DATA SCIENCE 2 - DATA & A.I. 3

V MACHINE LEARNING

4-7 ML TECHNIQUES

Karel de Grote Hogeschool

DATA SCIENCE 2
DATA & A.I. 3

I — **PYTHON BASICS**
Python for data science

II — **WORKING WITH ARRAYS**
Numpy

III — **DATA ENGINEERING**
pandas

IV — **DATA VISUALISATION**
Matplotlib

V — **MACHINE LEARNING**
Automatically find patterns

V

MACHINE LEARNING

scikit-learn

MACHINE LEARNING

4

SUPERVISED LEARNING: REGRESSION

Linear and Polynomial Regression

REGRESSION (SUPERVISED LEARNING)

What?

prediction of numerical outcomes

How?

Fit a mathematical function to the data to model the relationship between features and the target.

Linear Regression: Fits a linear function to predict a numerical outcome.
Polynomial Regression: Fits a polynomial function (with varying degrees) to capture non-linear relationships in the data.

REGRESSION
(SUPERVISED LEARNING)

Basic code framework:

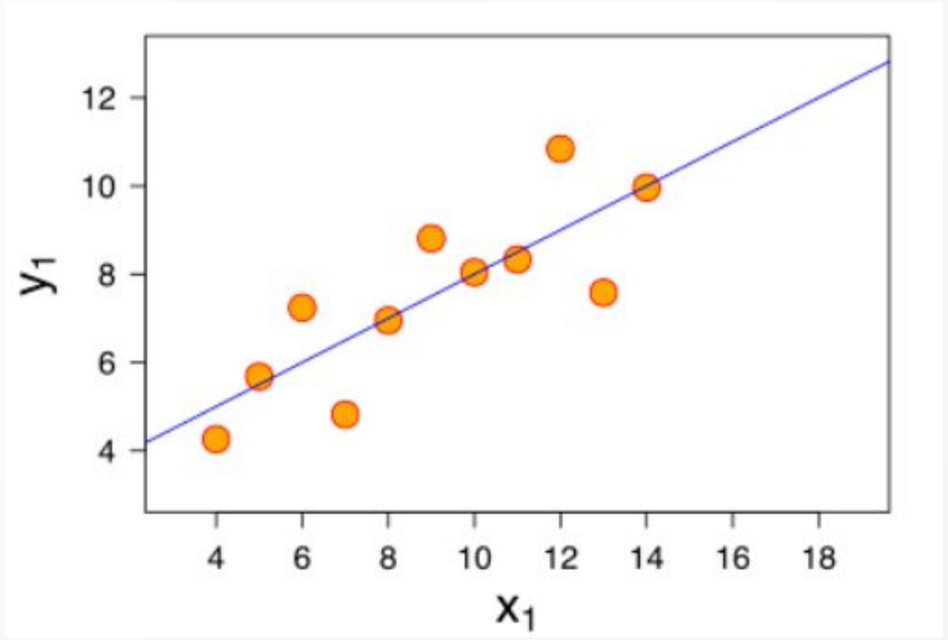
```
from sklearn.model_family import ModelAlgo
mymodel = ModelAlgo(param1,param2)
mymodel.fit(X_train,y_train)
predictions = mymodel.predict(X_test)

from sklearn.metrics import error_metric
performance = error_metric(y_test,predictions)
```

This framework will be similar for any supervised machine learning algorithm.
Let’s begin exploring it further with Linear Regression!

LINEAR REGRESSION

prediction of numerical features
Try to fit a line ($y = ax + b$) to the data in the best possible way

See Data & A.I. 2



LINEAR REGRESSION WITH SCIKIT-LEARN

```
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)
```

Hyperparameters:

fit_intercept

- **Type**: boolean, default = True
- **Description**: Determines whether or not the model should calculate the intercept (also called the bias term).If True, the model calculates the intercept. If False, the model assumes the data is already centered around the origin (i.e., the intercept is 0).
- **Usage**: fit_intercept=False may be useful if you know the data is already centered or you want a model without an intercept.

copy_X

- **Type**: boolean, default = True
- **Description**: If True, X will be copied before fitting the model. If False, changes to the original data (such as scaling) will affect the original dataset. It's usually safer to leave this as True.

n_jobs

- **Type**: int, default = None
- **Description**: This specifies the number of CPU cores to use when fitting the model. If n_jobs=-1, all available cores will be used. This can help speed up the model fitting when working with large datasets.

LINEAR REGRESSION WITH SCIKIT-LEARN

```
# DATA PREPARATION
import pandas as pd
pd.options.display.max_rows = None
import seaborn as sns
iris = sns.load_dataset('iris')
X = iris[['sepal_width', 'sepal_length', 'petal_width']] # Predictors
y = iris['petal_length'] # Target feature to predict

# MODEL SELECTION AND HYPERPARAMETER SELECTION (MODEL SPECIFIC)
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)
print(model)
# List all selected hyperparameters
print(model.get_params(deep=True))

# DERIVE MODEL FROM LABELED DATA (TRAIN MODEL/FIT MODEL)
model.fit(X,y)
```

LINEAR REGRESSION WITH SCIKIT-LEARN

```
# DISPLAY COEFFICIENTS (slope and intercept)
print(f"Intercept: {model.intercept_}")
print(f"Coefficients: {model.coef_}")

# SHOW REGRESSION LINE IN SCATTERPLOT
plt.figure(figsize=(8,6))
sns.regplot(x='X_test',y='y_test',data=iris,scatter_kws={"color":"blue"}, line_kws={"color":"red"})
plt.title("Linear Regression: Actual vs Predicted")
plt.xlabel("X values")
plt.ylabel("y values")
plt.show()
```

LINEAR REGRESSION WITH SCIKIT-LEARN

```
# VALIDATE MODEL USING LABELED DATA
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error,
root_mean_squared_error, r2_score

# Predict target feature for the labeled data
y_pred = pd.Series(model.predict(X), name='y_pred')

# Calculate the difference between predicted and real values for the labeled data
err = pd.Series(y_pred-y, name='err')
display(pd.concat([y, y_pred, err], axis=1))

# Metrics
mae = mean_absolute_error (y_true=y, y_pred=y_pred)
mape = mean_absolute_percentage_error (y_true=y, y_pred=y_pred)
rmse = root_mean_squared_error (y_true=y, y_pred=y_pred)
r2 = r2_score(y_test, y_pred)
print(f'MAE : {mae:.3f} - MAPE : {mape:.3f} - RMSE : {rmse:.3f} - R^2 : {r2:.3f}')
```

LINEAR REGRESSION WITH SCIKIT-LEARN

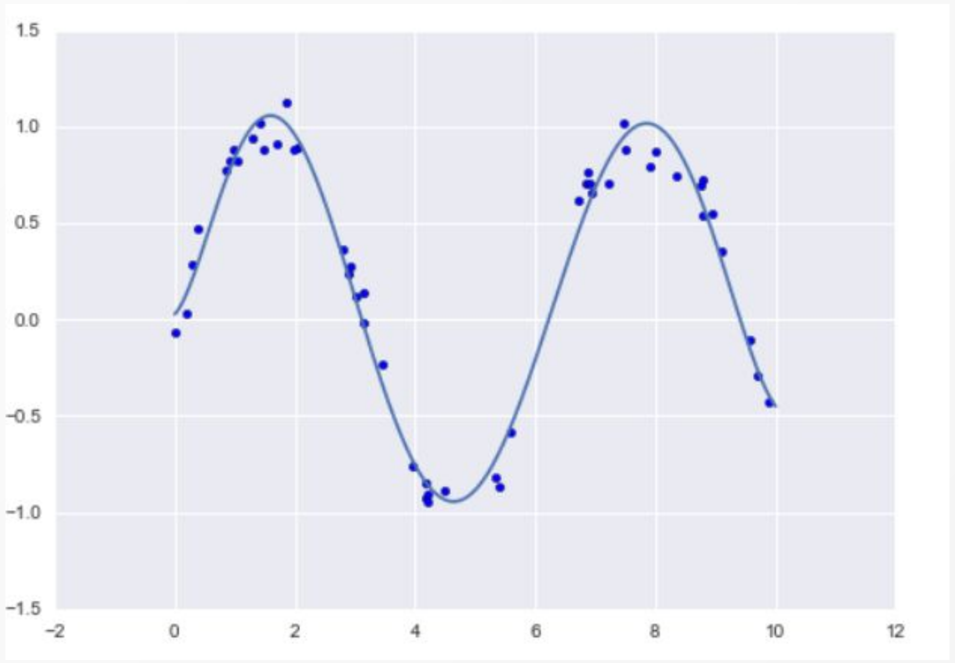
```
# APPLY MODEL ON NEW DATA
X_pred = …. (new feature data to predict the target feature for)
y_pred = model.predict(X_pred)
```

POLYNOMIAL REGRESSION

prediction of numerical features

Try to fit a polynomial function (y = ax^2 + bx + c) to the data in the best possible way

See Data & A.I. 2



POLYNOMIAL REGRESSION WITH SCIKIT-LEARN

1. Import necessary libraries: No major change here, still using numpy, pandas, and scikit-learn like with linear regression.

2. Transform features for Polynomial Regression: We need to **add polynomial features** (e.g., squared, cubic terms) to capture non-linear relationships:

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2) # Choose the degree (2 for quadratic, 3 for cubic, etc.)
X_poly = poly.fit_transform(X)
```

3. Fit the Polynomial Regression model: Same as with linear regression, but now using the transformed polynomial features:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_poly, y)
```

4. Make Predictions: Use the polynomial-transformed features for predictions:

```
y_pred = model.predict(poly.transform(X_test))
```

5. Evaluate the model: Same steps as with linear regression

Notebook:

See
05.06-Linear-Regression.ipynb

Exercises:
05.07-EX.ipynb

SUPERVISED LEARNING: DECISION TREES

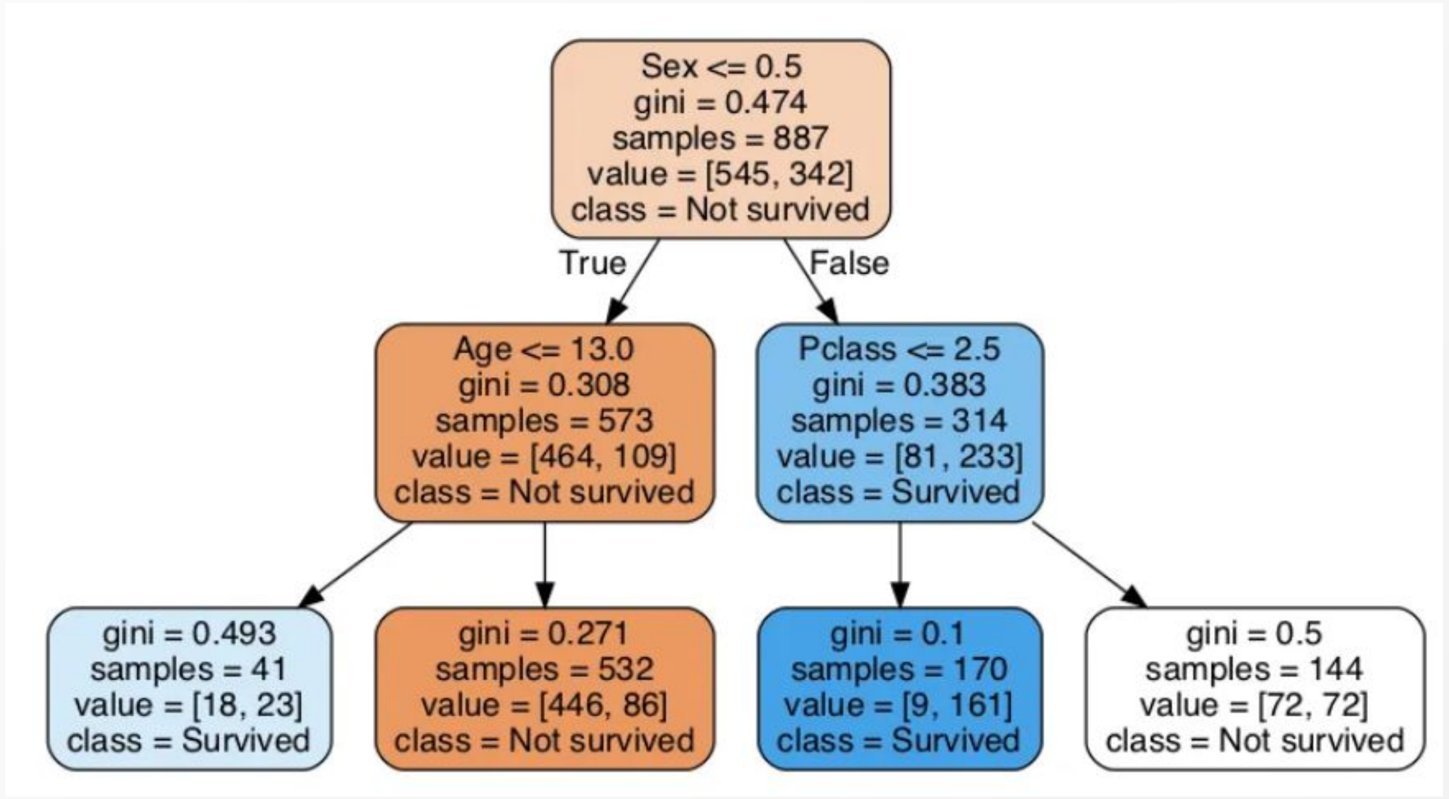
Decision Trees and Random Forests

5

DECISION TREES

Decision trees:
A supervised learning algorithm used for **classification** and **regression** tasks creating a tree-like model of decisions based on input features

See Data & A.I. 2



DECISION TREES WITH SCIKIT-LEARN

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion='gini’)
```

Hyperparameters:

criterion

- **Type**: string, default='gini’
- **Description**: This function measures the quality of a split. Supported criteria are 'gini' for the Gini impurity and 'entropy' for the information gain.
- **Usage**: Choose 'gini' for the Gini impurity (default) or 'entropy' for information gain when building the decision tree.

max_depth

- **Type**: int, default=None
- **Description**: The maximum depth of the tree. If None, nodes are expanded until all leaves are pure or contain fewer samples than the min_samples_split.
- **Usage**: Set this to limit the depth of the tree and avoid overfitting.

min_samples_split

- **Type**: int or float, default=2
- **Description**: The minimum number of samples required to split an internal node. If an integer, it's the minimum number. If a float, it's the fraction of the total number of samples.
- **Usage**: Increase this value to reduce overfitting by making the tree more generalized.

DECISION TREES WITH SCIKIT-LEARN

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion='gini')
```

Hyperparameters:

min_samples_leaf
- **Type**: int or float, default=1
- **Description**: The minimum number of samples required to be at a leaf node. A smaller value allows smaller leaves and may lead to overfitting.
- **Usage**: A higher value can make the model more conservative and less prone to overfitting.

max_features
- **Type**: int, float, string or None, default=None
- **Description**: The number of features to consider when looking for the best split. If None, all features are considered.
- **Usage**: Restricting the number of features can lead to a more generalized model.

DECISION TREES WITH SCIKIT-LEARN: EXAMPLE

```
# DATA PREPARATION
import pandas as pd
pd.options.display.max_rows = None
import seaborn as sns
iris = sns.load_dataset('iris')
y = iris['species'] # Target feature to predict
X = iris.copy().drop('species', axis=1) # Predictors

# MODEL SELECTION AND HYPERPARAMETER SELECTION (MODEL SPECIFIC)
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(max_depth=1)
print(model)
# List all selected hyperparameters
print(model.get_params(deep=True))

# DERIVE MODEL FROM LABELED DATA (TRAIN MODEL/FIT MODEL)
model.fit(X,y)
```

DECISION TREES WITH SCIKIT-LEARN: EXAMPLE

```
# DISPLAY MODEL (MODEL SPECIFIC)
from sklearn.tree import plot_tree
plot_tree(model)

# VALIDATE MODEL USING LABELED DATA
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,
f1_score, precision_recall_fscore_support, classification_report
import matplotlib.pyplot as plt

# Predict target feature for the labeled data
y_pred = pd.Seeries(model.predict(X), name='y_pred')

# Calculate the difference between predicted and real values for the labeled data
err = pd.Series(y_tst_pred.reset_index(drop=True)!=y_tst.reset_index(drop=True),
name='err').astype(int)
display(pd.concat([y_tst.reset_index(drop=True), y_tst_pred.reset_index(drop=True), err],
axis=1))
```

DECISION TREES WITH SCIKIT-LEARN: EXAMPLE

```
# Confusion matrix
# Display as text (console output)
class_labels = sorted(list(pd.concat([y_tst,y_tst_pred], axis=0).unique()))
# Alternative : model.classes_
cm = confusion_matrix(y_true = y_tst, y_pred = y_tst_pred)
print('Predicted label')
print(class_labels)
print(cm)
# Display as heatmap (nicer output in Jupyter)
disp = sns.heatmap(cm, square=True, annot=True, cbar=True, cmap='Greys',
xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted label')
plt.ylabel('True label')
disp.xaxis.tick_top()                # Put x-axis tickers on top
disp.xaxis.set_label_position('top') # Put x-axis label on top
```

DECISION TREES WITH SCIKIT-LEARN: EXAMPLE

```
# Metrics
acc = accuracy_score(y_true=y, y_pred=y_pred)
prec = precision_score(y_true=y, y_pred=y_pred, average='weighted')
rec = recall_score(y_true=y, y_pred=y_pred, average='weighted')
f1 = f1_score(y_true=y, y_pred=y_pred, average='weighted')
# Mind this is a multiclass classification problem, so precision, recall and F1
# are calculated by class and averaged.
print(f'ACC : {acc:.3f} - PREC : {prec:.3f} - REC : {rec:.3f} - F1 : {f1:.3f}')

# The easiest way to get results by class is to use precision_recall_fscore_support
class_labels = sorted(list(pd.concat([y_tst,y_tst_pred], axis=0).unique()))
# Display precision/recall/fscore/support table as text (consule output)
print(class_labels)
display(precision_recall_fscore_support(y_true=y_tst, y_pred=y_tst_pred))
# Display precision/recall/fscore/support as pandas dataframe (nicer outputin Jupyter)
display(pd.DataFrame(precision_recall_fscore_support(y_true=y_tst, y_pred=y_tst_pred),
index=['prec','rec','fscore','sup'], columns=class_labels))

# Or use classification_report
print(classification_report(y_true=y_tst, y_pred=y_tst_pred,
target_names=class_labels))
```

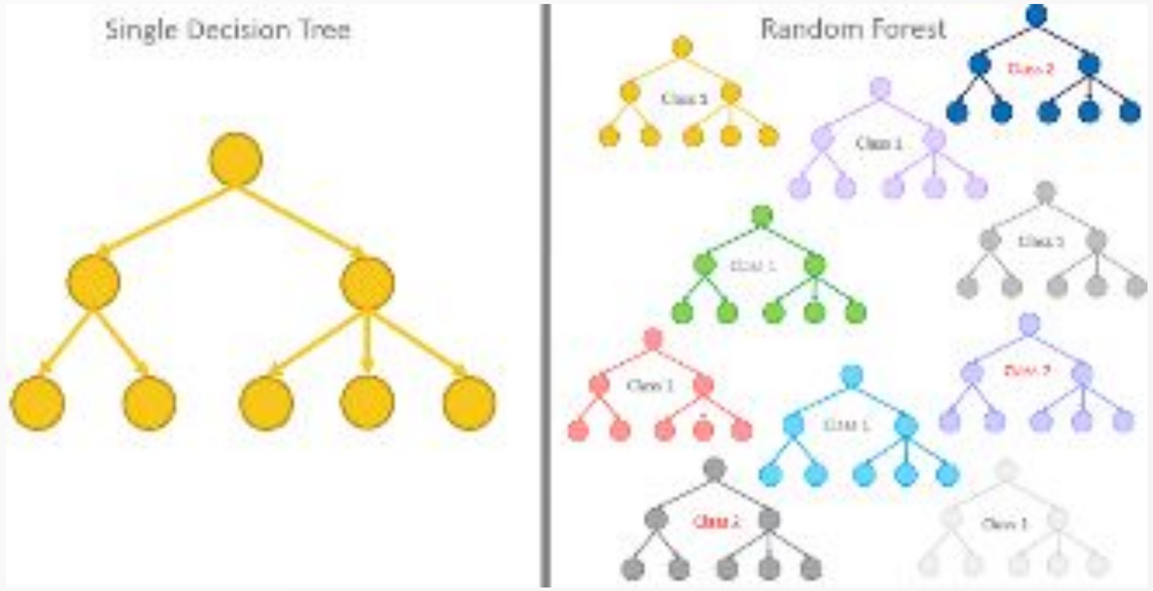
DECISION TREES WITH SCIKIT-LEARN: EXAMPLE

```
# APPLY MODEL ON NEW DATA
X_pred = …. (new feature data to predict the target feature for)
y_pred = model.predict(X_pred)
```

RANDOM FORESTS

Random forests:
An ensemble method that builds multiple decision trees and aggregates their results for improved accuracy in classification and regression tasks.

See Data & A.I. 2



RANDOM FORESTS WITH SCIKIT-LEARN

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100)
```

Hyperparameters:

n_estimators
- **Type**: int, default=100
- **Description**: The number of trees in the forest. More trees usually lead to better performance but increase computation time.
- **Usage**: Use higher values (e.g., 500, 1000) for larger datasets to improve performance.

max_depth
- **Type**: int, default=None
- **Description**: The maximum depth of each tree in the forest. If None, nodes are expanded until all leaves are pure.
- **Usage**: Restrict this to prevent overfitting, especially when building deep trees.

max_features
- **Type**: int, float, string or None, default='auto'
- **Description**: The number of features to consider when looking for the best split. 'auto' uses the square root of the number of features.
- **Usage**: Try different values such as 'sqrt' (square root) or 'log2' to tune the model for better performance.

min_samples_split
- **Type**: int or float, default=2
- **Description**: The minimum number of samples required to split an internal node in each tree.
- **Usage**: Adjust this to balance between underfitting and overfitting.

RANDOM FORESTS WITH SCIKIT-LEARN: EXAMPLE

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Create some example data
# For illustration, we generate a synthetic dataset
from sklearn.datasets import make_classification

X,    y    =    make_classification(n_samples=1000,    n_features=10,    n_classes=2,
random_state=42)

# Split the data into training and test sets
X_train,   X_test,   y_train,   y_test   =   train_test_split(X,   y,   test_size=0.3,
random_state=42)

# Initialize and train the Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
model.fit(X_train, y_train)
```

RANDOM FORESTS WITH SCIKIT-LEARN: EXAMPLE

```
# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

DECISION TREES AND RANDOM FORESTS

Notebook:

See
05.08-Random-Forests.ipynb

Exercises:
05.08and11_EX.ipynb

UNSUPERVISED LEARNING: CLUSTERING

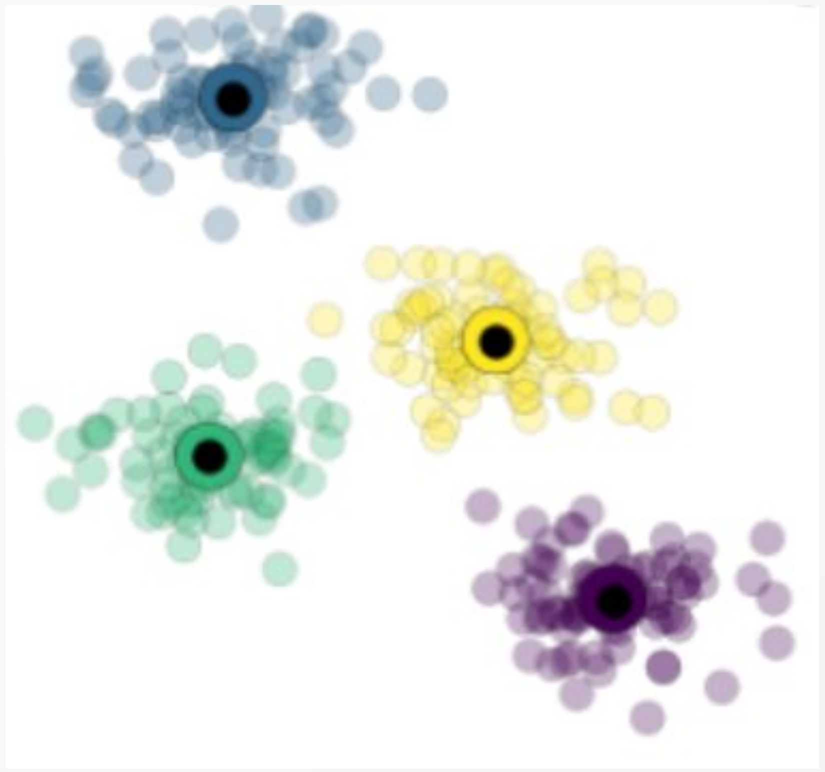
K-Means

6

K-MEANS CLUSTERING

K-Means Clustering:
An unsupervised machine learning algorithm used to group similar data points into a predefined number of clusters based on their feature similarities.

See Data & A.I. 2



K-MEANS CLUSTERING WITH SCIKIT-LEARN

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=4)
```

Hyperparameters:

n_clusters
- **Type**: int, default=8
- **Description**: The number of clusters to form, and the number of centroids to generate.
- **Usage**: Set this to specify how many groups you want the data divided into.

init
- **Type**: {‘k-means++’, ‘random’}, default=‘k-means++’
- **Description**: Method for initializing the cluster centroids. k-means++ chooses centroids to speed up convergence, while random selects random points.
- **Usage**: k-means++ is typically preferred, but random can be used for experimentation.

max_iter
- **Type**: int, default=300
- **Description**: The maximum number of iterations allowed for the algorithm to run until it converges.
- **Usage**: Increase this if your model is not converging, but usually the default works well.

K-MEANS CLUSTERING WITH SCIKIT-LEARN

Hyperparameters:

tol

- **Type**: float, default=1e-4
- **Description**: The tolerance for the convergence criterion. The algorithm stops when the difference in the cluster centers falls below this threshold.
- **Usage**: Decrease this value to make the algorithm more precise at the cost of computation time.

n_init

- **Type**: int, default=10
- **Description**: The number of times the algorithm will be run with different centroid seeds. The final result will be the best output of these runs.
- **Usage**: Increase this to get a more stable result by running the algorithm multiple times with different initializations.

algorithm

- **Type**: {'auto', 'full', 'elkan'}, default='llkan'
- **Description**: The algorithm to use for clustering. elkan is faster with sparse data, while full uses the standard EM-style algorithm.
- **Usage**: elkan is recommended for efficiency, especially when handling large datasets.

random_state

- **Type**: int, default=None
- **Description**: The seed used by the random number generator. Setting this ensures the same results each time the algorithm is run.
- **Usage**: Useful when you want reproducible results.

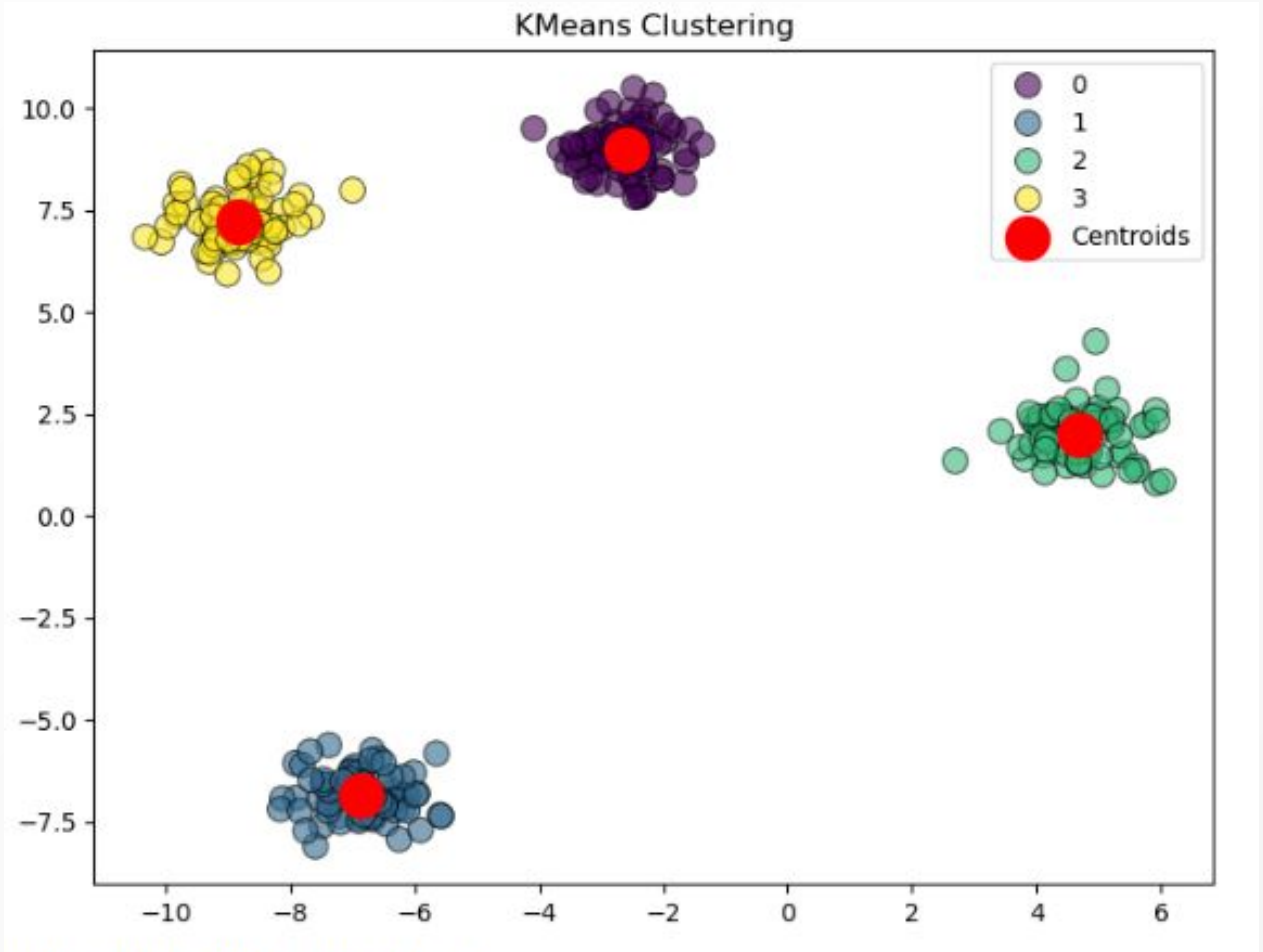
K-MEANS CLUSTERING WITH SCIKIT-LEARN EXAMPLE

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import seaborn as sns
import matplotlib.pyplot as plt

# Create or load some example data
# For illustration, we generate a synthetic dataset
from sklearn.datasets import make_blobs

# Generate synthetic data with 4 clusters
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=42)

# Initialize and fit the KMeans model
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, random_state=42)
kmeans.fit(X)

# Get the cluster centers and labels
centroids = kmeans.cluster_centers_
labels = kmeans.labels_
```

K-MEANS CLUSTERING WITH SCIKIT-LEARN EXAMPLE

```
# Get the cluster centers and labels
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

# Visualize the clusters
plt.figure(figsize=(8, 6))

# Plot the points and color by cluster
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette="viridis", s=100, alpha=0.6,
edgecolor="k")

# Plot the centroids
plt.scatter(centroids[:, 0], centroids[:, 1], s=300, c='red', label='Centroids')
plt.title('KMeans Clustering')
plt.legend()
plt.show()

# Predicting new points
new_points = np.array([[0, 0], [5, 5], [-5, -5]])
predictions = kmeans.predict(new_points)
print(f"Cluster labels for new points: {predictions}")
```

K-MEANS CLUSTERING WITH SCIKIT-LEARN EXAMPLE



```
Cluster labels for new points: [2 2 1]
```

K-MEANS CLUSTERING

Notebook:

See **05.11-K-Means.ipynb**

Exercises:
05.08and11_EX.ipynb

TEMPLATE

Dark colour :	RGB 67 67 67 (HEX #434343)
Dark colour greyed out :	RGB 191 191 191 (HEX #BFBFBF)
Light colour :	RGB 242 242 242 (HEX #F2F2F2)
Light colour greyed out:	RGB 191 191 191 (HEX #BFBFBF)
Base font :	Roboto Condensed (size 16 / 14)
Alternative font:	Exo 2 (bold, size 28 / 20 / 16 / 14) or Proxima Nova or Verdana
Dark gradient fill	Linear 45° RGB 35 35 35 - RGB 67 67 67
Light gradient fill	Linear 45° RGB 255 255 255 - RGB 242 242 242

WHAT IS THIS ALL ABOUT

INTRODUCTION TO EXPLORATIVE DATA ANALYSIS

Why is this happening

UNIVARIATE

One variable at a time

MULTIVARIATE

Multiple variables at the same time

IV

SECTION 4

Whatever

V

SECTION 5

Whatever

WHAT IS THIS ALL ABOUT

WHAT IS DATA ANALYTICS

Data driven decision support

1

TITLE CHAPTER 2

Subtitle chapter 2

TITLE CHAPTER 3

Subtitle chapter 3

TITLE CHAPTER 4

Subtitle chapter 4

4

TITLE CHAPTER 5

Subtitle chapter 5

TITLE CHAPTER 6

Subtitle chapter 6

SUMMARY

WAARDECREATIE

- Data analytics gaat over het creëren van waarde;
- Maatschappelijke waarde en bedrijfswaarde;
- Door allerlei producten en diensten te verbeteren;
- In alle mogelijke afdelingen;
- In alle mogelijke sectoren.

3 MANIEREN VAN WAARDECREATIE

- Kostreductie: doe bestaande dingen goedkoper;
- Snellere en betere beslissingen: doe bestaande dingen beter;
- Nieuwe producten en diensten: doe nieuwe dingen.

WAT IS NIEUW?

- Niet nieuw, statistiiek bestaat al meer dan 200 jaar;
- Maar groeit en evolueert snel, omwille van:
  - massieve beschikbaarheid van data,
  - massive computerkracht,
  - geavanceerde technieken en gereedschappen.

DATA DRIVEN COMPETITION

- De huidige boom is geen hype;
- Technologische factoren (beschikbaarheid van data en computerkracht, technieken en gereedschappen) zijn enkel maar faciliterende factoren;
- De echte drijvende factor is innovatie;
- Niet de traditionele incrementele innovatie (alles wordt elk jaar een beetje beter);
- Maar radicale innovatie (radicaal nieuwe of betere producten);
- Daar zit het geld;
- De kracht van data analytics zit in het potentieel voor radicale innovatie;
- Resulteert in data driven competitiie: je verwerft een competitief voordeel dat niet genegeerd kan worden.

RADICALE INNOVATIE

- Procesinnovatie: maak bestaande producten en diensten goedkoper;
- Productinnovatie: maak bestaande producten en diensten beter of creëer nieuwe producten en diensten;
- Marktinnovatie: herschrijf de regels;