Classification

Classification

- Briscoe, E., & Feldman, J. (2011). Conceptual complexity and the bias/variance tradeoff. *Cognition*, *118*(1), 2-16.
- Guyon, I., Elisseeff, A., 2006. An Introduction to Feature Extraction, in: Guyon, I., Nikravesh, M., Gunn, S., Zadeh, L. (Eds.), Feature Extraction, *Studies in Fuzziness and Soft Computing*. Springer Berlin Heidelberg, pp. 1–25
- Mitchell, T. M. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, *45*(37), 870-877.
- Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.

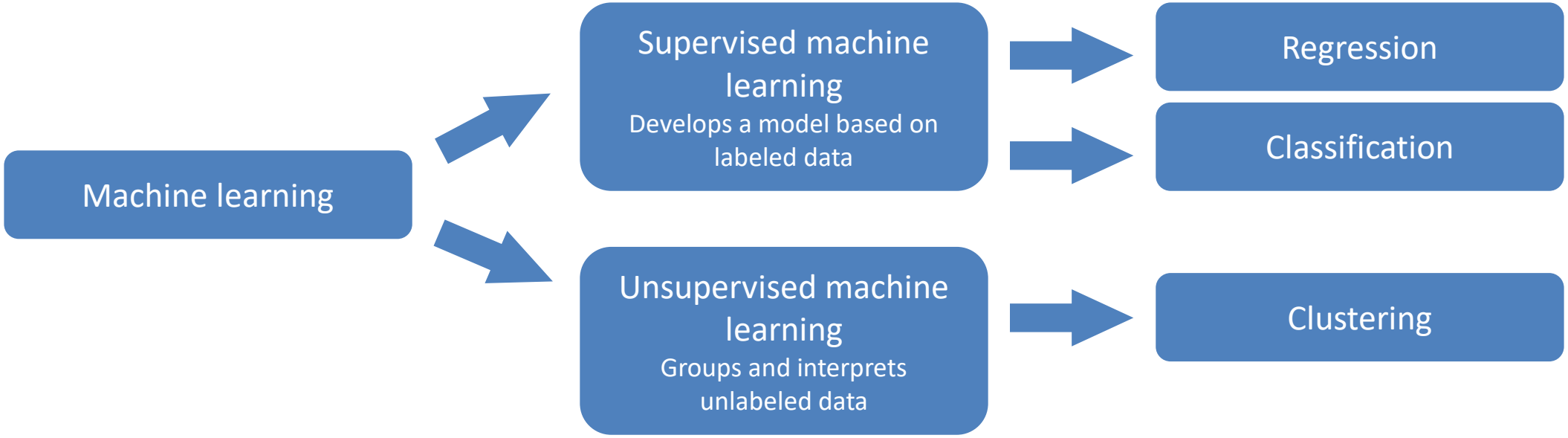
Classification

- **Introduction**
- Decision Trees
- Classifier Evaluation
- k-Nearest Neighbors (kNN)
- Naive Bayes
- Logistic Regression
- Advanced Classifier Evaluation

Recap: Supervised vs unsupervised learning

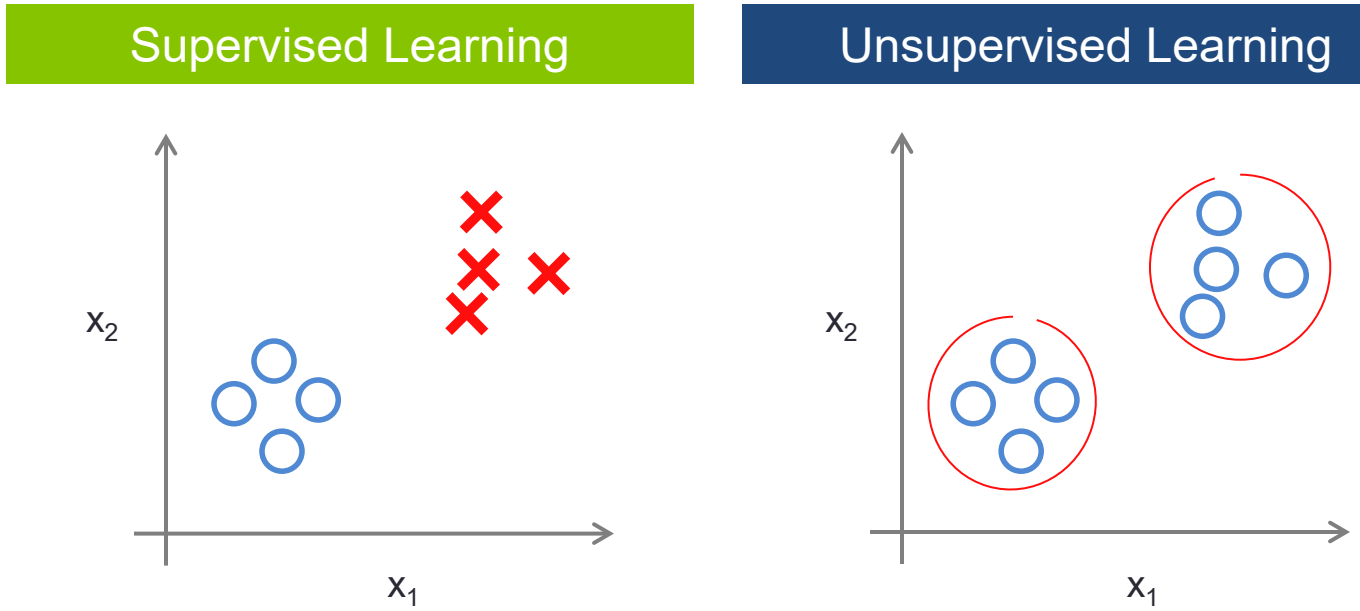
- Supervised machine learning ("Making predictions based on labeled data")
  - Refers to the process and techniques for assigning a label to each object by inferring a function from the data with supervision from a "teacher" (i.e., assigned labels on training samples)
  - Typical supervised learning tasks and approaches consist of regression, classification, or prediction.

- Unsupervised machine learning ("Finding structure in unlabeled data")
  - Refers to learning processes and approaches for which no labels are given and for which a learning system estimates the categorization and structures in its input
  - Unsupervised learning can discover hidden patterns, detect outliers in data, or conduct feature learning
  - Typical unsupervised learning approaches include clustering, profiling, data reduction



Recap: Supervised vs unsupervised learning

- Another example to show differences:



- Supervised Learning:
  - We know *ex-ante* that two categories exist (blue and red)
- Unsupervised learning
  - The result shows two groups (*ex-post)*.
- Supervised learning is analogous to learning with a teacher and then applying that knowledge to new data.

Source: Andrew Ng

Recap: Supervised vs unsupervised learning

- Classification is considered as an instance of supervised learning since labels of a test set are known and used to train the algorithm
  - Classification sorts observations to known classes based on training with a labeled data set
- Clustering is an unsupervised approach
  - No training data is needed and used
  - Clustering is often used to “reveal” characteristic groups
  - Clustering is discussed in the later lecture
- Classification and clustering are subsumed under the term “pattern recognition”.

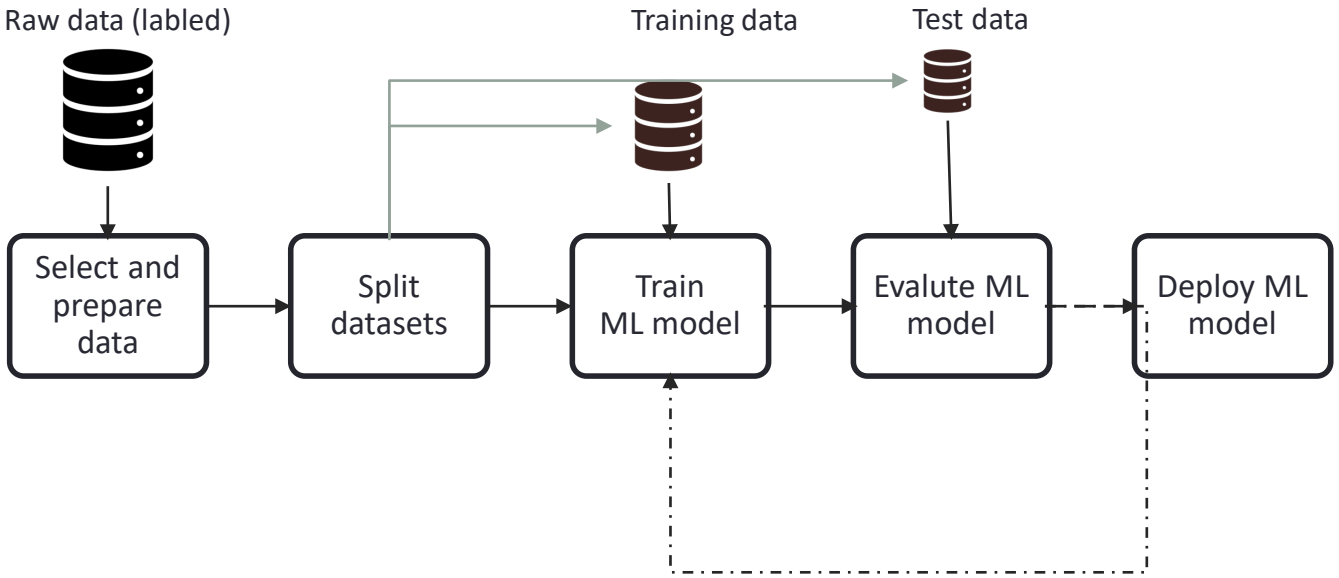
What is classification?

- Classification: Identifying to which sub-population a new observation belongs
  - Sub-populations are also called subgroups, groups, categories, or classes
  - Feature vectors describe observations.

- Training: Deriving how a set of explanatory variables indicate that an observation belongs to one subgroup
  - The decision is based on a training set of data where the group membership of each observation (with corresponding features) is known ("the data is labeled")
  - Explanatory variables are also called features
  - Explanatory variables may be, among others, ...
    - … categorical (employed/unemployed),
    - … ordinal (small or large apartment),
    - … integer (number of fridges),
    - … real-valued (income).

Classification process

- 1. Select and prepare data
  - Choosing features and manipulating the dataset
- 2. Split dataset
  - Train and test dataset
- 3. Train ML model
  - Input train dataset into a machine learning model
- 4. Evaluate ML model
  - If desired performance is not reached: tune the model and repeat Step 3, otherwise deploy ML model



Training data and test data

- In machine learning, we usually split our data into a training and test set

- Training Set
  - Used to learn the model
  - Often around 60-80% of the raw data

- Test Set
  - Separated from the training set
  - Used *only* to evaluate the accuracy of the learned model over subsequent data
  - No more learning / adjustment of parameters when applying the test set
  - Disjunct from the training data

- There are the advanced mechanism for splitting, training, and validating the data, e.g., cross-validation.

Example applications of classification

- Given a person's blood pressure, previous heart diseases, and blood lipid concentration
  - needs treatment (yes / no)
- Given an email's subject, body, and outgoing address
  - Classify as spam?
  - Adult content?
  - Important / time-critical?
- Giving the grocery shopping behaviour
  - High income?
  - Pregnant?
  - Alcoholics?
  - Bargainer?
  - Will he also purchase products X, Y, and Z?
- PC-user behaviour to recognize
  - Burn-outs
  - Stress levels
- Tracking history of goods to fight
  - Smuggling
  - Counterfeit goods
- Financial transactions to reveal
  - Tax evasion
  - Money laundry

Terminologies for classification

- Common terminologies in *statistics*
  - Properties of observation = independent variables or explanatory variables
  - Categories to be predicted = outcomes or dependent variables

- Common terminologies in *machine learning*
  - Properties of observation = instances /attributes /features
  - Categories to be predicted = classes

Running example

- As a healthcare data analyst, your job is to identify patients or sufferers that have a higher chance of a particular disease, for example, diabetes
- These predictions will help you to treat patients before the disease occurs
- As an analyst, you have first to define the problem that you want to solve using classification and then identify the potential features that predict the labels accurately
  - Features are the columns or attributes that are responsible for prediction.
  - In diabetes prediction problems, health analysts will collect patient information, such as the number of pregnancies the patient has had, their BMI, insulin level, age

- Terminology:
  - Classification is the process of categorizing customers into two or more categories.
  - The classification model predicts the categorical class label, such as whether the customer is potential or not.
  - In the classification process, the model is trained on available data, makes predictions, and evaluates the model performance.
  - Developed models are called classifiers

Running example

- The data set
  - Pregnancies: Number of times pregnant
  - Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
  - BloodPressure: Diastolic blood pressure (mm Hg)
  - SkinThickness: Triceps skinfold thickness (mm)
  - Insulin: 2-Hour serum insulin (mu U/ml)
  - BMI: Body mass index (weight in kg/(height in m)^2)
  - Pedigree: Diabetes pedigree function
  - Ages: age in years
  - Lable: Class variable (0 or 1)

```python
# Import libraries
import pandas as pd

# read the dataset
diabetes = pd.read_csv("diabetes.csv")

# Show top 5-records
diabetes.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Classification

- Introduction
- **Decision Trees**
- Classifier Evaluation
- k-Nearest Neighbors (kNN)
- Naive Bayes
- Logistic Regression
- Advanced Classifier Evaluation

Decision Trees

- **Foundations of decision trees**
- Splitting criteria for decision trees
- Tree pruning
- Summary
- Python example
- Hyperparameter for decision trees

Decision Trees

- A Decision Tree is a Supervised Machine Learning algorithm that looks like an inverted tree. Each node represents a feature, the link between the nodes represents a decision and each leaf node represents an outcome.

- Goal:
  - The goal of applying decision tree methods is to generate an ML model by which unknown data objects can be assigned to particular given data objects. This assignment is based on rules, which a classification tree can represent.

- Characteristics:
  - Supervised learning algorithm
  - One of the main techniques used in Data Science
  - Decision trees have a natural “if … then … else …” construction

Decision Trees

- Basic procedure
  - The total dataset is divided into a
    - a training set and
    - a test set.

- The training set is then successively split so that the resulting subsets (“partitions”) contain more homogeneous data sets concerning the classification variables.

- The partitioning of the datasets can be represented by a (decision) tree in which each node indexes a data set to which a homogeneity measure is assigned.

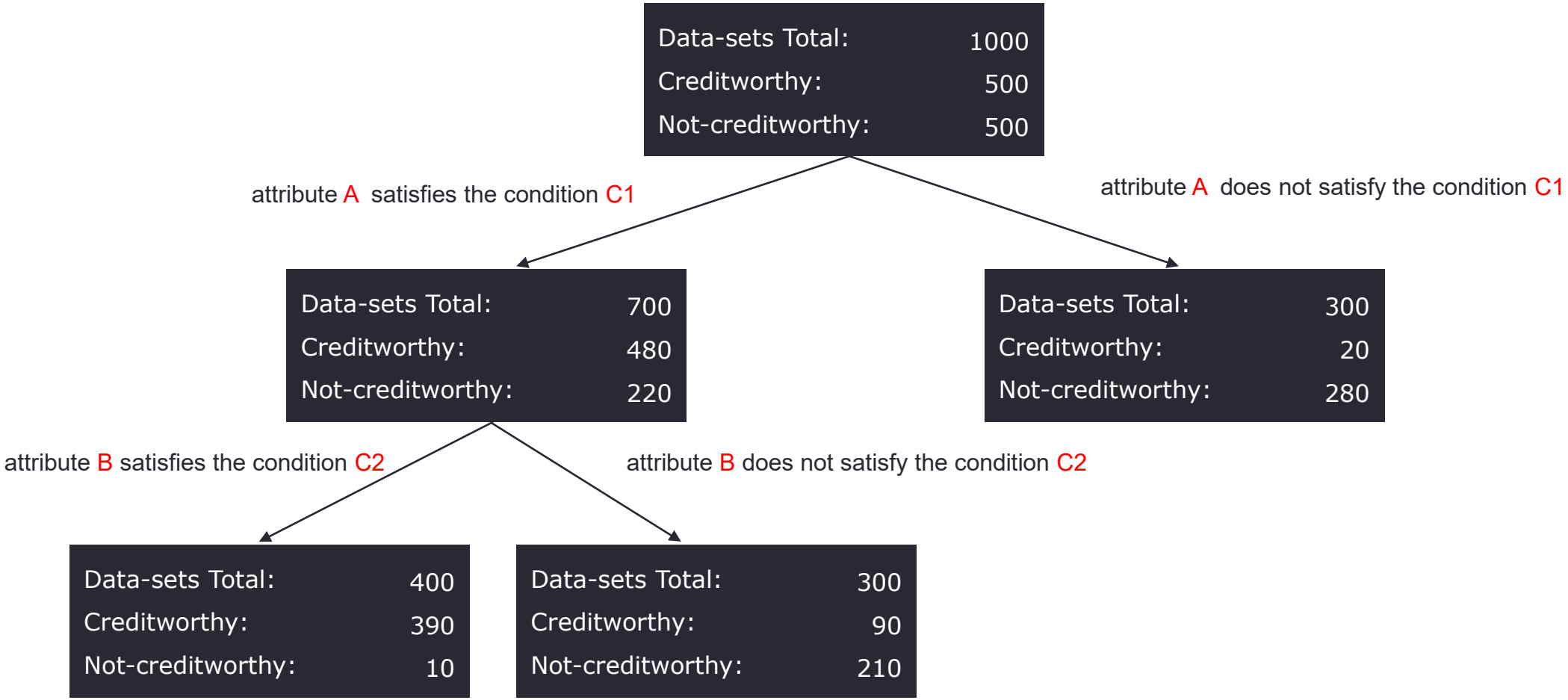
- If this homogeneity measure reaches a given value, the node is assigned to a certain class.

Decision Trees: Example decision tree for creditworthiness check

- The customer data (e.g.. Age, Income) is available for a creditworthiness check
- In this case
  - 50% of data records are referred to as „creditworthy” and
  - 50% of data records are referred to as „not creditworthy“.

- Task
  - By dividing the entire data based on a feature into two subsets, there are more observations with „creditworthy“ property in one subset and more observations with „not-creditworthy“ in another subset.
  - Thus, both subsets have a better homogeneity about the classification variables (creditworthiness) than the initial database.

Decision Trees: Example decision tree for creditworthiness check



Attribute A: Monthly income
Condition C1 : >= 3000

Attribute B: there exists collateral,
Condition C2: y/ n

Decision Trees: Example decision tree for creditworthiness check

- Rules
  - After such a tree has been generated, each new data record can now be assigned to its *expected* class based on the tree

- * In the example before, these are :

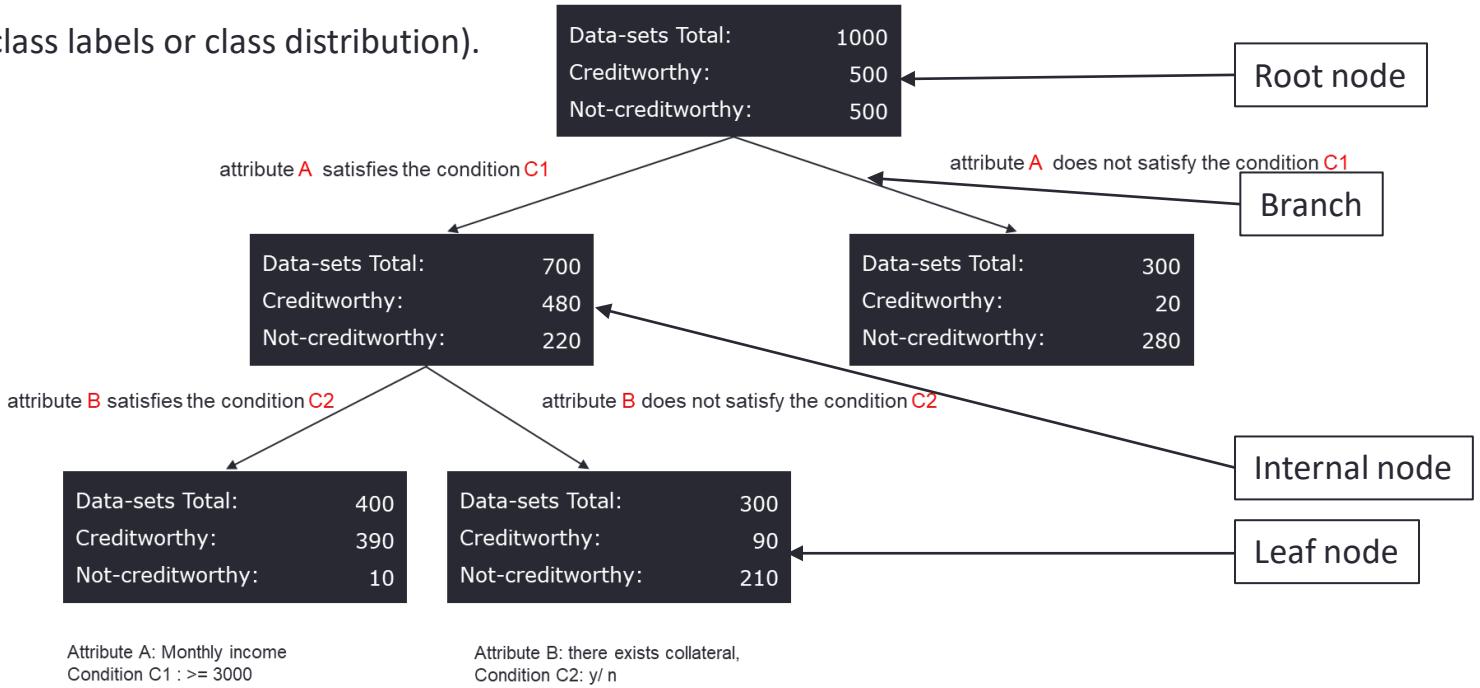
1. IF a minimal monthly income is not exceeded,
   THEN the credit is not granted.

2. IF a minimal monthly income is exceeded
   AND collateral security is provided,
   THEN the credit is granted.

3. IF a minimal monthly income is exceeded
   AND no collateral security is provided,
   THEN the credit is not granted.

Decision Trees

- Decision Trees consist of:
  - Root node:
    The root node is the starting point of a tree. At this point, the first split is performed.
  - Internal nodes:
    Test for the value of a certain attribute, that eventually leads to the prediction of the outcome.
  - Branch:
    Correspond to the outcome of a test and connect to the next node or leaf.
  - Leaf nodes:
    Terminal nodes that predict the outcome (represent class labels or class distribution).



Decision Trees

- Almost all decision tree algorithms follow the same procedure:
- Given
  - Training set T
  - Minimum value for the node homogeneity: min-conf

- Algorithm (Pseudocode)

```
DecisionTreeConstruction (Trainingset T, Float min-conf)

        if at least min-conf of objects from T in Class c then return;
        else
                for each Attribute A do
                        for each possible Split of A do
                        evaluate the quality of the partitioning, that would result from the split; do the
                                best of all of these splits; with T1, T2, .. . , Tm as the partitions resulting
                                from this split;

DecisionTreeConstruction( T1, min-conf);
…
DecisionTreeConstruction( Tm, min-conf);
```

Decision Trees

- Foundations of decision trees
- **Splitting criteria for decision trees**
- Tree pruning
- Summary
- Python example
- Hyperparameter for decision trees

Decision Trees: splitting criteria

- Splitting criteria:
  - *„do the best of all of these splits".*
    → *How do we decide what attribute best splits the data?*
    → *There exist attribute selection measures, i.e., split criteria, which we focus on now.*

  - Split criteria are based on a *homogeneity measure*, which can be defined, e.g., with the help of the relative frequency P_i of the occurrence of certain data records of a certain class *i*.

	Number of data records	Relative frequency
Total	100	-----
Class 1	30	p1 = 30%
Class 2	50	p2 = 50%
Class 3	20	p3 = 20%

  - The more different these frequencies P; are, the more homogeneous a node is.
  - A distribution concentrated on one characteristic would be ideal, e.g. (100%, 0%, 0%).

- Popular splitting criteria:
  - Information Gain
  - Gini index

Decision Trees: information gain

- Information gain:
  - Befor explaining information gain, we need to explain (Shannon) entropy
  - Information gain uses entropy to express the information content of a variable:
    - $Entropy(T) = -\sum_{i=1}^{k} p_i \cdot \log_2(p_i)$
  - Where p_i is the probability of randomly picking an element class C_i (i.e., the proportion of the dataset made up of class C_i). $\frac{|C_{i,D}|}{|D|}$
  - Entopy is the average amount of information needed to identify the class label of a tuple in D.
  - The Entropy describes the homogeneity ("disorder", "contamination") of a node T
  - The entropy of a node can have values between 0 and 1
  - The entropy increases with decreasing homogeneity until there is a uniform distribution
    - Entropy (T) = 0 ⇔ node T is homogeneous
    - Entropy (T) = 1 ⇔ node T is inhomogeneous (for number of classes k = 2 with p_i = 1/2)

Decision Trees: information gain

- Example: entropy
  - The easiest way to understand this is with an example. Consider a dataset with 1 blue, 2 greens, and 3 reds:
    - O OO OOO

$$Entropy\ (T) = -(p_b \log_2 p_b + p_g \log_2 p_g + p_r \log_2 p_r)$$

We know $p_b = \frac{1}{6}$ because $\frac{1}{6}$ of the dataset is blue. Similarly, $p_b = \frac{2}{6}$ (greens) and $p_b = \frac{3}{6}$ (reds)

Thus,

$$Entropy\ (T) = -(\frac{1}{6}\log_2(\frac{1}{6}) + \frac{2}{6}\log_2(\frac{2}{6}) + \frac{3}{6}\log_2(\frac{3}{6}) = 1.46$$

Decision Trees: information gain

- *Information gain (IG)* is defined for the split of a node T according to the different values of an attribute A

$$IG(T, A) = Entropy(T) - \sum_{a \epsilon A} \frac{|T_a|}{|T|} \cdot Entropy(T_a)$$

- The following applies:
  - The information gain IG describes the expected reduction in entropy if the value a of the attribute A is known.
  - The IG is calculated for all attributes that have not yet been taken into account in the tree structure.
  - For the expansion or splitting of the tree, the attribute that has the *largest IG* is selected

Decision Trees: information gain

- Information gain:
  - Dealing with numeric variables:
    - Suppose that attribute A is continuous-valued (raw values on age).
    - We must determine the “best” split-point for A.
    - Sort the values of A in increasing order.
    - Typically, the midpoint between each pair of adjacent values is a possible split-point:

$$\frac{a_i + a_{i+1}}{2}$$

    - Given v values of A, v − 1 possible splits are evaluated.
    - For each possible split-point for A, evaluate $Info_A(D)$
    - Choose the best split (with minimal $Info_A(D)$).

Decision Trees: exercise information gain

- Exercise: Information gain
  - What attribute has the highest information gain? Make a split from the node

RID	age	income	student	credit_ rating	class: buys.computer
1	youth	high	no	fair	no
2	youth	high	no	fair	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Decision Trees: exercise information gain

- Exercise: Information gain - Solution

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

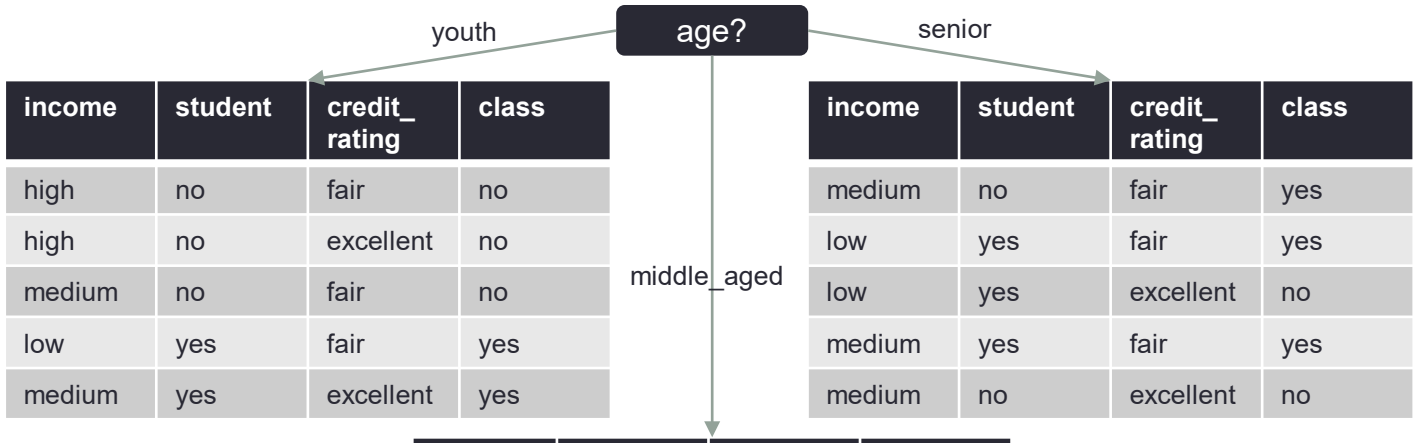
$$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) + \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right) + \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) = 0.694 \text{ bits.}$$

$Gain(age) = Info(D) - Info_{age}(D) = 0.940. - 0.694 = 0.246$

$Gain(income) = 0.029$ bits,

$Gain(student) = 0.151$ bits,

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



income	student	credit_ rating	class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	student	credit_ rating	class
high	no	fair	yes
low	yes	excellent	yes
medium	no	fair	yes
high	yes	fair	yes

income	student	credit_ rating	class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

Gain(student) = 0.151 bits,

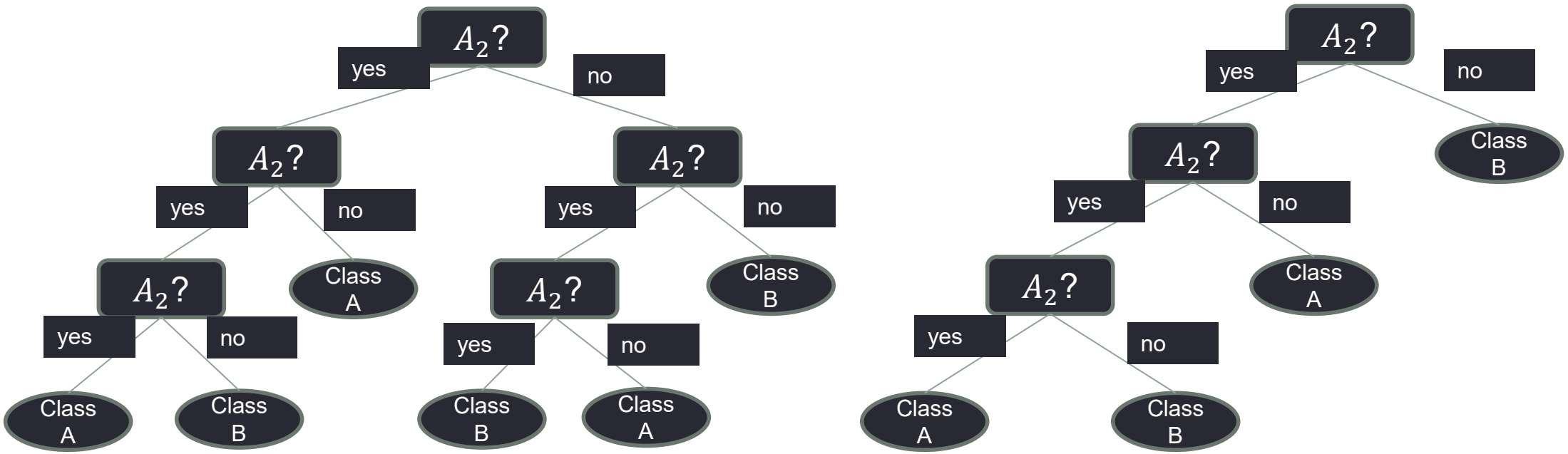
Gain(credit_rating) = 0.048 bits.

Decision Trees

- Foundations of decision trees
- Splitting criteria for decision trees
- **Tree pruning**
- Summary
- Python example
- Hyperparameter for decision trees

Decision Trees

- Tree pruning
  - Is needed to avoid data overfitting
  - Two approaches:
    - **Prepruning**, use some attribute selection measure to halt the tree construction early. The node becomes a leaf, e.g. using majority voting.
    - **Postpruning**, remove branches from the fully grown tree, based on the most common class in the sub-tree (“class B” on the figure).



Pruning

- One of the well-known post pruning strategies is *error reduction pruning*
  - Here, in each pruning step, the subtree T of the overall tree E is determined, the removal of which reduces the classification error on the validation set to the greatest extent.
  - This subtree is then removed.

The „ Error Reduction Pruning „ Algorithm (nach Ester und Sander (2000, S. 134))

ErrorReductionPruning (decision tree E, valditation set V)

let *BestError* be the classification error from E to V;

B:= *E;*

loop

for each node *K* aus *B* do

delete *K* and its associated subtree from *B* and get *B-K*; determine the classification error *NewError* from *B-K* to *V*;

note the minimum of the values for *NewError* in *BestNewError,*

if *BestNewError* < *BestError* then

remove the corresponding node *K* and its subtree from *B*,

i.e. B:= B-K;

note as *BestError* the classification error for the new *B* from *V*;

else return B;

Other methods take into account the cost of misclassification.

Decision Trees

- Foundations of decision trees
- Splitting criteria for decision trees
- Tree pruning
- **Summary**
- Python example
- Hyperparameter for decision trees

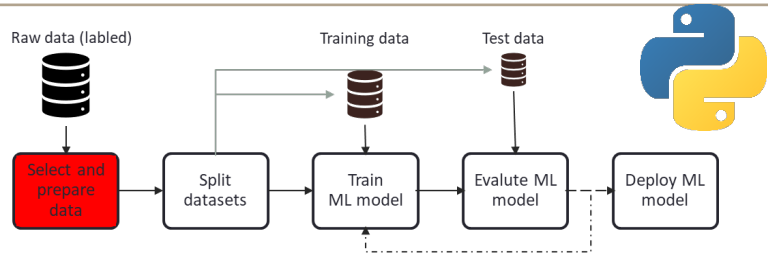
Decision Trees

- Evaluation:
  - Pros:
    - Easy to understand
    - Useful in data exploration:
      E.g., when working on a problem with information available in hundreds of variables, the decision tree will help identify the most significant variable.
    - Decision trees implicitly perform variable screening or **feature selection.**
    - Decision trees require relatively little effort from users for data preparation.
      → **does not require feature scaling**
    - Data type is not a constraint (numerical and categorical variables)
    - Non-parametric method
    - **Non-linear relationships** between parameters do not affect tree performance.
  - Cons:
    - Overfitting: Decision-tree learners can create over-complex trees that do not generalize the data well
      (this problem can get solved by setting constraints on model parameters and pruning)
    - Decision trees can be unstable because small variations in the data might result in a completely different tree being generated.
    - Information gain in a decision tree with categorical variables gives a biased response for attributes with a greater no. of categories.
    - Generally, it gives low prediction accuracy for a dataset compared to other machine learning algorithms.

Decision Trees

- Foundations of decision trees
- Splitting criteria for decision trees
- Tree pruning
- Summary
- **Python example**
- Hyperparameter for decision trees

Decision Trees in Python



Decision Tree Classification

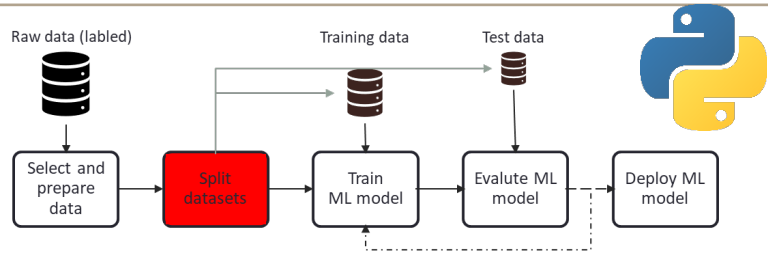
Select and prepare data

```
# Import libraries
import pandas as pd

# read the dataset
diabetes = pd.read_csv("diabetes.csv")

# Show top 5-records
diabetes.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# Splitting dataset in two parts: feature set and target label
feature_set = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree','skin']

X = diabetes[feature_set]
y = diabetes.label
```

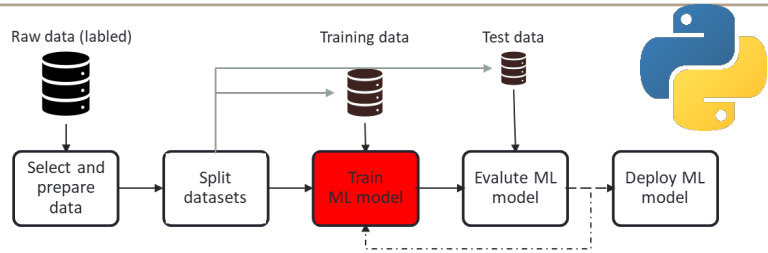
Decision Trees in Python



Split dataset

```
# Partition data into training and testing set
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

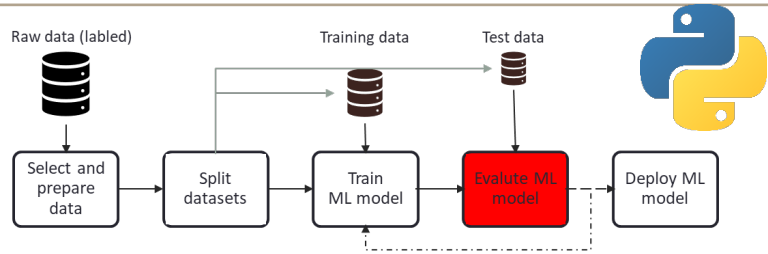
Decision Trees in Python



Train decision tree model

```
# Import Decision Tree model
from sklearn.tree import DecisionTreeClassifier

# Create a Decision Tree classifier object
clf_tree = DecisionTreeClassifier(random_state = 42, min_samples_leaf = 0.04)

# Train the model using training dataset
clf_tree = clf_tree.fit(X_train,y_train)
```

Decision Trees in Python

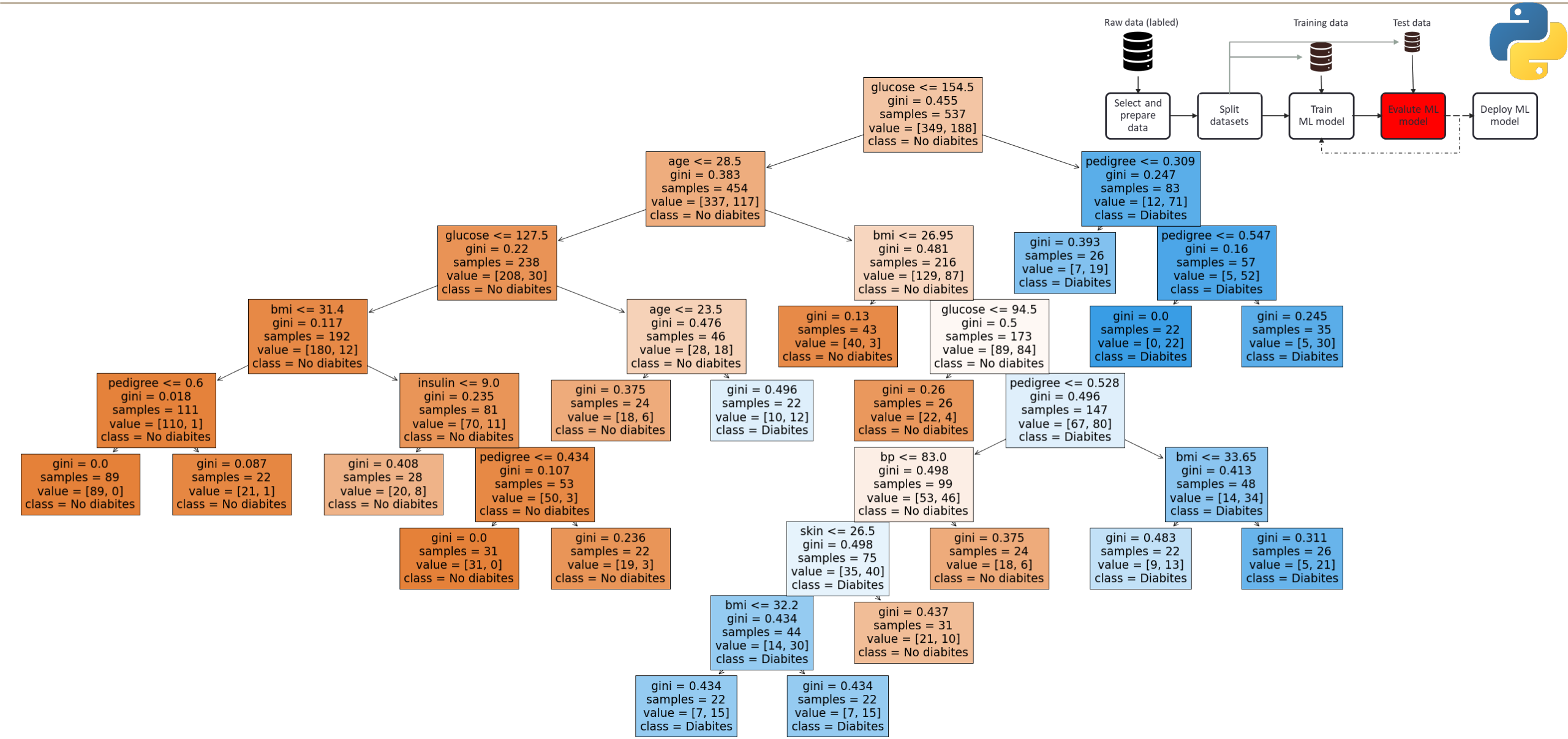


Evaluate Decision tree model (training data)

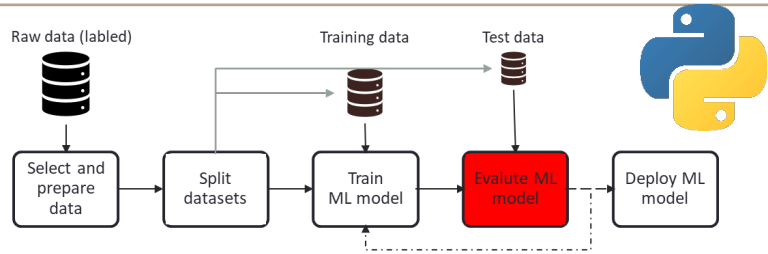
Visualization the decision tree

```
import matplotlib.pyplot as plt
from sklearn import tree
plt.figure(figsize=(40,20))  # customize according to the size of your tree
_ = tree.plot_tree(clf_tree,feature_names = X_train.columns, filled=True, class_names= ["No diabites","Diabites"])
plt.show()
```

Decision Trees in Python

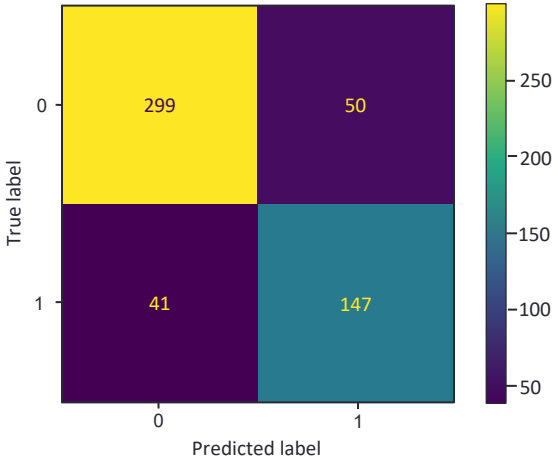


Decision Trees in Python



Confusion matrix (training data)

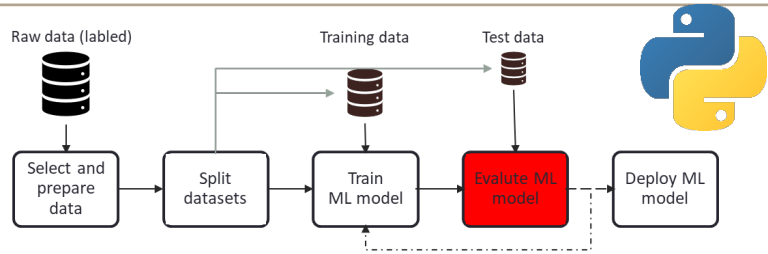
```
# Import the confusion matrix
from sklearn.metrics import plot_confusion_matrix

# Plot Confusion matrix
plot_confusion_matrix(clf_tree , X_train, y_train, values_format='d')
```




```
import numpy as np
# Predict the response for training dataset
y_pred_train = clf_tree.predict(X_train)
np.sum(np.equal(y_train, y_pred_train)) / len(y_train)
```

```
0.8305400372439479
```

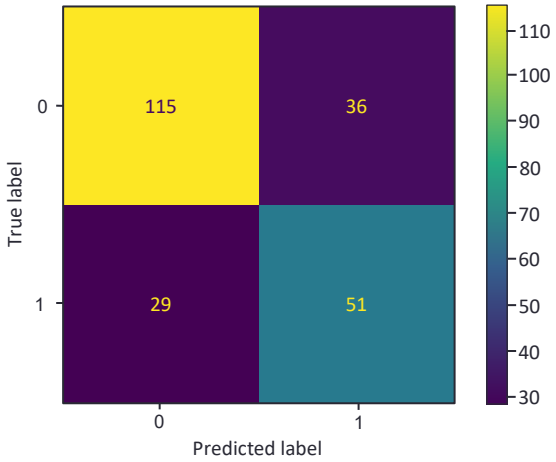
Decision Trees in Python



Evaluate Decision tree model (test data)

Confusion matrix (test data)

```
# Predict the response for test dataset
y_pred_test = clf_tree.predict(X_test)

# Import the confusion matrix
from sklearn.metrics import plot_confusion_matrix

# Plot Confusion matrix
plot_confusion_matrix(clf_tree , X_test, y_test, values_format='d')
```




```
import numpy as np
np.sum(np.equal(y_test, y_pred_test)) / len(y_pred_test)
```

```
0.7186147186147186
```

Decision Trees

- Foundations of decision trees
- Splitting criteria for decision trees
- Tree pruning
- Summary
- Python example
- **Hyperparameter for decision trees**

Hyperparameter for decision trees

- Gini Impurity or Entropy?
  - By default, the Gini impurity measure is used, but you can select the entropy impurity measure instead of by setting the criterion hyperparameter to "entropy".
  - Gini impurity is slightly faster to compute, so it is a good default.
  - However, when they differ, Gini impurity tends to isolate the most frequent class in its branch of the tree, while entropy tends to produce slightly more balanced trees.

- Regularization Hyperparameters
  - Decision Trees make very few assumptions about the training data (as opposed to linear models, which assume that the data is linear, for example)
  - The tree structure will adapt itself to the training data, fitting it very closely → tend to overfit
  - To avoid overfitting the training data, you need to restrict the Decision Tree's freedom during training

- Example Hyperparameter in Scikit-Learn
  - max_depth: restrict the maximum depth of the Decision Tree
  - 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
  - max_leaf_nodes: the maximum number of leaf nodes

You can learn all this in an interactive online course!

https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/classification-and-regression-trees?ex=1

Decision-Tree for Classification

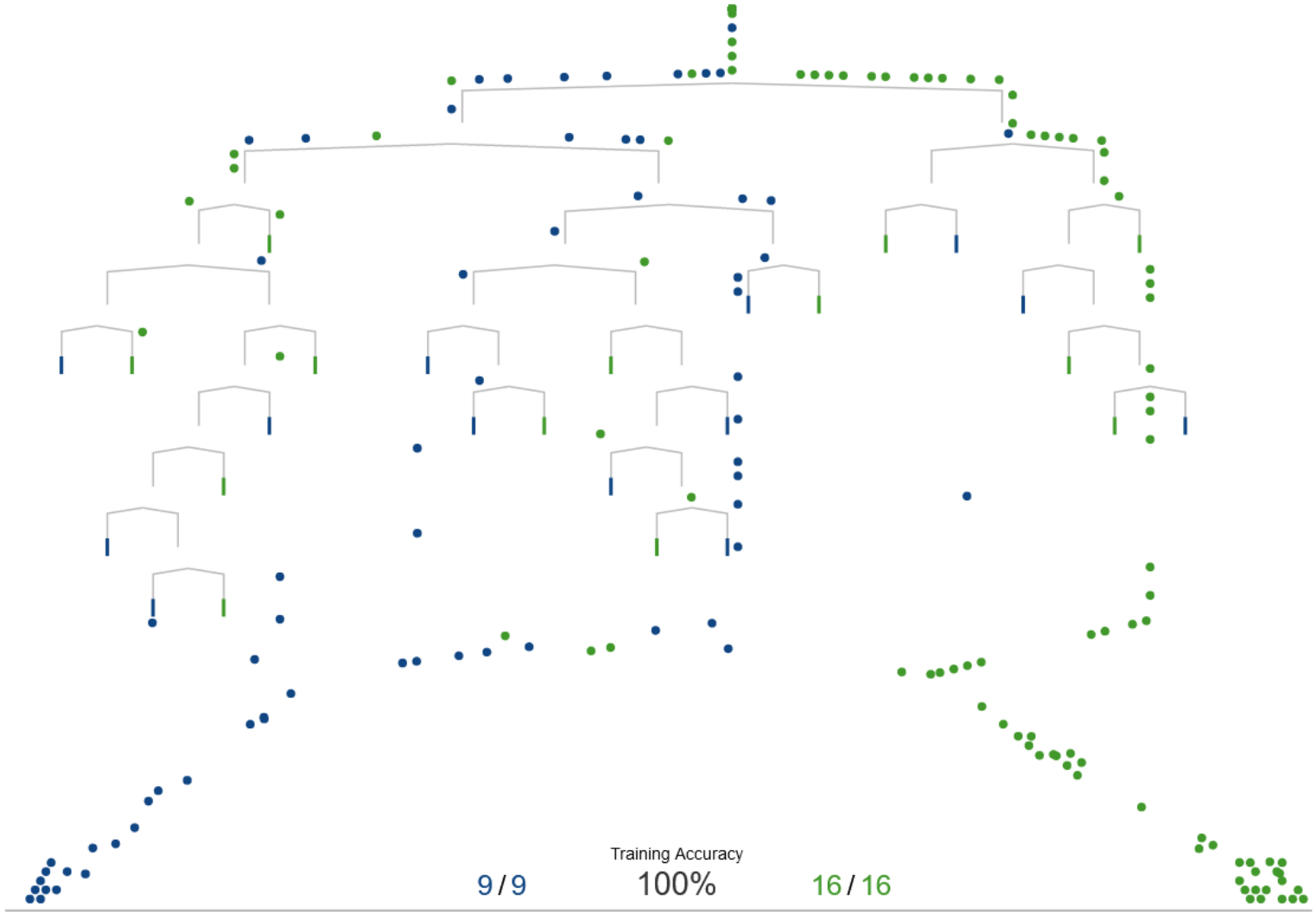
MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

datacamp

Making predictions

The newly-trained decision tree model determines whether a home is in San Francisco or New York by running each data point through the branches.



Here you can see the data that was used to train the tree flow through the tree.

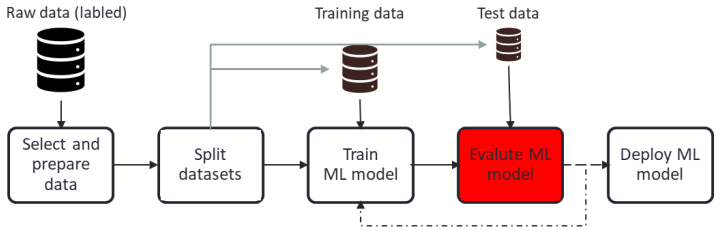
This data is called **training data** because it was used to train the model.

An intuitive visualization of decision trees: http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

Classification

- Introduction
- Decision Trees
- **Classifier Evaluation**
- k-Nearest Neighbors (kNN)
- Naive Bayes
- Logistic Regression
- Advanced Classifier Evaluation

Classifier Evaluation



- The classifier is constructed based on the training set data records.
- Thus, the question arises:
  - How powerful is the ("learned") classifier generated from this training data for data sets with unknown class membership?
- Classifier evaluation process includes:
  - Examination of the model on a test data set
  - Comparison of the actual and the classified classes of the test data. The error rate that occurs here reflects the classifier's accuracy.
  - Use of quality measures to determine the suitability of the generated decision tree

The confusion matrix

- A confusion matrix is an approach that gives a brief statement of prediction results on a binary and multi-class classification problem.
  - After the classification is done, the confusion matrix can tell us how well a classifier recognizes class membership
  - The confusion matrix is created by comparing the prediction results of the target class with the true/actual/reference values of the target class of the test set
  - The concept behind the confusion matrix is to find the number of right predictions and mistaken predictions, which are further summarized and separated into each class.

	Predicted (YES)	Predicted (YES)
Actual (YES)	**TP = 500**	**FN = 25**
Actual (NO)	**FP = 50**	**TN = 250**

The confusion matrix

- Basic terminology of the confusion matrix:
  - True positives (TP): truly positive observations and that also got classified as positive
  - False positives (FP): truly positive observations, but that got classified as negative
  - True negatives (TN): truly negative observations and that also got classified as negative
  - False negatives (FN): truly negative observations, but that got classified as positive

- The false positives (FP) and the false negatives (FN) can be further interpreted
  - False positives (FP): false alarms. Classify someone as ill, although she is *not*. **Type I error**
  - False negatives (FN): missed alarms. Someone was classified as not ill, although she *is* ill. **Type II error.**

	Predicted (YES)	Predicted (YES)
Actual (YES)	**TP = 500**	**FN = 25**
Actual (NO)	**FP = 50**	**TN = 250**

The confusion matrix

- The following phenomenon has been observed very often in empirical studies:
  - *An improvement in the misclassification rate on the training set is initially accompanied by its improvement on the test set. At a certain point, however, the misclassification rate on the test set increases again.*
- This phenomenon is known as overfitting.
- Possible reasons
  - erroneous test data (noise)
  - the machine learning model perfectly remembers the training data
  - too small data set to learn
- In practice, overfitting is a problem that should not be underestimated.

The confusion matrix: most common classifier performance indices

Measure	Description	Formula
Sensitivity, **Recall** or True Positive Rate TPR	„Ability to detect positives“, measure for completeness	TPR=TP/(TP+FN)=TP/P
Specificity or True Negative Rate TRN	„Ability to detect negatives“	TNR=TN/(FP+TN)=TN/N
Precision or Positive Predictive Value	„Actual positives among positively tested“, measure for exactness	PPV=TP/(TP+FP)
Negative Predictive Value	„Actual negatives among negatively tested“	NPV=TN/(TN+FN)
Accuracy, Recognition Rate	„Rate of correct decisions“	ACC=(TP+TN)/(P+N)
Error Rate, Misclassification Rate	“Rate of false classifications”	ERR=(FP+FN)/(P+N)
F, F1, **F-Score**	“Trade-off between recall and precision”, harmonic mean between precision and recall	F1=2TP/(2TP+FP+FN)

The confusion matrix: most common classifier performance indices

- Accuracy
  - tells us how accurate our predictive model is
  - ACC=(TP+TN)/(P+N) = 0.91

	Predicted (YES)	Predicted (YES)	
Actual (YES)	TP = 500	FN = 25	Recall $=\frac{TP}{TP+FN}$ $=\frac{500}{500+25}$ $=0.9524$
Actual (NO)	FP = 50	TN = 250	Specificity $=\frac{TN}{TN+FP}$ $=\frac{250}{250+50}$ $=0.8333$
	Precision $=\frac{TP}{TP+FP}$ $=\frac{500}{500+50}$ $=0.9091$	Negative Predictions $=\frac{TN}{TN+FN}$ $=\frac{250}{250+25}$ $=0.9091$	Total = 825

The confusion matrix: most common classifier performance indices

- F, F1, **F-Score**
  - F-measure is considered as one of the better ways to assess the model.
  - In lots of areas of data science, competition model performance is assessed using F-measure.
  - It is a harmonic mean of precision and recall.
  - The higher the value of the F1-score, the better the model is considered.
  - F1-score provides equal weightage to precision and recall, which means it indicates a balance between both
  - F1=2TP/(2TP+FP+FN)=0.93
  - One drawback of F-measure is that it assigns equal weightage to precision and recall, but in some examples, one needs to be higher than the other, which is why the F1-score may not be an exact metric.

	Predicted (YES)	Predicted (YES)	
Actual (YES)	**TP = 500**	**FN = 25**	**Recall** $=\frac{TP}{TP+FN}$ $=\frac{500}{500+25}$ $=0.9524$
Actual (NO)	**FP = 50**	**TN = 250**	**Specificity** $=\frac{TN}{TN+FP}$ $=\frac{250}{250+50}$ $=0.8333$
	Precision $=\frac{TP}{TP+FP}$ $=\frac{500}{500+50}$ $=0.9091$	**Negative Predictions** $=\frac{TN}{TN+FN}$ $=\frac{250}{250+25}$ $=0.9091$	**Total = 825**

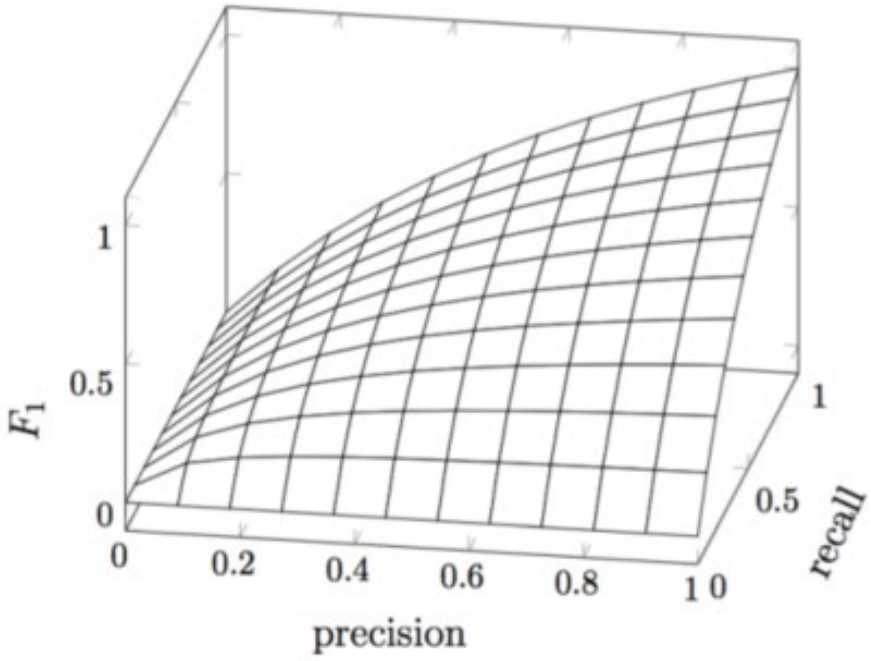
The confusion matrix: the F1 score

- The F1 score is a trade-off between precision and recall
  - It is the harmonic mean of precision and recall
  - F1 ∈ [0;1], with 1 being the optimal case

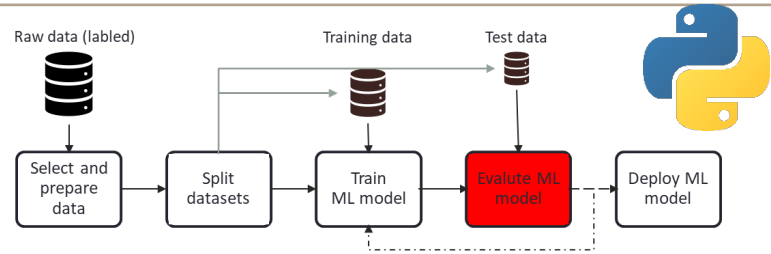
- To weight either precision or recall, the parameter β can be introduced

$$F_\beta = (1 + \beta^2) \frac{precision * recall}{(\beta^2 * precision) + recall}$$

- The F1 score is especially useful for imbalanced classes
  - Accuracy is not a useful measure there



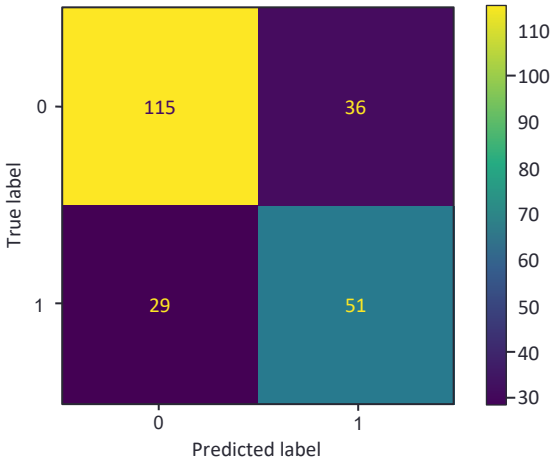
Evaluation of decision tree model



Evaluate Decision tree model (test data)

Confusion matrix (test data)

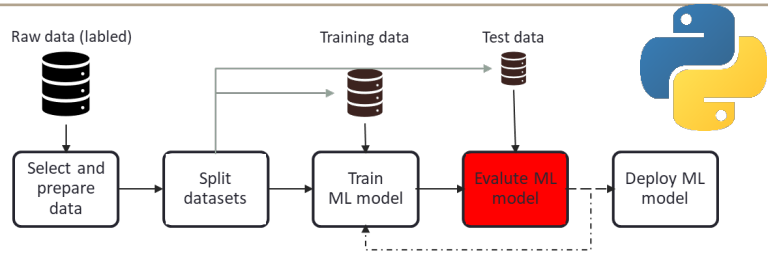
```
# Predict the response for test dataset
y_pred_test = clf_tree.predict(X_test)

# Import the confusion matrix
from sklearn.metrics import plot_confusion_matrix

# Plot Confusion matrix
plot_confusion_matrix(clf_tree , X_test, y_test, values_format='d')
```




```
import numpy as np
np.sum(np.equal(y_test, y_pred_test)) / len(y_pred_test)
```

```
0.7186147186147186
```

Evaluation of decision tree model



Performance metrics

```
# Import metrics module for performance evaluation
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
# Calculate model accuracy
print("Accuracy:",accuracy_score(y_test, y_pred_test))
# Calculate model precision
print("Precision:",precision_score(y_test, y_pred_test))
# Calculate model recall
print("Recall:",recall_score(y_test, y_pred_test))
# Calculate model f1 score
print("F1-Score:",f1_score(y_test, y_pred_test))
```

```
Accuracy: 0.7186147186147186
Precision: 0.5862068965517241
Recall: 0.6375
F1-Score: 0.6107784431137724
```

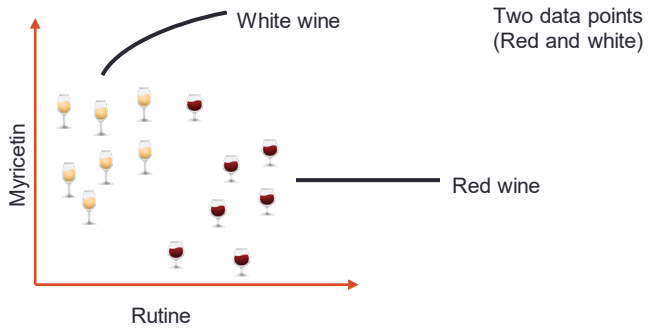
Classification

- Introduction
- Decision Trees
- Classifier Evaluation
- **k-Nearest Neighbors (kNN)**
- Naive Bayes
- Logistic Regression
- Advanced Classifier Evaluation

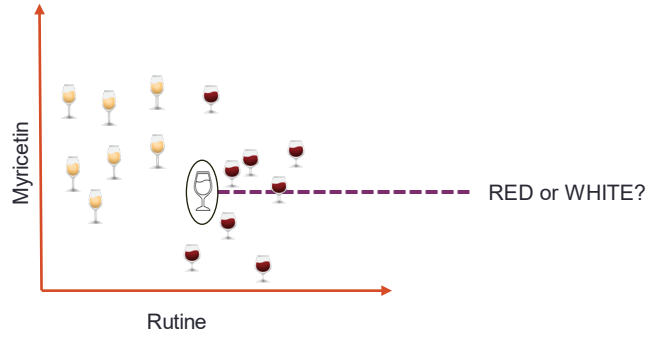
K-nearest neighbors

- Characteristics
  - kNN = k-Nearest Neighbors
  - Used since the beginning of the 1970s
  - Part of supervised machine learning
  - One of the most used and popular machine learning algorithms
  - It is mainly based on feature similarity. KNN checks how similar a data point is to its neighbor and classifies the data point into the class it is most similar to
  - Non-parametric
    - The algorithm is distribution-free and there is no need for parameters such as mean and standard deviation.
  - Lazy learning algorithm
    - KNN does not train the model; that is, the model is trained in the testing phase
    - This makes for faster training but slower testing
    - It is also more time- and memory-consuming
  - Instance-based learning
    - the predicted outcome is based on the similarity with its nearest neighbors.
    - kNN does not create any abstract equations or rules for prediction; instead, it stores all the data and queries each record
- Purpose
  - Using a database in which the data points are separated into some classes to predict the classification of a new sample point.

- Example: Wine
  - Rutime and Myricetin are two components in wine
  - The following graph reflects the level of Rutine and Myricetin – depending on white and red wine

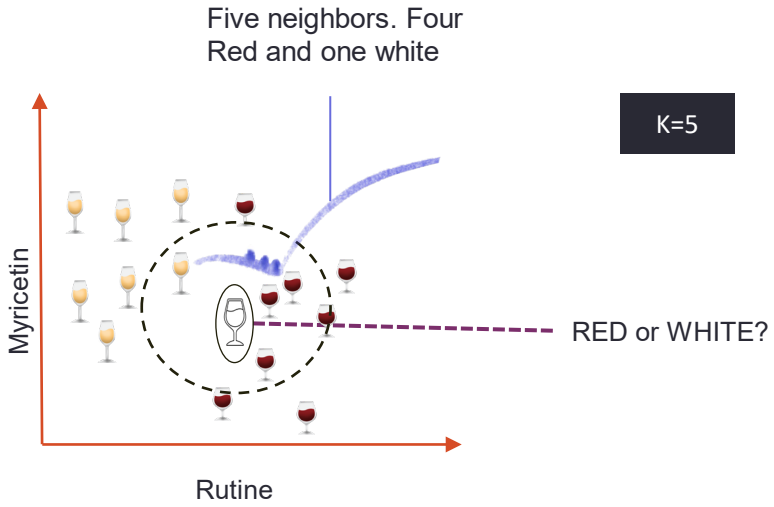


  - So, if we now include an additional glass of wine in the dataset, for which we do not know whether it is red or white wine – what would you suggest? Is the new wine red or white?
    [So far, we know the level of Myricetin and Rutine of the new wine and the levels of previous ones as well as whether these are red or white]



Note:
While you might also use n-dimensions, we bring an example with two dimensions as this is easier to illustrate.

- Example: Wine
  - In order to determine whether the wine is red or white, we…
    - … aim to find out what the neighbors are in this case.
    - … among others, we say k = 5, and the new data point is classified by the majority of votes from its five neighbors
    - … based on that, the new point would be classified as red since four out of five neighbors are red.



- Consequences of that example:
  - Which steps are relevant to find the closest k values
    - → we explain the algorithm on the next page
  - We see the importance of k -> how to select that value
    - → we present some insights on that topic on the following pages

Note:
This example is easy to understand and illustrates selecting the k-nearest neighbors by hand. It is useful to understand crucial issues. Based on that, we explain the algorithm and related information on how to choose k on the next pages.

K-nearest neighbors: Algorithm

- Algorithm
  - Step 1: Prepare your data
  - Step 2: Initialize k to your chosen number of neighbors (see the following page explaining how to choose k)
  - Step 3: For each example in the data
    - Calculate the distance between the query example and the current example from the data.
    - Add the distance and the index of the example to an ordered collection
  - Step 4: Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
  - Step 5: Pick the first k entries from the sorted collection
  - Step 6: Get the labels of the selected k entries
  - Step 7: Return the mode of the k labels

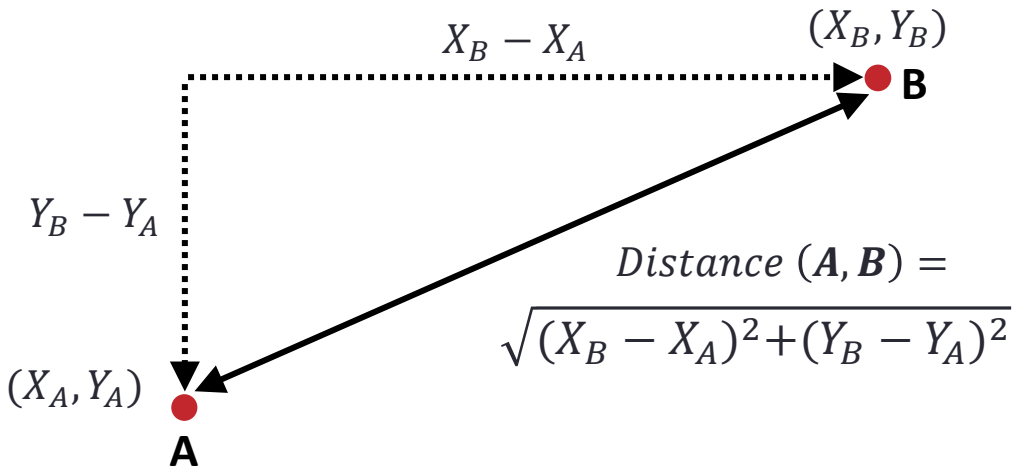
K-nearest neighbors: Hyperparamter tuning - k

- The ‘k’ in the kNN algorithm is based on feature similarity.
- So, choosing the right value of k (called hyperparameter tuning) is complex and crucial for better accuracy.
- Some ideas on picking a value for ‘k’:
  - There is no best way (e.g., physical or biological) to determine and identify the best value for k
    - it is more or less to trial-and-error and test some values before settling on one.
  - When choosing small values for k, noise might affect the result, so you are subject to the effects of outliers.
    - Decreasing the value of k to 1 means that predictions become less stable.
  - When choosing large values for k, you will have a smoother decision boundary (lower variance) but increased bias.
  - Practice suggest choosing the value of k is k = sqrt(N) (N stands for the number of samples in your training dataset)
    - Sqrt = square root
  - Keep the value of k odd (so you avoid confusion between two classes of data)
  - To select the k that’s right for your data, we run the kNN algorithm several times with different values of k and choose the k that reduces the number of errors we encounter while maintaining the algorithm’s ability to accurately make predictions when it’s given data it hasn’t seen before.

K-nearest neighbors: Hyperparamter tuning - distance

- Beyond picking a value for ‘k’, it is important to know how to calculate the distance between two data
- There are different possibilities.

- Euclidean Distance (most often used and relevant for that course)



- Further possibilities
  - $d_{Manhattan}(\boldsymbol{X}, \boldsymbol{Y}) = \|\boldsymbol{X} - \boldsymbol{Y}\|_1 = |x_1 - y_1| + |x_2 - y_2| + \cdots$
  - $d_{Jaccard}(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$
  - $d_{Cosine}(\boldsymbol{X}, \boldsymbol{Y}) = 1 - \frac{\boldsymbol{X} \cdot \boldsymbol{Y}}{\|\boldsymbol{X}\|_2 \cdot \|\boldsymbol{Y}\|_2}$

K-nearest neighbors: distance – scale issues

- Problem: Data on different scales can cause undesirable results when using distances
- Example:
  - Consider a data set containing two features: age, and income.
    - Age ranges from 0–100,
    - Income ranges from 0 to 1 Mio.
  - So, these two features are in very different ranges
  - Income is about 10,000 times larger than age
  - Many classifiers (like here KNN) calculate the distance between two points by the Euclidean distance.
  - If one of the features has a broad range of values, the distance will be governed by this particular feature.
    → it will influence the results more, even if its not necessarily more important

- Solution: using Feature Scaling
  - A technique to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information.
  - This can be done using two common techniques:
    - Min-max normalization
    - Z-score normalization

$$z = \frac{x - \min(x)}{\max(x) - \min(x)}$$

$$z = \frac{x - \mu}{\sigma}$$

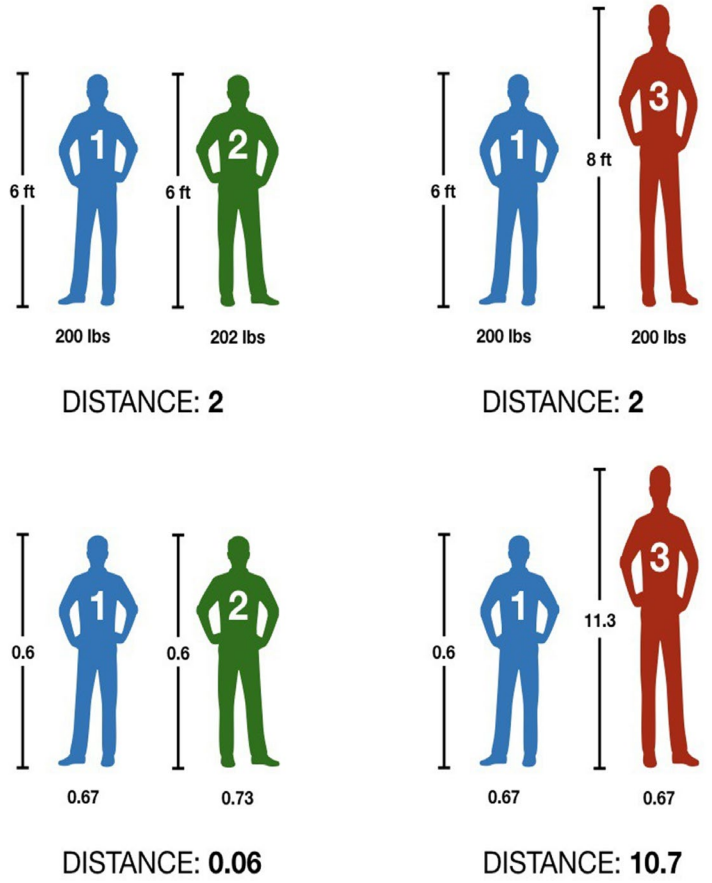
$\mu = \text{Mean}$

$\sigma = \text{Standard Deviation}$

K-nearest neighbors: distance – scale issues

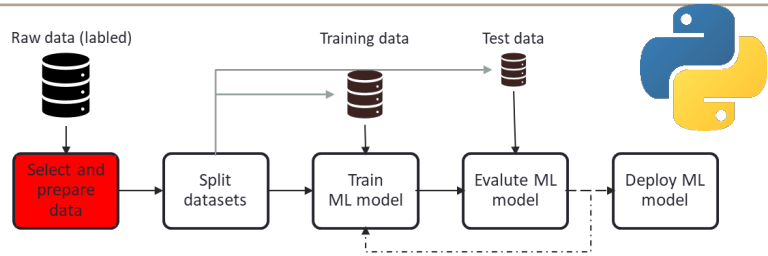
- Problem: Data on different scales can cause undesirable results when using distances
- Solution: Scale data so that features have same mean and standard deviation (standardization of data)
  1) Subtract mean of a feature from all observations
  2) Divide each feature by the standard deviation of the feature
     → Normalized features have a mean of zero and a standard deviation of one

- Example
  - Distance of two persons (first figure):
    - Person 1 (200 lbs) vs. person 2 (202 lbs) → distance 2
    - Person 1 (6 ft) vs. person 3 (8ft) → distance 2
    - Same distance but not comparable because of differ scales
  - Sale data by dividing by the standard deviation
    - Person 1 (0.57) vs. person 2 (0.73) → distance 0.06
    - Person 1 (0.67) vs. person (10.7) → distance 10.7



K-nearest neighbors: summary

- Evaluation of kNN
  - Pros of kNN
    - Easy and simple to implement
    - Flexible to distance choices
    - Can do well in practice with enough representative data
    - No need to build a model, tune several parameters, or make additional assumptions
    - The algorithm is versatile
    - It can be used for classification but also other purposes, including regression and search
  - Cons of kNN
    - No clear guidance of how to determine the value of parameter k
    - Computation costs are relatively high (distance of each query instance to all training samples must be calculated)
    - Data storage
    - The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase



K-Nearest Neighbours Classification

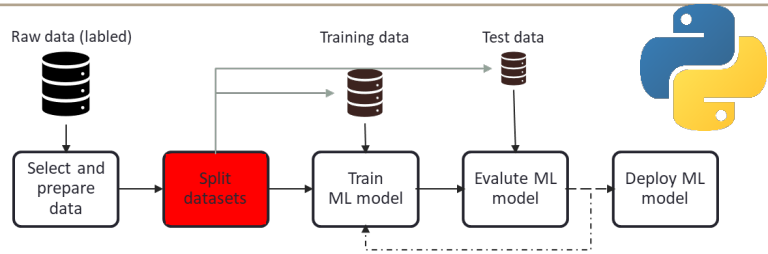
Select and prepare data

```
# Import libraries
import pandas as pd

# read the dataset
diabetes = pd.read_csv("diabetes.csv")

# Show top 5-records
diabetes.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# Splitting dataset in two parts: feature set and target label
feature_set = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree','skin']

X = diabetes[feature_set]
y = diabetes.label
```

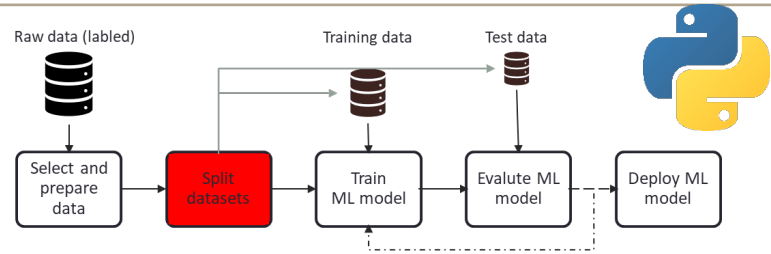
kNN model



Split dataset

```
# Partition data into training and testing set
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
```

kNN model

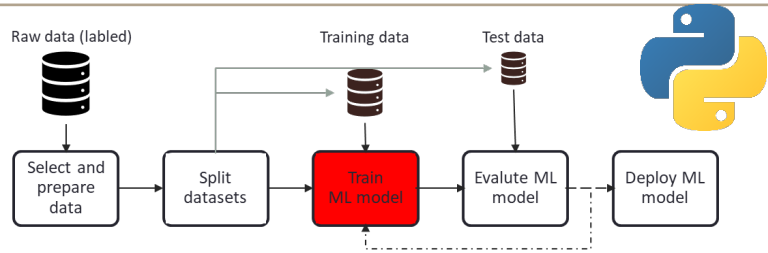



```
from sklearn.preprocessing import StandardScaler
# scale the data
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
pd.DataFrame(X_train_scaled)
```

	0	1	2	3	4	5	6	7
0	-0.836294	-0.189732	-1.060153	-0.948610	-0.800051	-0.535764	-0.614216	-0.157146
1	0.390728	2.130203	0.646467	-0.434667	-0.490543	0.128044	-0.909738	0.553619
2	-1.143050	1.478536	1.355371	-0.777296	0.437979	-0.093226	-0.306991	1.393614
3	0.083972	0.748669	0.147609	-0.434667	0.314176	-0.093226	-0.906812	0.036699
4	-0.836294	0.027491	1.486650	-0.006380	-0.552445	-2.195284	-0.839515	1.135154
...	...	...	...	...	...	...	...	...
532	0.390728	0.522758	-0.443144	-0.605981	0.561782	-0.314495	-0.172397	0.941309
533	-0.836294	-0.693688	-1.257071	-0.520324	-0.769100	2.893910	-0.769292	-1.320215
534	1.924505	-0.693688	1.788590	0.421906	-0.614346	0.902486	1.948921	1.070539
535	-1.143050	-0.693688	1.368499	-0.349009	0.623683	-3.854804	-0.775144	-1.320215
536	-1.143050	-0.693688	-1.243943	-1.034268	0.128472	1.455659	-0.608364	-1.320215

- **Scaling the data**
  - The StandardScaler uses z-score normalization
  - We fit the StandardScaler only with training data, so that the test data does not influence the StandardScaler. This means the StandardScaler learns the z-score normalization only based on the training data and the kNN model is not influence by the test data
  - The test data is then scaled based on the z-score normalization of the training data.
  - https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

kNN model

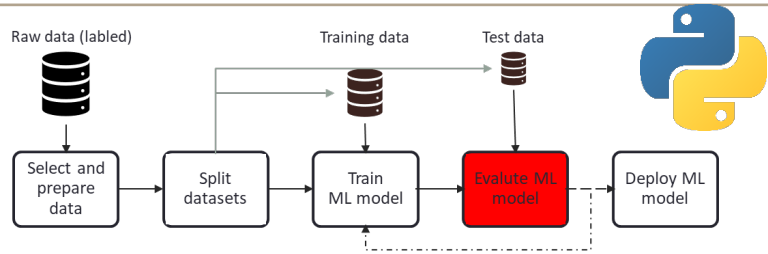


Train kNN model

```
# Import KNN model
from sklearn.neighbors import KNeighborsClassifier

# Create a KNN classifier object
clf_kNN = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training dataset
clf_kNN.fit(X_train_scaled,y_train)
```

```
KNeighborsClassifier(n_neighbors = 3)
```

Evaluation of the kNN model

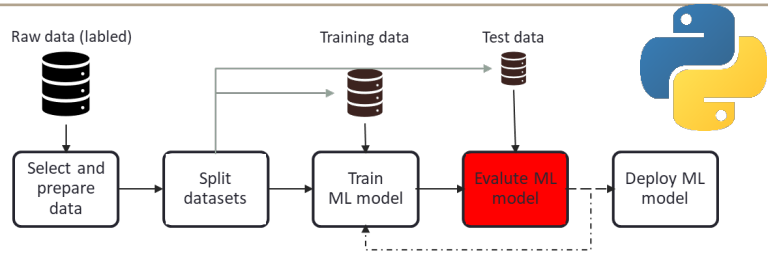


Evaluate kNN model (training dataset)

```
# Predict the y variable for train dataset
y_pred_train_knn = clf_kNN.predict(X_train_scaled)

# Import metrics module for performance evaluation
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# Calculate model accuracy
print("Accuracy:",accuracy_score(y_train, y_pred_train_knn))
# Calculate model precision
print("Precision:",precision_score(y_train, y_pred_train_knn))
# Calculate model recall
print("Recall:",recall_score(y_train, y_pred_train_knn))
# Calculate model f1 score
print("F1-Score:",f1_score(y_train, y_pred_train_knn))
```

```
Accuracy: 0.8584729981378026
Precision: 0.8294117647058824
Recall: 0.75
F1-Score: 0.7877094972067038
```

Evaluation of the kNN model



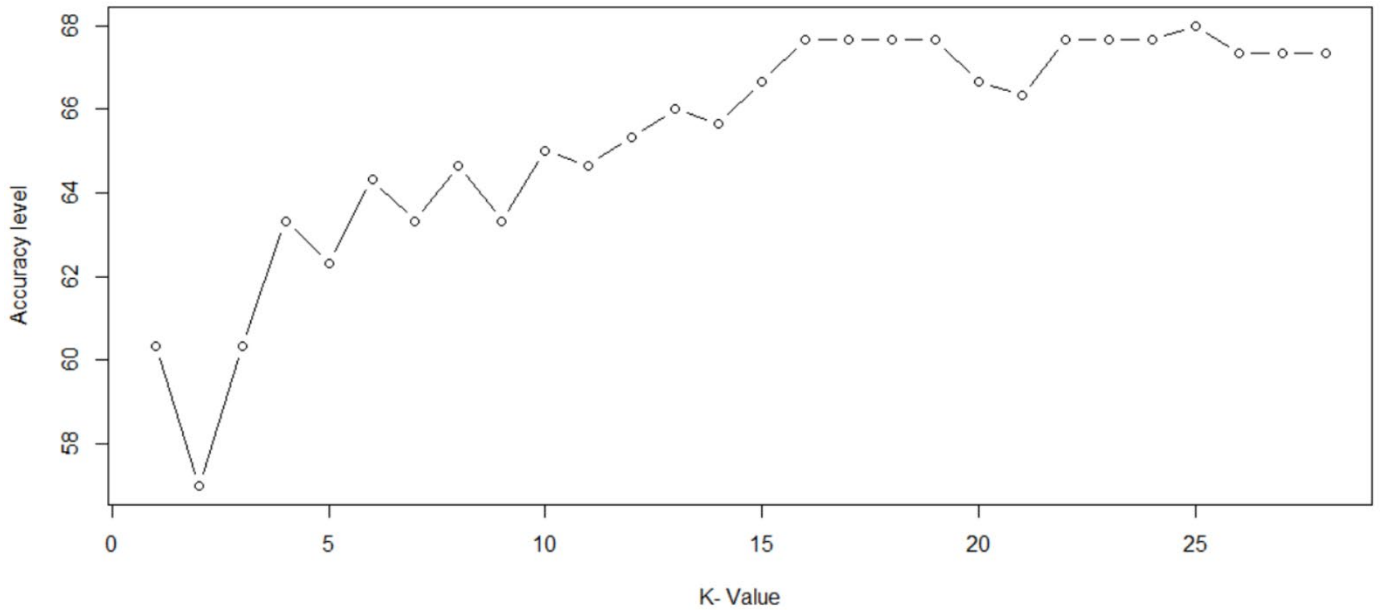
Evaluate kNN model (test dataset)

```
# Predict the y variable for test dataset
y_pred_test_knn = clf_kNN.predict(X_test_scaled)

# Import metrics module for performance evaluation
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# Calculate model accuracy
print("Accuracy:",accuracy_score(y_test, y_pred_test_knn))
# Calculate model precision
print("Precision:",precision_score(y_test, y_pred_test_knn))
# Calculate model recall
print("Recall:",recall_score(y_test, y_pred_test_knn))
# Calculate model f1 score
print("F1-Score:",f1_score(y_test, y_pred_test_knn))
```

```
Accuracy: 0.70995670995671
Precision: 0. 5970149253731343
Recall: 0.5
F1-Score: 0.5442176870748299
```

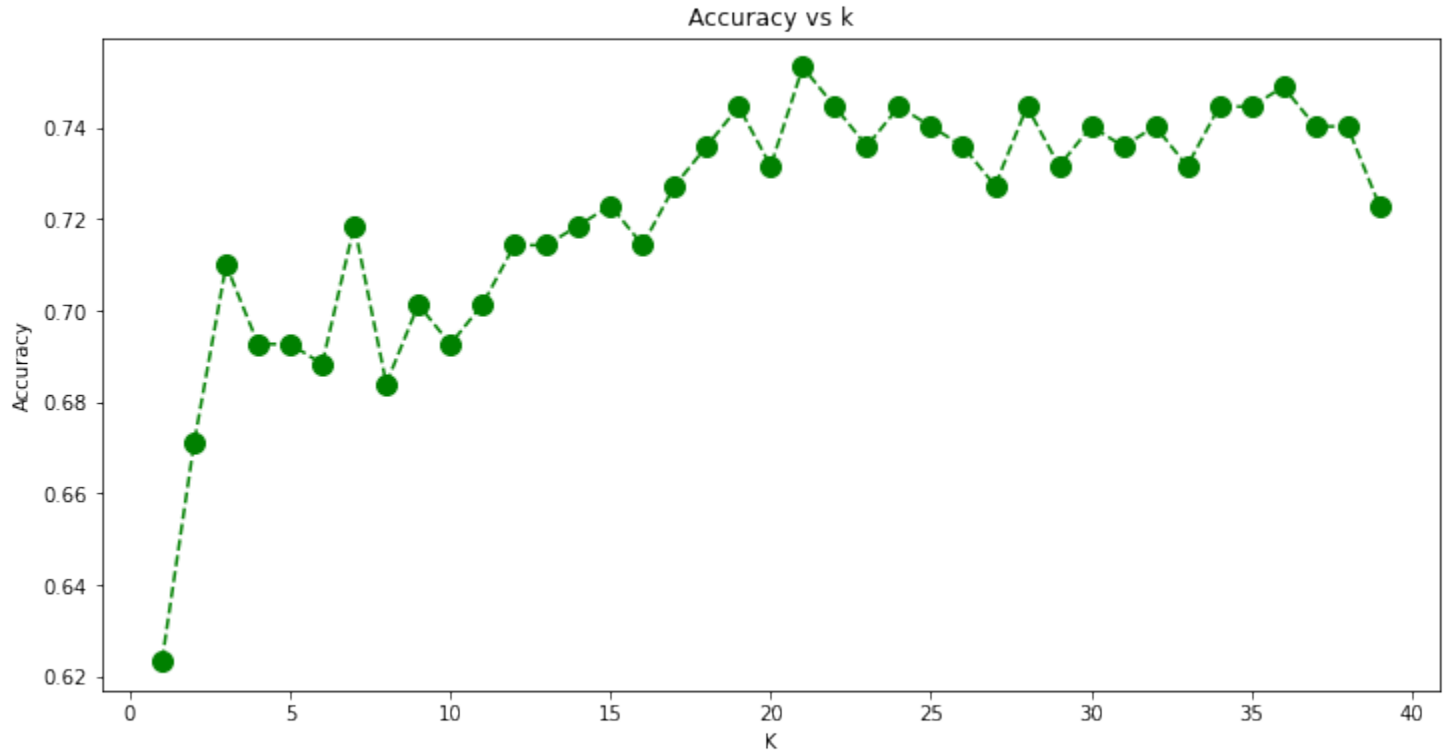
K-nearest neighbors

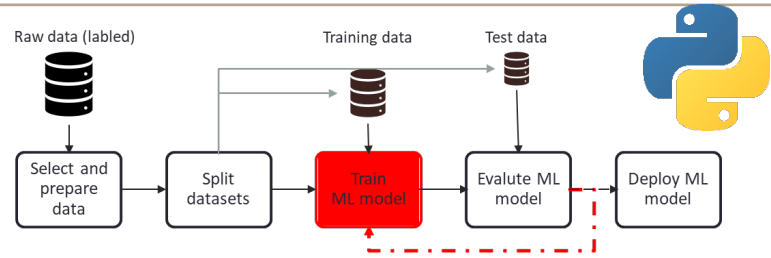
- The right number for k
  - k is often set to the square root of the number of observations in the data set
  - It is important to try different numbers for k to test whether the results get better
  - For this, a for-loop structure can be used and be plotted



K-nearest neighbors


```
accuracy = []

# Calculating error for K values between 1 and 40
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train_scaled, y_train)
    pred_i = knn.predict(X_test_scaled)
    accuracy.append(accuracy_score(y_test, pred_i))
```

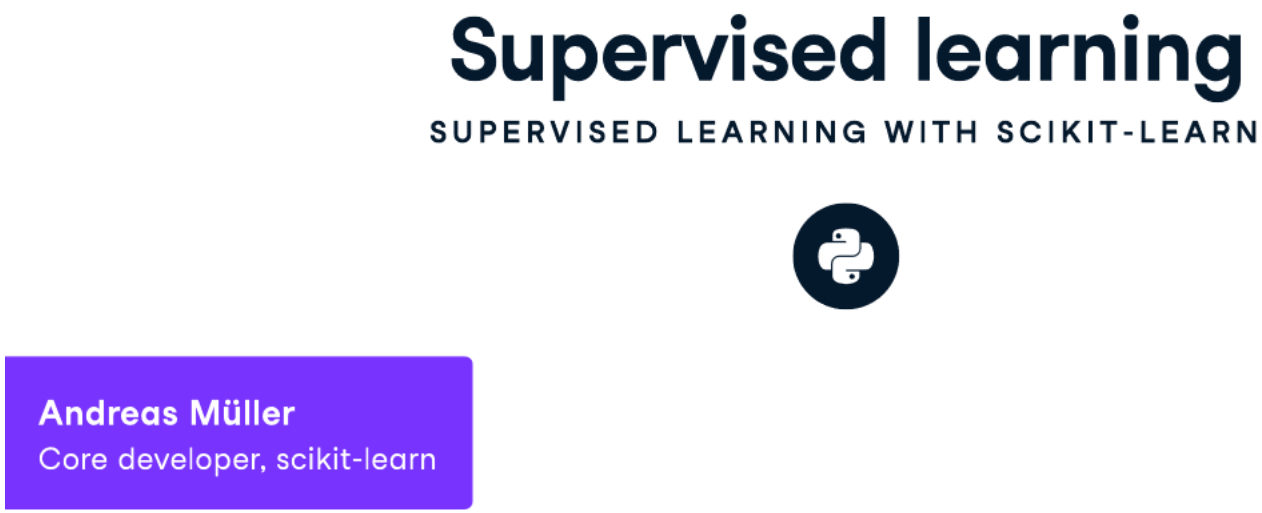




- **The right number for k**
  - One way to find the right number for k is to loop through different k values and compare the performance metric (e.g., accuracy)
  - More advanced methods are discussed in the next lecture

You can learn all this in an interactive online course!

https://campus.datacamp.com/courses/supervised-learning-with-scikit-learn/classification?ex=1



Classification

- Introduction
- Decision Trees
- Classifier Evaluation
- k-Nearest Neighbors (kNN)
- **Naive Bayes**
- Logistic Regression
- Advanced Classifier Evaluation

Naive Bayes: Overview

- Characteristics
  - Simple technique for constructing classifiers
  - Based on the Bayes Theorem for calculating probabilities and conditional probabilities
  - It includes an assumption of independence among predictors – and also that features are equal

- Purpose
  - Simply said a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature, and with that assumption, classifications are possible

- Application of Naive Bayes in real-life situations:
  - Real time Prediction
  - Multi class Prediction (classifying instances into one of three or more classes)
  - Text classification
  - Spam Filtering
  - Sentiment Analysis
  - Recommendation System

Naive Bayes: Example

- Example
  - There is a data set with data on 1,000 pieces of fruit.
  - The fruit being a Banana, Orange or some (unknown) Other fruit
  - Three features of each fruit are known
    - long,
    - Sweet
    - Yellow
  - Frequency Table:

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	**500**	**650**	**800**	**1000**

  - What we now know based on that is
    - 50% of the fruits are bananas;
    - 30% are oranges;
    - 20% are other fruits

    - From 500 bananas, 400 (0.8) are Long, 350 (0.7) are Sweet, and 450 (0.9) are Yellow
    - Out of 300 oranges, 0 are Long, 150 (0.5) are Sweet, and 300 (1) are Yellow
    - From the remaining 200 fruits, 100 (0.5) are Long, 150 (0.75) are Sweet, and 50 (0.25) are Yellow

Naive Bayes: Example

- Example
  - Let‘s say we are interested in predicting a class of fruit that is Long, Sweet, and Yellow
  - Based on that given information, we can classify it using the following formula:
    - Banana:

$$P(Banana|Long, Sweet, Yellow) =$$

$$= \frac{P(Long|Banana) \cdot (Sweet|Banana) \cdot (Yellow|Banana) \cdot P(Banana)}{P(Long) \cdot P(Sweet) \cdot P(Yellow)}$$

$$= \frac{0.8 \cdot 0.7 \cdot 0.9 \cdot 0.5}{0.5 \cdot 0.65 \cdot 0.8} = \frac{0.252}{0.5 \cdot 0.65 \cdot 0.8} \approx 0.97$$

    - Orange:

$$P(Orange|Long, Sweet, Yellow) = 0$$

    - Other fruit:

$$P(Other|Long, Sweet, Yellow) =$$

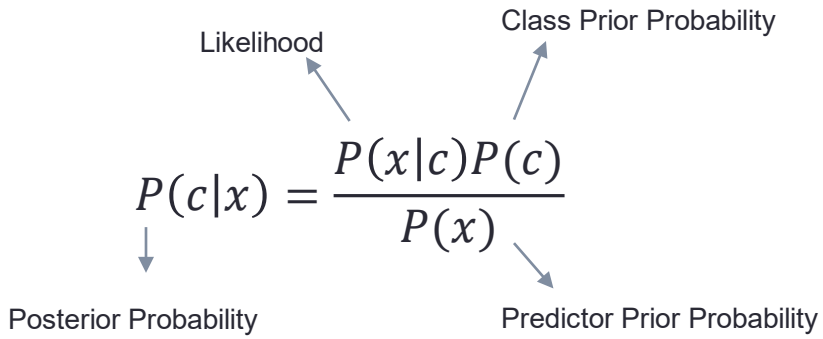
$$= \frac{P(Long|Other) \cdot (Sweet|Other) \cdot (Yellow|Other) \cdot P(Other)}{P(Long) \cdot P(Sweet) \cdot P(Yellow)}$$

$$= \frac{0.5 \cdot 0.75 \cdot 0.25 \cdot 0.2}{0.5 \cdot 0.65 \cdot 0.8} = \frac{0.01875}{0.5 \cdot 0.65 \cdot 0.8} \approx 0.07$$

  - The one with the highest probability being the winner is the banana!

Naive Bayes: Formula

- Formula
  - Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c).
  - The equation (known as Bayes' rule):



$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

  - So,
    - P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
    - P(c) is the prior probability of class.
    - P(x|c) is the likelihood which is the probability of predictor given class.
    - P(x) is the prior probability of predictor.

- Bayes' rule says: that we can compute the probability of our hypothesis given some evidence by looking at the probability of the evidence given the hypothesis and the unconditional probability of the hypothesis and the evidence.

- Note: Try to use the example from the previous page to classify P(c), P(x), …. To the numbers provided in the table or to the calculated ones P(c|x). Try to interpret them!

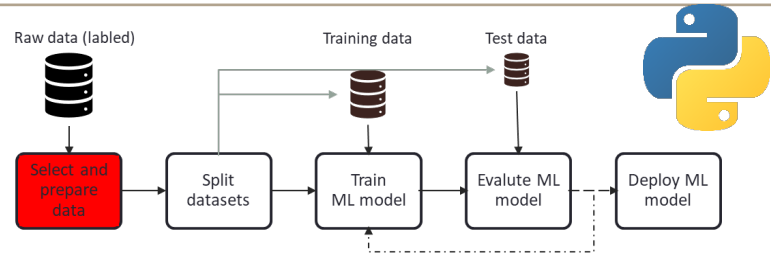
Naive Bayes: Algorithm

- Algorithm
  - Step 1: Use the data and convert them into a frequency table
  - Step 2: Calculate the likelihoods (e.g.by setting up a table)
  - Step 3: Use Naive Bayesian equation/formula to calculate the posterior probability for each class.
  - Step 4: The class with the highest posterior probability is the outcome of prediction.

Naive Bayes: Summary

- Evaluation of Naive Bayes
  - Pros
    - easy to build
    - particularly useful for huge data sets
    - Naive Bayes is known to outperform even highly sophisticated classification methods
    - Very fast
    - easily trained, even with a small dataset
    - not sensitive to irrelevant features
  - Cons
    - It assumes every feature is independent, which isn't always the case
      - is practically impossible for all the predictors to be fully independent
    - Zero frequency problem
      - means that if any category in the feature is missing, then it will have a zero frequency count
      - Can be solved by Laplacian correction. (or Laplace transformation)

Naive Bayes



Naive Bayes

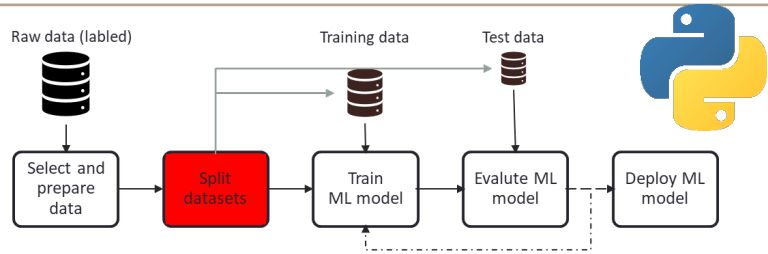
Select and prepare data

```
# Import libraries
import pandas as pd

# read the dataset
diabetes = pd.read_csv("diabetes.csv")

# Show top 5-records
diabetes.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# Splitting dataset in two parts: feature set and target label
feature_set = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree','skin']
X = diabetes[feature_set]
y = diabetes.label
```

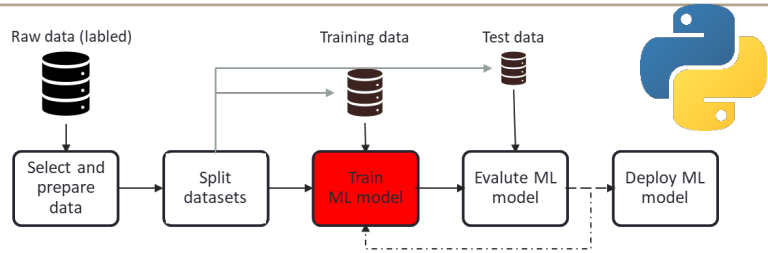
Naive Bayes



Split dataset

```
# Partition data into training and testing set
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Naive Bayes

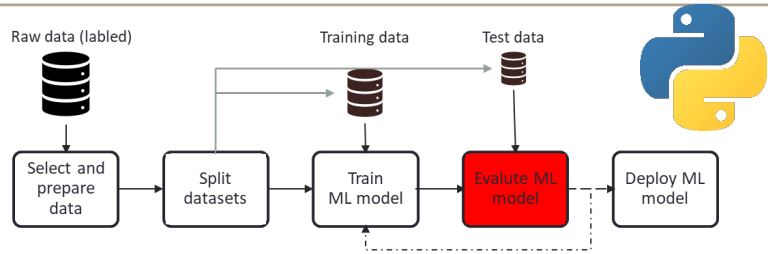


Train naive bayes model

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

# Create a Gaussian Classifier
clf_gnb = GaussianNB()

# Train the model using the training dataset
clf_gnb.fit(X_train,y_train)
```

```
GaussianNB()
```

Naive Bayes

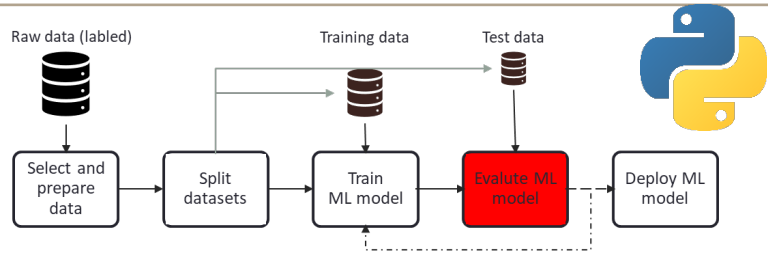


Evaluate kNN model (training data)

```
# Predict the y variable for train dataset
y_pred_train = clf_gnb.predict(X_train)

# Import metrics module for performance evaluation
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# Calculate model accuracy
print("Accuracy:",accuracy_score(y_train, y_pred_train))
# Calculate model precision
print("Precision:",precision_score(y_train, y_pred_train))
# Calculate model recall
print("Recall:",recall_score(y_train, y_pred_train))
# Calculate model f1 score
print("F1-Score:",f1_score(y_train, y_pred_train))
```

```
Accuracy: 0.7672253258845437
Precision: 0.7006369426751592
Recall: 0.5851063829787234
F1-Score: 0.6376811594202899
```

Naive Bayes



Evaluate kNN model (test data)

```
# Predict the y variable for test dataset
y_pred_train = clf_gnb.predict(X_test)

# Import metrics module for performance evaluation
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# Calculate model accuracy
print("Accuracy:",accuracy_score(y_test, y_pred_test))
# Calculate model precision
print("Precision:",precision_score(y_test, y_pred_test))
# Calculate model recall
print("Recall:",recall_score(y_test, y_pred_test))
# Calculate model f1 score
print("F1-Score:",f1_score(y_test, y_pred_test))
```

```
Accuracy: 0.7445887445887446
Precision: 0.6235294117647059
Recall: 0.6625
F1-Score: 0.6424242424242423
```

Classification

- Introduction
- Decision Trees
- Classifier Evaluation
- k-Nearest Neighbors (kNN)
- Naive Bayes
- **Logistic Regression**
- Advanced Classifier Evaluation

Logistic regression

- One specific type of regression, which can be used to classify elements is the so-called logistic regression

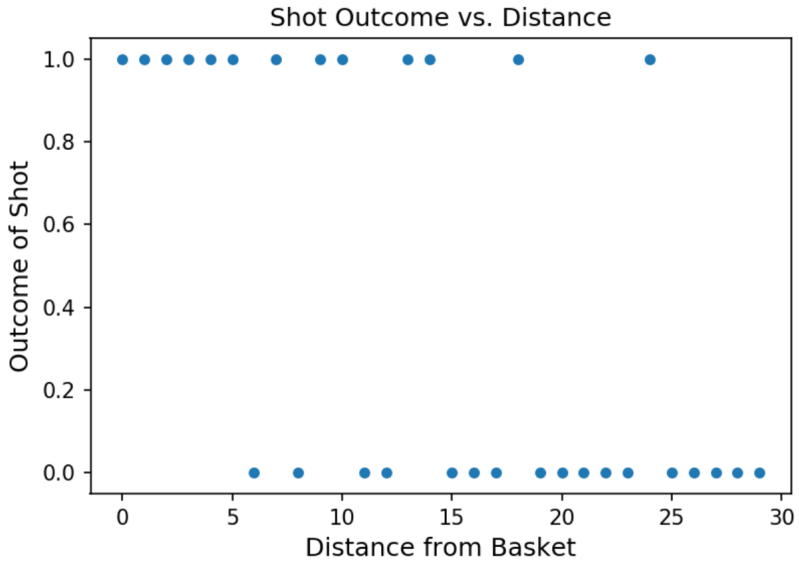
- Characteristics:
  - Developed by David Cox in 1958
  - Outcome is measured with a dichotomous variable (only two possible outcomes)
  - Outcomes might be: 1 / 0, Yes / No, True / False
  - It is as a special case of linear regression
  - It predicts the probability of occurrence of an event by fitting data to a **logit** function

- Purpose:
  - A statistical method for analyzing a dataset, in which there are one or more independent variables that determine a dichotomous outcome.

Example

- Objective:
  Examining the relationship between the basketball shooting accuracy and the distance from the basket [in feet].

- Data that are known:
  - 1 means make
  - 0 means miss



- **Interpretation**:
  - The further you get from the basket, the less accurately you shoot.
  - So, we can already see the rough outlines of our model: when given a small distance, it should predict a high probability, and when given a large distance, it should predict a low probability.

Example

- Logistic regression works a lot like linear regression
- So we start with the familiar linear regression equation

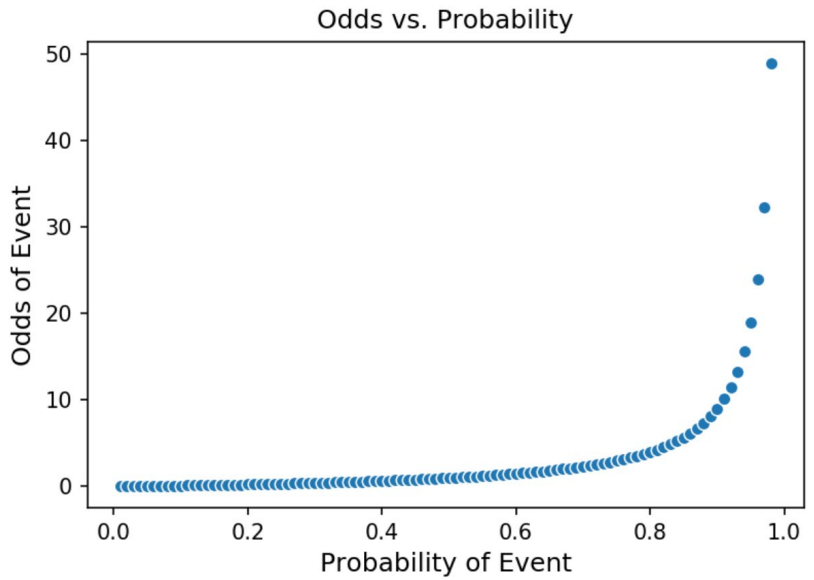
- In linear regression, the output is in the same units as the target variable.
- In logistic regression the output is in log odds.
  Odds is just another way of expressing the probability of an event, ***P(Event)***.

  Odds = P(Event) / [1-P(Event)]

- In our example, we now say that one shot 100 free throws and made 70.
  Based on this sample, the probability of making a free throw is 70%.
  So, the odds of making a free throw can be calculated as

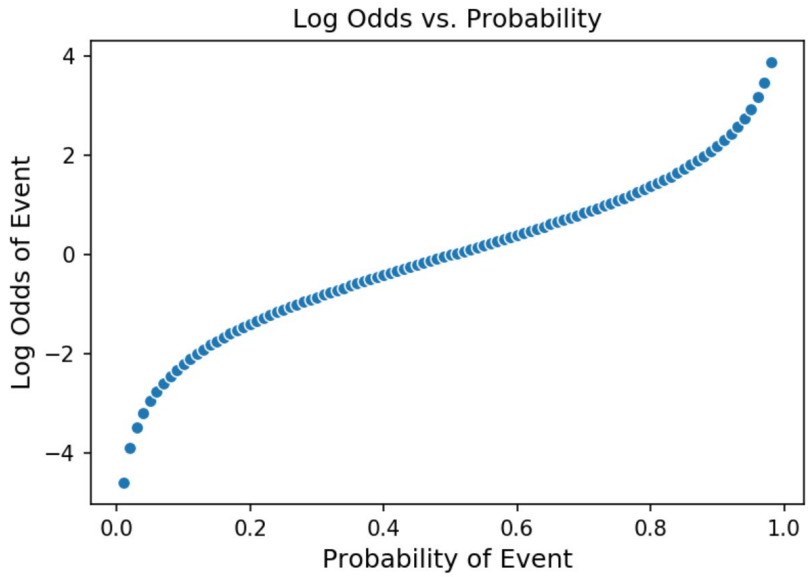
  Odds = 0.70 / (1–0.70) = 2.333

- So, we see that:
  Probabilities are bounded between 0 and 1 [this becomes a problem in regressio
  Odds range from 0 to infinity [see example]
  - E.g., see the relationship between odds and probability in the right picture



Example

- Taking the natural log of the odds provides a way to get log odds that are
  … unbounded [reflecting that they range from negative to positive infinity]
  … roughly linear across most probabilities!
- As we can estimate the log odds via logistic regression, we can estimate probability because log odds are just probability stated another way.



Notice that the middle section of the plot is linear

→ logistic regression equation:

Z = B0 + B1*distance_from_basket where Z = log(odds_of_making_shot)

Example

- And to get probability from ***Z,*** which is in log odds, we apply the sigmoid function. Applying the sigmoid function is a fancy way of describing the following transformation:

$$\textbf{Probability of making shot} = \frac{1}{1 + e^{-Z}}$$

- After understanding how we can go from a linear estimate of log odds to a probability, we aim to examine how the coefficients ***B0*** and ***B1*** are actually estimated in the logistic regression equation that we use to calculate ***Z***.
- This is, among others, known as a cost function and provides an explanation of how to do that.

Example

- The cost function
  - Logistic regression aims to minimize a cost function, which measures how wrong you are.
    - If a prediction was right, then there should be no cost,
    - if I am just a tiny bit wrong, there should be a small cost,
    - if I am massively wrong, there should be a high cost.

      → This is easy to visualize in the linear regression world where we have a continuous target variable (and we can square the difference between the actual outcome and our prediction to compute the contribution to cost of each prediction).

      → But what happens when we deal with a target variable that contains only 0s and 1s?

  - The example illustrated that the first shot was made from right underneath the basket.

    So: [Shot Outcome = 1 | Distance from Basket =0].
    How to translate this into a cost?
    - First, my model needs to spit out a probability. Let's say it estimates 0.95, which means it expects to hit 95% of shots from 0 feet.
    - There is only one shot from 0 feet in the actual data, and it made it, so the actual (sampled) accuracy from 0 feet is 100%.

  - So, the model was wrong because the answer according to our data was 100%, but it predicted 95%. But it was only slightly wrong, so we want to penalize it only a little bit:

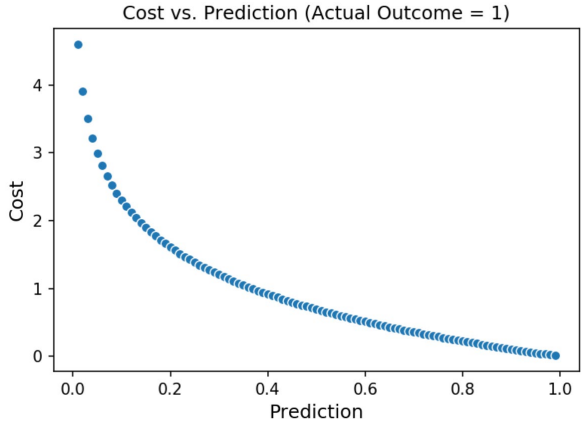
    -log(0.95) = 0.0513
  - Thus, the penalty, in this case, is 0.0513.

  - If we build another model, it spits out a probability of 0.05. In this case, we are massively wrong, and our cost would be:

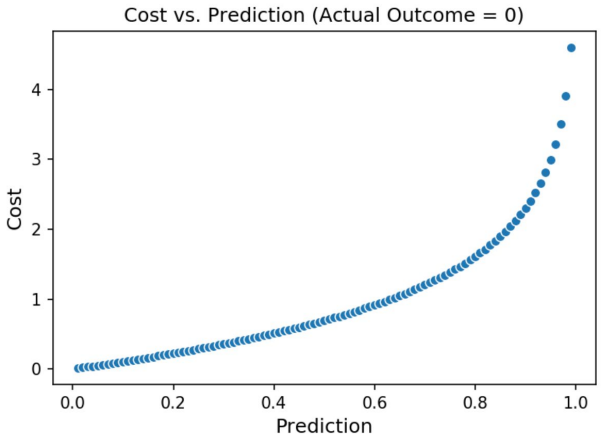
    -log(0.05) = 2.996
  - This cost is significantly higher. In other words, the model was pretty sure that one would miss and it was wrong so we want to (strongly) penalize it; we can do so thanks to taking the natural log.

Example

- The cost function
  - The plots show how the cost relates to the prediction
    - The first plot depicts how cost changes relative to our prediction when the ***Actual Outcome =1***



    - The second plot shows the same but when the ***Actual Outcome = 0***.



Example

- The cost function
  - So for a given observation, we can compute the cost as:

    If Actual Outcome = 1, then Cost = -log(pred_prob)

    Else if Actual Outcome = 0, then Cost = -log(1-pred_prob)

    with pred_prob is the predicted probability that pops out of our model.

  - For the entire data set, we can compute the total cost by:

    Computing the individual cost of each observation using the procedure above.

    Summing up all the individual costs to get the total cost.

  - This total cost is the number we aim to minimize, and we can do so with a gradient descent optimization.
  - In other words, we can run an optimization to find the values of B0 and B1 that minimize total cost.

Example

- Altogether
  - We initially use optimization to search for the values of ***B0*** and ***B1*** that minimize our cost function:

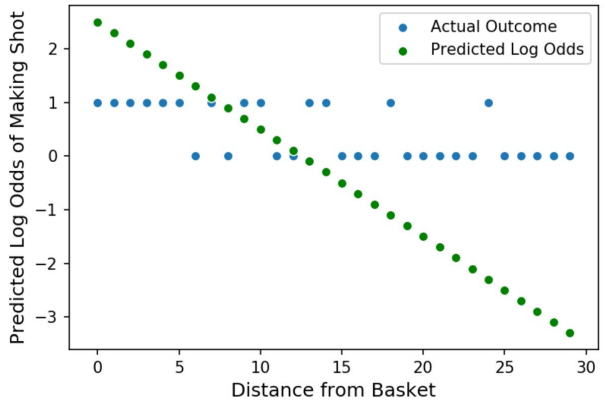
    Z = B0 + B1*X

    Where B0 = 2.5 and B1 = -0.2 (identified via optimization – not done here)

  - We can look at our slope coefficient, ***B1***, which measures the impact that distance has on shooting accuracy.
  - We estimated ***B1*** to be -0.2:
  - This means that for every 1 foot increase in distance, the log odds of making the shot decreases by 0.2.

  - ***B0***, the y-intercept, has a value of 2.5. The model's log odds prediction when to shoot from 0 feet (right next to the basket).
  - Running that the sigmoid function gives us a predicted probability of 92.4%.

  - In the following plot, the green dots depict ***Z***, our predicted log odds.

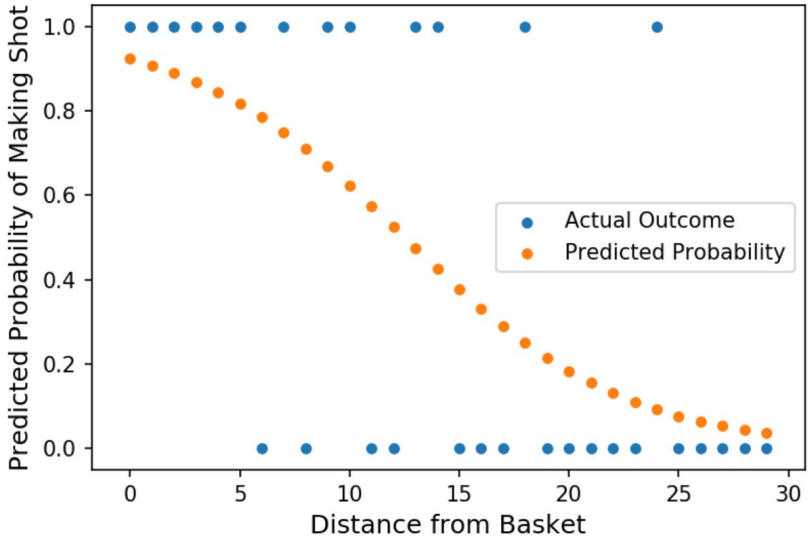


Example

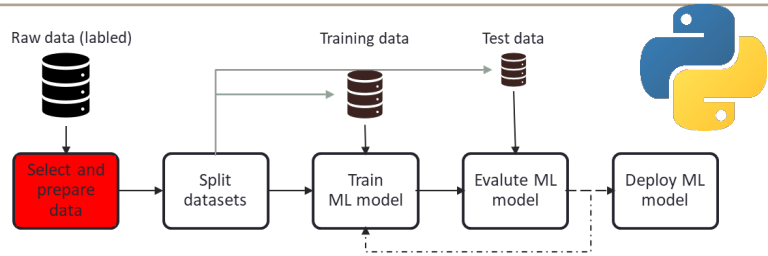
- Altogether
  - Since ***Z*** is in log odds we need to use the sigmoid function to convert it into probabilities:

    Probability of Making Shot = 1 / [1 + e^(-Z)]

  - ***Probability of Making Shot,*** the ultimate output we are looking for is shown in the following plot [orange dots in the Figure below]



  - The curvature reflects that the relationship between the distance and the target is not linear.
  - In probability space (unlike with log odds or linear regression), we cannot say that there is a constant relationship between the distance and the probability of making the shot.
  - Rather, the impact of distance on probability (the slope of the line that connects the orange dots) is itself a function of how far one stands from the basket.



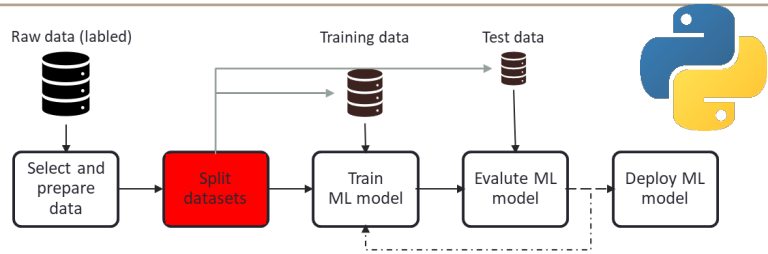
Logistic Regression Classification

Select and prepare data

```
# Import libraries
import pandas as pd

# read the dataset
diabetes = pd.read_csv("diabetes.csv")
```

```
# Splitting dataset in two parts: feature set and target label
feature_set = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree','skin']
X = diabetes[feature_set]
y = diabetes.label
```

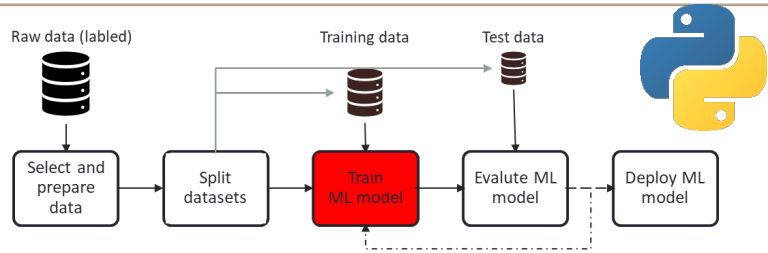
Logistic regression in Python



Split dataset

```
# Partition data into training and testing set
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Logistic regression in Python

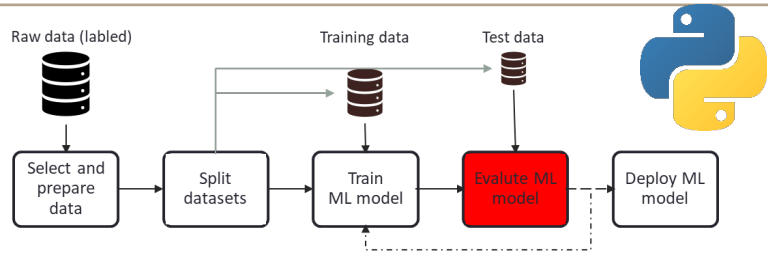


Train logistic regression model

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import LogisticRegression
from sklearn.naive_bayes import accuracy_score # for performance evaluation

# instantiate the model
clf_logr = LogisticRegression(solver='lbfgs',max_iter=500)

# fit the model with data
clf_logr.fit(X_train,y_train)
```

```
LogisticRegression(max_iter=500)
```

Logistic regression in Python

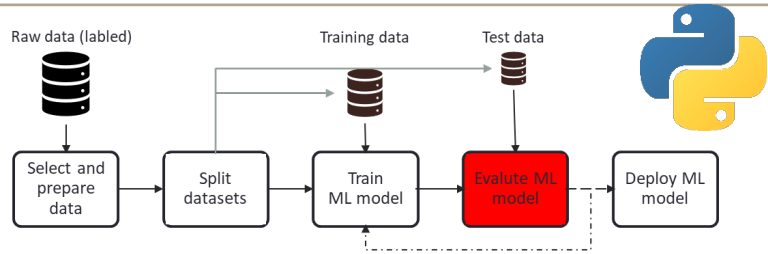


Evaluate logistic regression model (training data)

```
# Predict the y variable for train dataset
y_pred_train = clf_logr.predict(X_train)

# Import metrics module for performance evaluation
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# Calculate model accuracy
print("Accuracy:",accuracy_score(y_train, y_pred_train))
# Calculate model precision
print("Precision:",precision_score(y_train, y_pred_train))
# Calculate model recall
print("Recall:",recall_score(y_train, y_pred_train))
# Calculate model f1 score
print("F1-Score:",f1_score(y_train, y_pred_train))
```

```
Accuracy: 0.7839851024208566
Precision: 0.7432432432432432
Recall: 0.5851063829787234
F1-Score: 0.6547619047619048
```

Logistic regression in Python



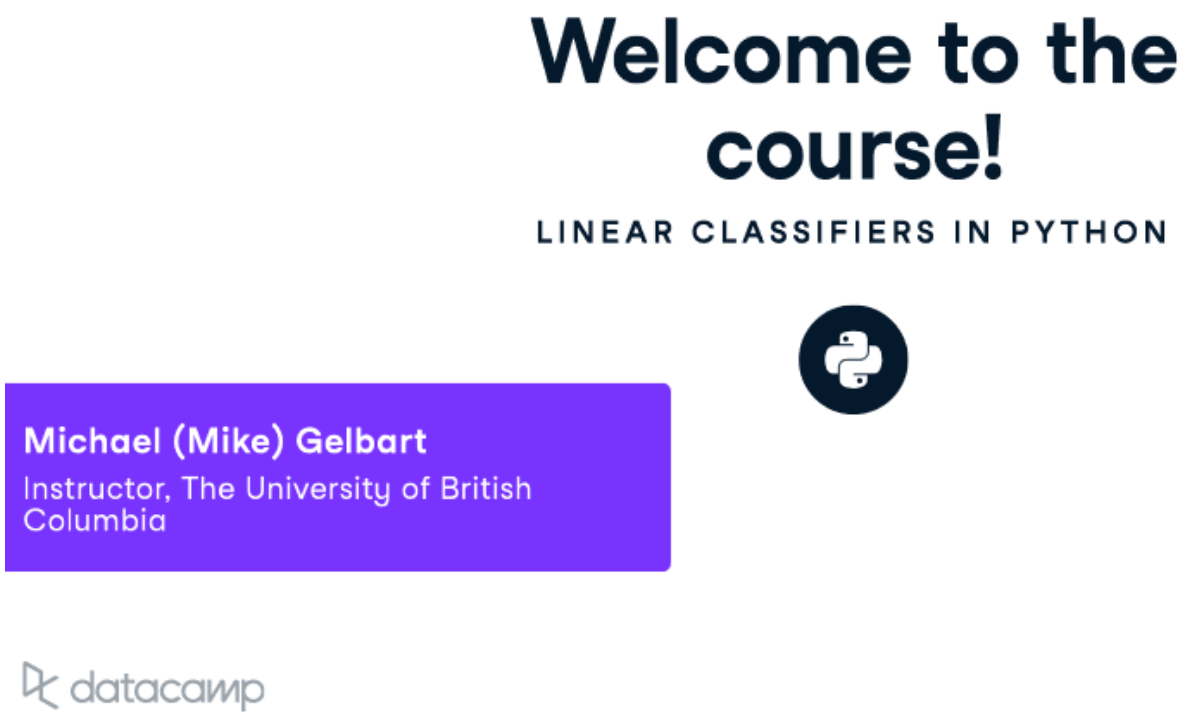
Evaluate logistic regression model (test data)

```
# Predict the y variable for test dataset
y_pred_train = clf_logr.predict(X_test)

# Import metrics module for performance evaluation
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# Calculate model accuracy
print("Accuracy:",accuracy_score(y_test, y_pred_test))
# Calculate model precision
print("Precision:",precision_score(y_test, y_pred_test))
# Calculate model recall
print("Recall:",recall_score(y_test, y_pred_test))
# Calculate model f1 score
print("F1-Score:",f1_score(y_test, y_pred_test))
```

```
Accuracy: 0.7359307359307359
Precision: 0.6172839506172839
Recall: 0.625
F1-Score: 0.6211180124223602
```

You can learn all this in an interactive online course!

https://app.datacamp.com/learn/courses/linear-classifiers-in-python



Classification

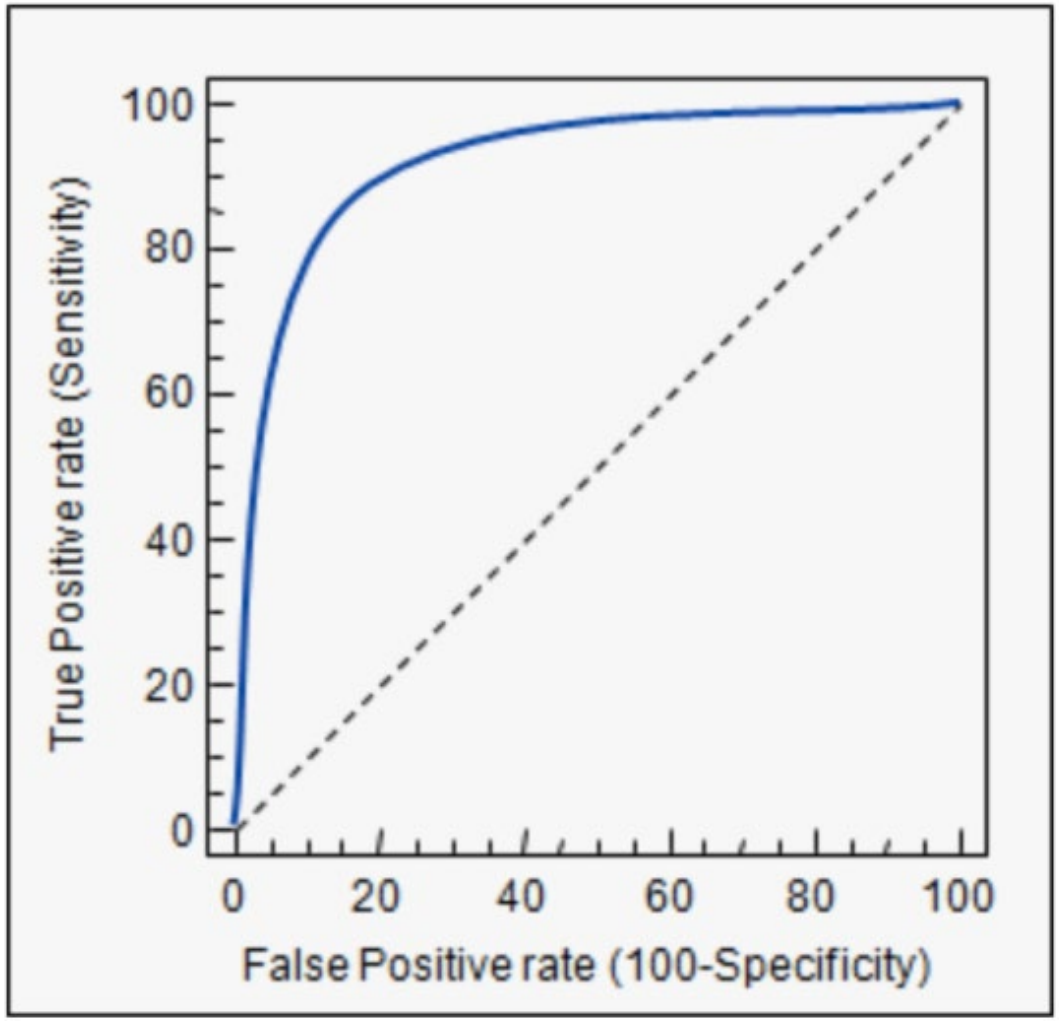
- Introduction
- Decision Trees
- Classifier Evaluation
- k-Nearest Neighbors (kNN)
- Naive Bayes
- Logistic Regression
- **Advanced Classifier Evaluation**

Further Benchmarking possibilities: ROC curves and Area under curve (AUC)

- ROC Curves
  - ROC = “Receiver-Operating-Characteristics”
  - Plot the true positive rate (y-axis) over the false positive rate (x-axis) for a number of candidate thresholds between 0 and 1
  - The false positive rate is also referred to as inverted specificity
    - False positive rate = 1 – specificity = 1 – TN/(FP+TN)
  - In other words: Plots sensitivity over (1-specificity) *or* the false alarm rate over the hit rate
  - Appropriate when the observations are balanced between each class

- Reasons why we use ROC curves
  - The curves of different models can, in general, be compared directly or for different thresholds
  - The *area under the curve* (AUC) can be used as a summary of the model skill

Further Benchmarking possibilities: Example ROC curve



Interpretation of a ROC curve

- The shape of the curve contains much information, including what we might care about most for a problem, the expected false positive rate, and the false negative rate.

- To make this clear:
  - Smaller values on the plot's x-axis indicate lower false positives and higher true negatives.
  - Larger values on the y-axis of the plot indicate higher true positives and lower false negatives.

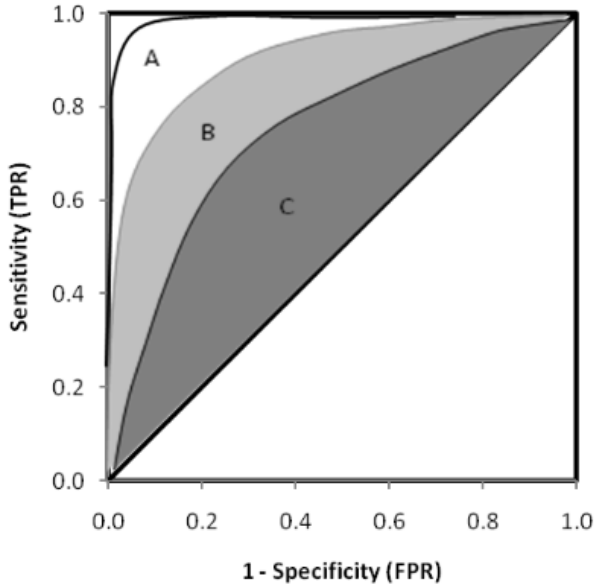
- Remember, when we predict a binary outcome, it is either a correct prediction (true positive) or not (false positive). There is tension between these options, the same with true and false negatives.
  - A skillful model will assign a higher probability to a randomly chosen real positive occurrence than a negative occurrence on average.
    This is what we mean when we say that the model has skill. Generally, skillful models are represented by curves that bow up to the top left of the plot.
  - A no-skill classifier cannot discriminate between the classes and would predict a random class or a constant class in all cases.
    A model with no skill is represented at the point (0.5, 0.5). A model with no skill at each threshold is represented by a diagonal line from the bottom left of the plot to the top right with an AUC of 0.5.

- A model with perfect skill is represented at a point (0,1).
  It is represented by a line that travels from the bottom left of the plot to the top left and then across the top to the top right.

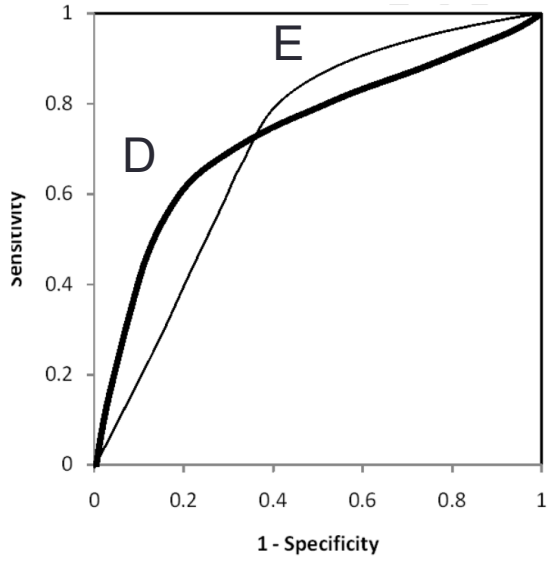
- An operator may plot the ROC curve for the final model and choose a threshold that gives a desirable balance between the false positives and false negatives.

Interpretation of a ROC curve

- AUC = “Area under Curve”
- Compare classifier against the AUC = 0.5 (under the diagonal)
- Here:
  Classifier A is better than classifier B; B is better than C



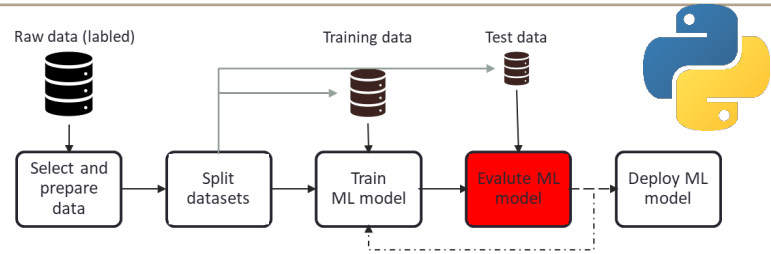
- Which classifier is better in the lower ROC curve?
  - Their AUC is the same but,
  - D is better in the low false positive rate range (low sensitivity range)
  - E is better in the high false positive range (high sensitivity range)



- Remember
  - Sensitive –> **S**creening
  - Specific –> **C**onfirmatory

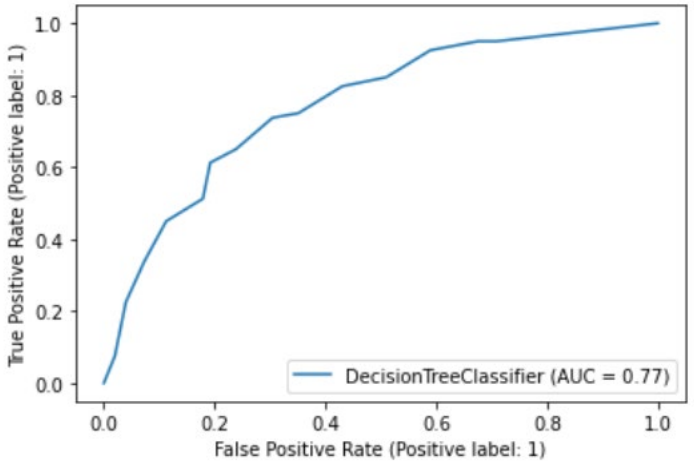
- Both are important
  - Screen for HIV with classifier/test E:
    if the result is negative, you can be sure that you are well
  - Confirm a positive result of classifier E with classifier D:
    if also positive, you can be very sure that you are ill

Evaluation of decision tree model



ROC and AUC

```
# import plot_roc_curve
from sklearn.metrics import plot_roc_curve

plot_roc_curve(clf_tree , X_test, y_test)
```



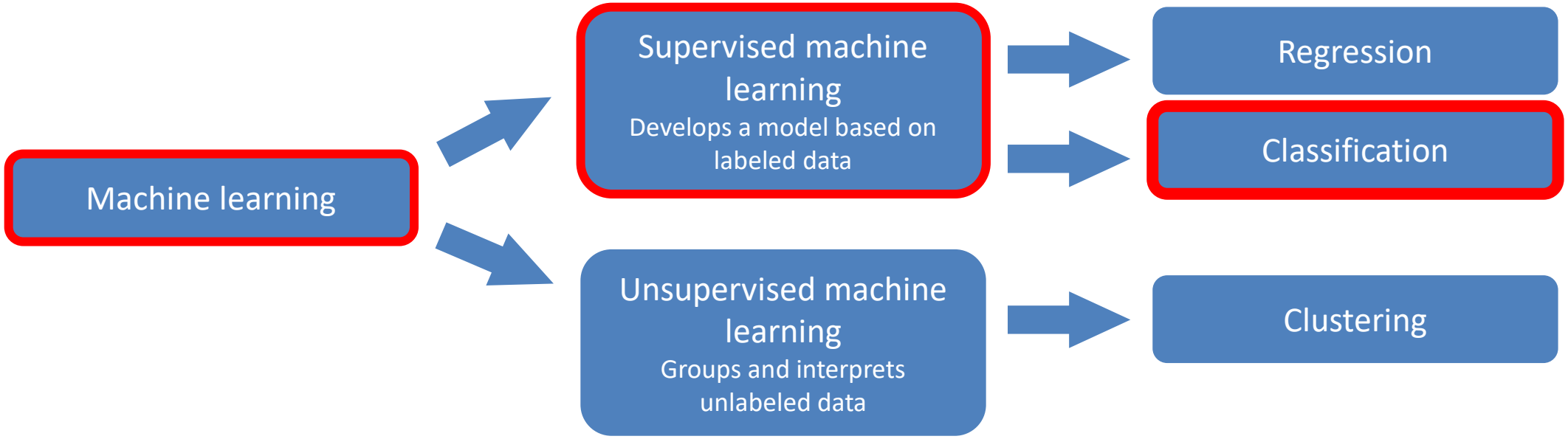

```
# import ROC AUC score
from sklearn.metrics import roc_auc_score

# Compute the area under ROC curve
auc = roc_auc_score(y_test, y_pred_test)

# Print auc value
print("Area Under Curve:",auc)
```

```
Area Under Curve: 0.699544701986755
```

Classification: Summary

Summary: Supervised learning

- Supervised machine learning ("Making predictions based on labeled data")
  - Refers to the process and techniques for assigning a label to each object by inferring a function from the data with supervision from a "teacher" (i.e., assigned labels on training samples)
  - Typical supervised learning tasks and approaches consist of regression, classification, or prediction.
- This lecture
  - introduced various classification algorithms
    - Linear / non-linear algorithms (next slides)
    - Eager / Lazy learning algorithms (next slides)
  - Introduced the basics to evaluate a supervised machine learning model
- Selecting the right machine learning model is an iterative process



- Linear classifiers
  - derive a score via a linear combination of the features and then decide class membership by comparing the score to a threshold

$$\text{score}(X_i, k) = \beta_k \cdot X_i$$

Vector of weights — feature vector

  - The category with the highest score is the category predicted for the observation
  - Linear classifiers include:
    - Logistic regression,
    - Naïve Bayes,
    - Linear support vector machines and many others

- Non-linear classifiers
  - Classifiers that do not make use of linear combinations of features
  - They use probabilistic classification: the probability of group membership is estimated
  - Non-linear classifiers include:
    - K-nearest neighbors,
    - Decision trees,
    - Multilayer neuronal networks (not covered in this lecture)
    - Non-linear support vector machines and others (not covered in this lecture)

- The performance of the classifiers depends on the data structure
  - There is no “one fits all” algorithm for the great variety of problems
  - Finding “the best” classifiers for a problem is often done by trial and error

There are two types of learners in classification: lazy and eager learners

- Lazy learners
  - Simply store the training data and wait until testing data appears
  - When it does, classification is conducted based on the most related data in the stored training data
  - Needs little training time but more time for classification
  - Primary motivation:
    - The data set is continuously updated with new entries (e.g., new items for sale at Amazon).
    - Because of the continuous update, the "training data" would be rendered obsolete in a relatively short time especially in areas like books and movies, where new best-sellers or hit movies/music are published/released continuously.
    - Therefore, one cannot talk of a "training phase“
  - Advantages:
    - The classification model will be approximated locally
    - Because the classification model is approximated locally for each query to the system, lazy learning systems can simultaneously solve multiple problems and deal successfully with changes in the problem domain
  - Disadvantages:
    - Large space requirement to store the entire training dataset
    - Particularly noisy training data increases the case base unnecessarily because no abstraction is made during the training phase
    - Lazy learning methods are usually slower making predictions
  - Example: k-nearest neighbors

There are two types of learners in classification: lazy and eager learners

- Eager learners
  - Construct a classification model based on the given training data before receiving data for classification
  - It must be able to commit to a single hypothesis that covers the entire instance space
  - Take a long time for train and less time to predict
  - Advantages:
    - The classification model will be approximated globally during training, thus requiring much less space than using a lazy learning system
    - Eager learning systems also deal much better with noise in the training data
    - Eager learning is an example of offline learning, in which post-training queries to the system do not affect the system itself, and thus the same query to the system will always produce the same result
  - Disadvantages:
    - Generally unable to provide good local approximations in the classification model
    - Need to be regularly retrained on new data
  - Examples: Naïve Bayes, Decision Tree algorithms, Artificial Neural Networks

Classification: Learning objectives

- After this lecture, students shall be able to…
  - Understand and interpret different evaluation criteria
  - Know how to use
    - k-Nearest Neighbors (kNN)
    - Naive Bayes
    - Logistic Regression
    - Decision Trees

    … for classification purposes

Please apply kNN by using AGE, INCOME, and CARDS as factors influencing RESPONSE

a) Calculate the Distance

b) Select different k values (e.g., 1, 2, 3, 4, 5) and interpret how to select k and how it influences the result.

Customer	Age	Income(1000s)	Cards	Response (target)	Distance from David
David	37	50	2	?	
John	35	35	3	Yes	
Rachel	22	50	2	No	
Ruth	63	200	1	No	
Jefferson	59	170	1	No	
Norah	25	40	4	Yes	

Please apply kNN by using AGE, INCOME, and CARDS as factors influencing RESPONSE

a) Calculate the Distance

b) Select different k values (e.g., 1, 2, 3, 4, 5) and interpret how to select k and how it influences the result.

Customer	Age	Income(1000s)	Cards	Response (target)	Distance from David
David	37	50	2	?	0
John	35	35	3	Yes	$\sqrt{(35-37)^2+(35-50)^2+(3-2)^2} = 15.16$
Rachel	22	50	2	No	$\sqrt{(22-37)^2+(50-50)^2+(2-2)^2} = 15$
Ruth	63	200	1	No	$\sqrt{(63-37)^2+(200-50)^2+(1-2)^2} = 152.23$
Jefferson	59	170	1	No	$\sqrt{(59-37)^2+(170-50)^2+(1-2)^2} = 122$
Norah	25	40	4	Yes	$\sqrt{(25-37)^2+(40-50)^2+(4-2)^2} = 15.74$

Please calculate the learned classification quality metrics

	True positive	True negative	Total
Predicted positive	**TP = 95**	**FP = 5**	**100**
Predicted positive	**FN = 0**	**TN = 0**	**0**
Total	**95**	**5**	**100**

	True positive	True negative	Total
Predicted positive	**TP = 90**	**FP = 4**	**94**
Predicted positive	**FN = 5**	**TN = 1**	**6**
Total	**95**	**5**	**100**

Please calculate the learned classification quality metrics

	True positive	True negative	Total
Predicted positive	**TP = 95**	**FP = 5**	**100**
Predicted positive	**FN = 0**	**TN = 0**	**0**
Total	**95**	**5**	**100**

Accuracy: 0.95
Precision: 0.95
Recall: 1.00

	True positive	True negative	Total
Predicted positive	**TP = 90**	**FP = 4**	**94**
Predicted positive	**FN = 5**	**TN = 1**	**6**
Total	**95**	**5**	**100**

Accuracy: 0.91
Precision: 0.9574
Recall: 0.9474