target (overfitting — high variance)



- Bias is reduced and variance is increased in relation to model complexity.
  - ⇒ As more and more parameters are added to a model, the complexity of the model rises and variance becomes our primary concern while bias steadily falls.
- Overfitting can be considered as focusing too much on the training set and learning complex relations which may not be valid in general for new data (test set).

# CROSS-VALIDATION

---

- Use the initial training data to generate multiple mini train-test splits.  
⇒ Use these splits to tune your model.
- In standard  $k$ -fold cross-validation, we partition the data into  $k$  subsets, called folds. Then, we iteratively train the algorithm on  $k-1$  folds while using the remaining fold as the test set (called the “holdout fold”).

```
from sklearn.model_selection import KFold,  
    cross_val_score  
  
cross_val_score(RandomForestClassifier(),  
    df_lp.drop("Loan_Status",1), df_lp["Loan_Status"],  
    cv=5, scoring='f1_macro')
```