COIT20277 Introduction to Artificial Intelligence

Week 3 - Lecture

- Supervised Learning: Regression
- Unsupervised Learning

BE WHAT YOU WANT TO BE
cqu.edu.au

Acknowledgement of Country

I respectfully acknowledge the Traditional Custodians of the land on which we live, work and learn. I pay my respects to the First Nations people and their Elders, past, present and future

CQUniversity AUSTRALIA

BE WHAT YOU WANT TO BE
cqu.edu.au

Acknowledgment

The content of this lecture has been adopted from the following book:

- Artificial Intelligence Programming with Python - From Zero to Hero, 2022, Perry Xiao, *John Wiley & Sons, Inc.*, ISBN 978-1-119-82086-4.

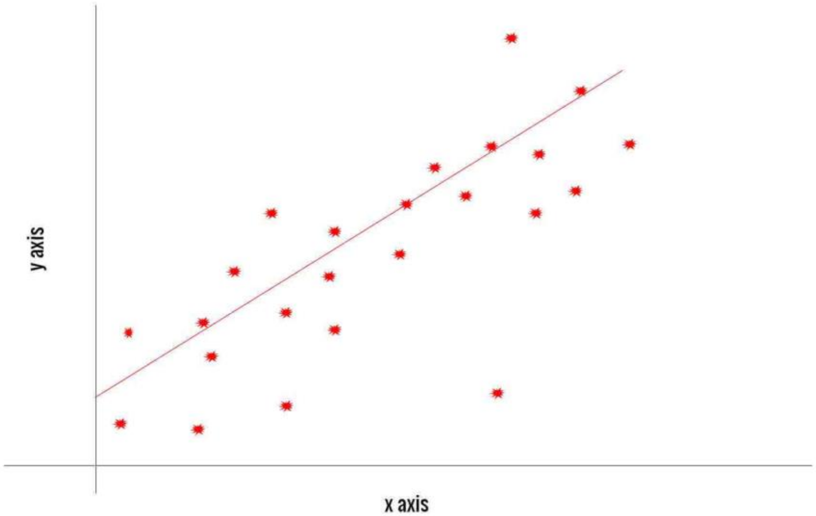
- Chapter 3 (Sections 3.3 - 3.4)

Supervised Learning: Regression

- Introduction to Regression
- Linear Regression
  - Definition and Key Concepts
  - Least Squares Fitting
  - Example with Python Code
- Nonlinear Regression
  - Polynomial Regression
  - Logistic Regression
- Resources and Conclusion

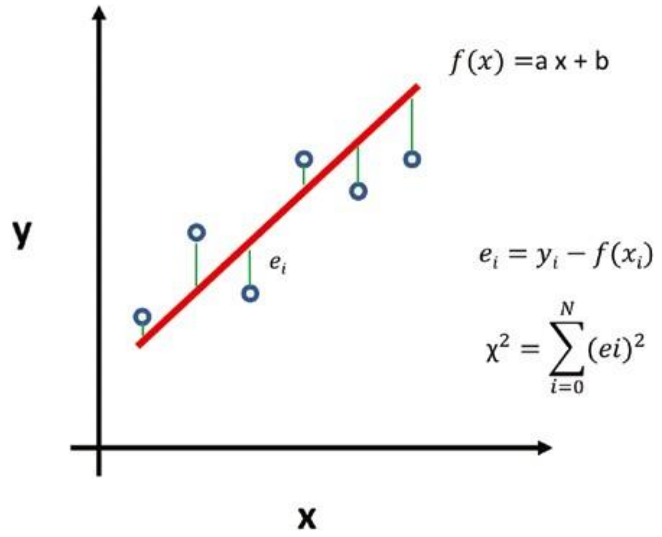
Introduction to Regression

- Regression is a supervised learning technique for predicting continuous values.
- It involves fitting a mathematical model to the data using a technique called *least squares fitting*.
- Regression can be used for various tasks, such as forecasting sales, predicting house prices, and analyzing relationships between variables.



Linear Regression

- Linear regression is the simplest form of regression, where the model is a straight line.
- It is represented by the equation: y = a * x + b, where:
  - y is the predicted value
  - x is the independent variable
  - a is the slope of the line
  - b is the y-intercept
- *Least squares fitting* finds the values of a and b that minimize the sum of squared errors between the predicted and actual values.



Example of Linear Regression in Python

- This slide showcases a Python code example for linear regression using the `stats` module of the `scipy` library.
- The code creates sample data, builds a linear regression model, fits the model to the data, and predicts new values.
- It also visualizes the data and the best-fit line.

jupyter LINEAR Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help — Trusted

Markdown — JupyterLab — Python 3 (ipykernel)

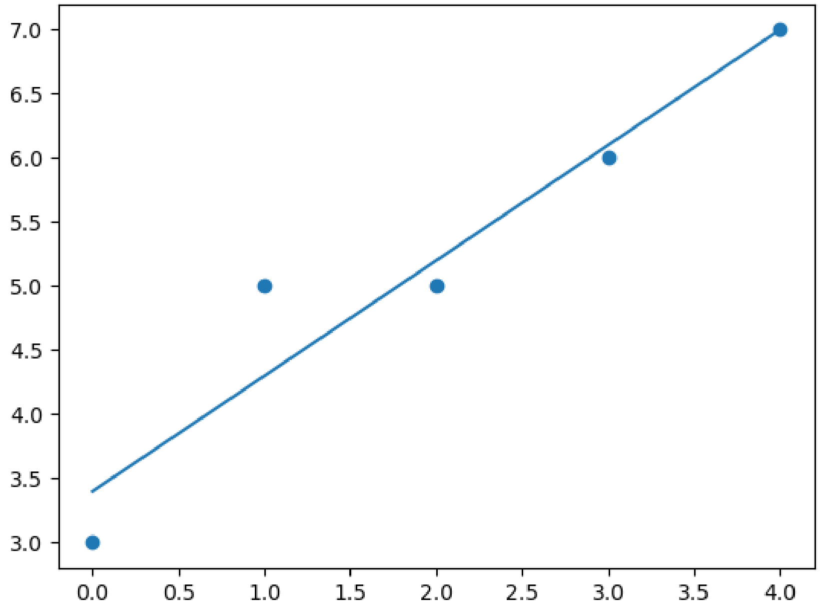
EXAMPLE 3.15 THE LINEARR.PY PROGRAM

```
[ ]:
#Example 3.15 Linear Regression
import matplotlib.pyplot as plt
from scipy import stats
x = [0,1,2,3,4]
y = [3,5,5,6,7]
slope, intercept, r, p, std_err = stats.linregress(x, y)
print("slope: ", slope)
print("intercept: ", intercept)
def myfunc(x):
    return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Linear Regression Example (Output)

- Lastly, it uses the functions `plt.scatter()` and `plt.show()` to display and plot the original x and y values and the best-fitted line.

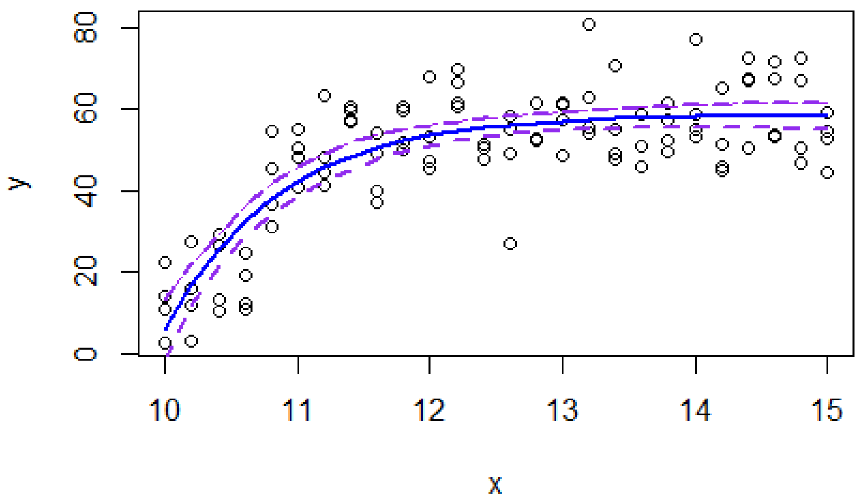
```
slope:  0.9
intercept:  3.4000000000000004
```



[ ]:

Nonlinear Regression

- Nonlinear regression models more complex relationships between variables using functions other than straight lines.
- Examples of nonlinear models include *polynomials*, *exponentials*, and *logistic* functions.
- Choosing the appropriate model depends on the nature of the data and the relationship you want to represent.



Polynomial Regression

- While linear regression excels at modeling linear relationships, real-world data often exhibits more intricate, curved patterns.
- Polynomial regression captures complex relationships beyond straight lines.
- Uses polynomial functions of different degrees.
- Choosing the right degree requires careful consideration of (a) the data, and (b) the desired level of detail.
- It has potential to unlock deeper insights from data that exhibit nonlinear trends.

Example of Polynomial Regression

- This example illustrates a two-dimensional (x and y) Python polynomial regression.
- It performs the polynomial regression function by calling

  `np.poly1d(np.polyfit(x, y, 3))`

- where the number 3 means three terms of a polynomial function, which is $y = ax^3 + bx^2 + cx + d$.

jupyter POLYR Last Checkpoint: 2 minutes ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

EXAMPLE 3.16 THE POLYR.PY PROGRAM

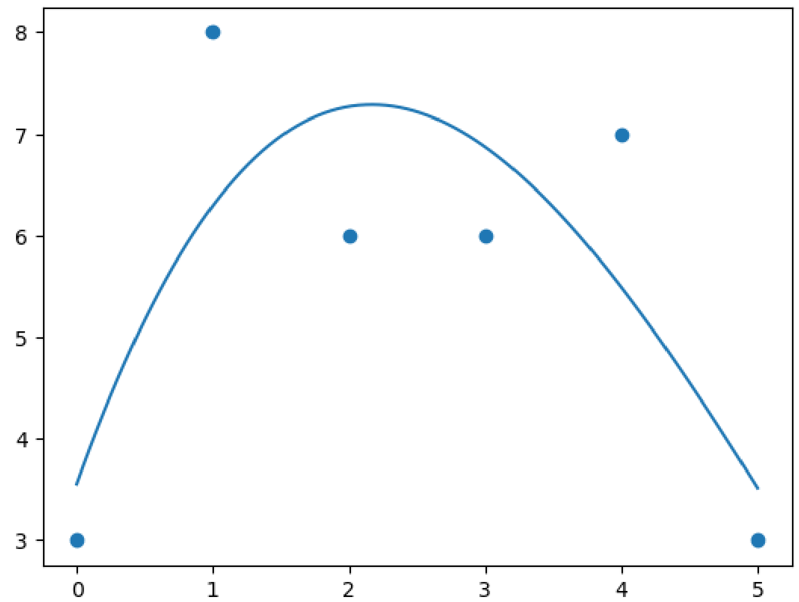
[ ]:

```
#Example 3.16 Polynomial Regression
import matplotlib.pyplot as plt
from scipy import stats
import numpy as np
x = [0,1,2,3,4,5]
y = [3,8,6,6,7,3]
mymodel = np.poly1d(np.polyfit(x, y, 3))
print(mymodel)
myline = np.linspace(0, 5, 100)
plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Polynomial Regression Example (Output)

- The following is the program output, the slope, and intercepts values.
- It shows the plot of the program; round dots are the x and y values, and the curved line is the best polynomial curve.

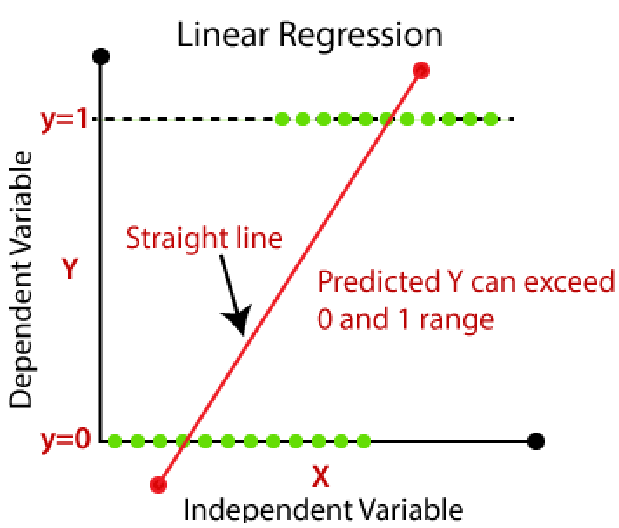
$$0.06481 x^3 - 1.075 x^2 + 3.749 x + 3.556$$



[ ]:

Logistic Regression

- Predicts binary outcomes (0 or 1): Classifies data points into two categories, such as "win/lose" or "spam/not spam."
- Uses the sigmoid function: A S-shaped function that transforms continuous input values into probabilities between 0 and 1, representing the likelihood of belonging to each category.



Ideal for classification tasks: Widely used in areas like *fraud detection*, *credit risk analysis*, and *sentiment analysis*.

Example of Logistic Regression

jupyter LOGR Last Checkpoint: 53 seconds ago

File Edit View Run Kernel Settings Help Trusted

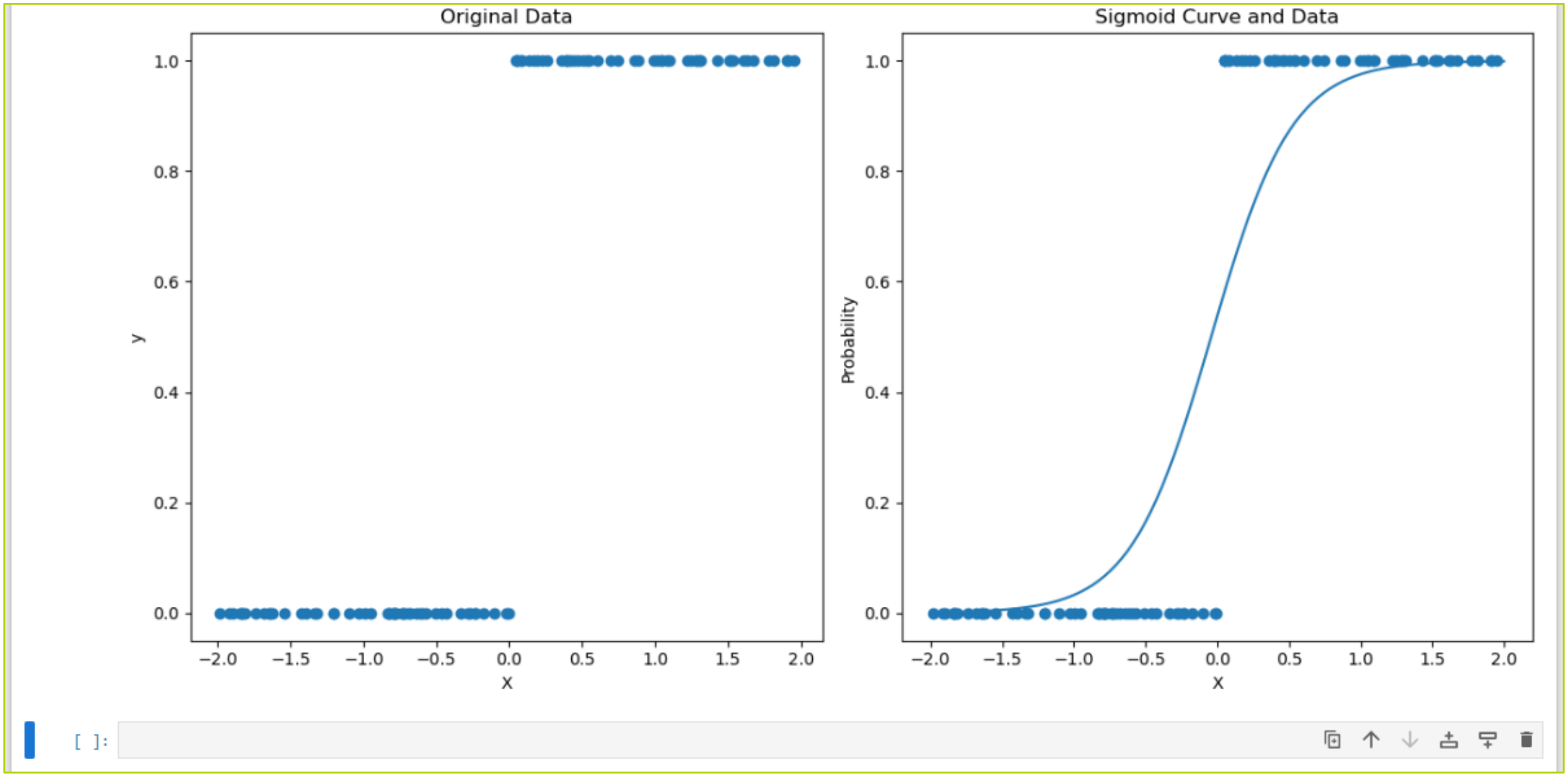
Markdown JupyterLab Python 3 (ipykernel)

EXAMPLE ON LOGISTIC REGRESSION

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# Generate sample data
np.random.seed(10)
X = np.random.rand(100, 1) * 4 - 2  # Create random x values between -2 and 2
y = (np.sin(X) > 0).astype(int)  # Generate binary labels based on sine function

# Create logistic regression model
model = LogisticRegression()
model.fit(X, y)

# Generate x values for plotting the sigmoid curve
x_plot = np.linspace(-2, 2, 100)

# Calculate predicted probabilities for the plot
y_pred = model.predict_proba(x_plot.reshape(-1, 1))[:, 1]

# Create the plot
plt.figure(figsize=(12, 6))

# Left plot: Original data
plt.subplot(1, 2, 1)
plt.scatter(X, y)
plt.xlabel("X")
plt.ylabel("y")
plt.title("Original Data")

# Right plot: Sigmoid curve and data
plt.subplot(1, 2, 2)
plt.plot(x_plot, y_pred)
plt.scatter(X, y)
plt.xlabel("X")
plt.ylabel("Probability")
plt.title("Sigmoid Curve and Data")

plt.tight_layout()
plt.show()
```

Logistic Regression Example (Output)

- The following is the program output, the slope, and intercepts values.
- It shows the plot of the program; round dots are the x and y values, and the curved line is the best polynomial curve.

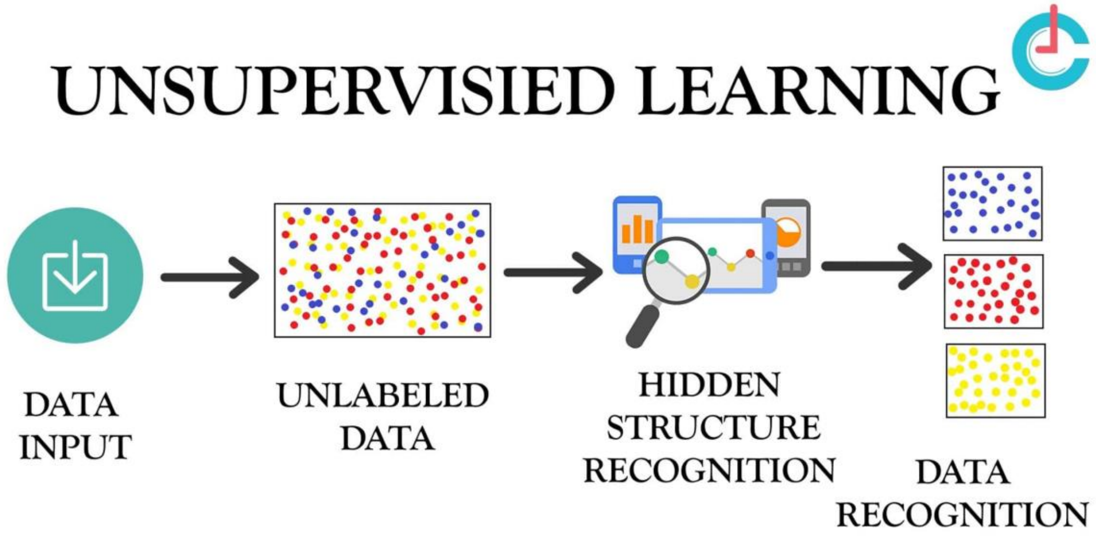


Unsupervised Learning

- Introduction to Unsupervised Learning
- Unsupervised vs. Supervised Learning: *Key Differences*
- Applications of Unsupervised Learning
- K-means Clustering

Introduction to Unsupervised Learning

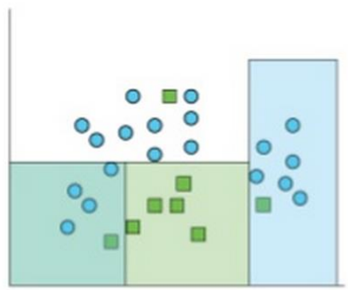
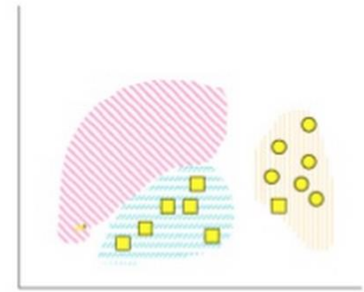
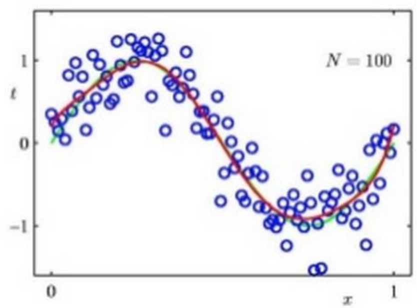
- Involves training models on ***unlabeled*** data, aiming to discover patterns, reduce dimensionality, or perform clustering without explicit guidance.
- Common techniques include clustering (e.g., K-Means), dimensionality reduction (e.g., PCA), anomaly detection, and association rule learning.
- Used in customer segmentation, anomaly detection, recommendation systems, and image/text clustering.
- Evaluation can be difficult due to the lack of labels, scalability issues, interpretability challenges, and dependency on data quality.



Key Differences

- Data:
  - Supervised: Labeled data (with predefined categories)
  - Unsupervised: Unlabeled data (without predefined categories)
- Learning Approach:
  - Supervised: Learns from labeled examples to make predictions
  - Unsupervised: Discovers patterns and relationships in data on its own
- Common Applications:
  - Supervised: Classification, regression, forecasting
  - Unsupervised: Clustering, dimensionality reduction, anomaly detection

Key Differences (cont…)

Predictive methods	Descriptive methods
Classification  Learns a method for predicting the instance class from pre-labeled (classified) instances	**Clustering**  Finds "natural" grouping of instances given un-labeled data
Regression  An attempt to predict a continuous attribute	**Association Rules**  Method for discovering interesting relations between variables in large DBs

Applications of Unsupervised Learning

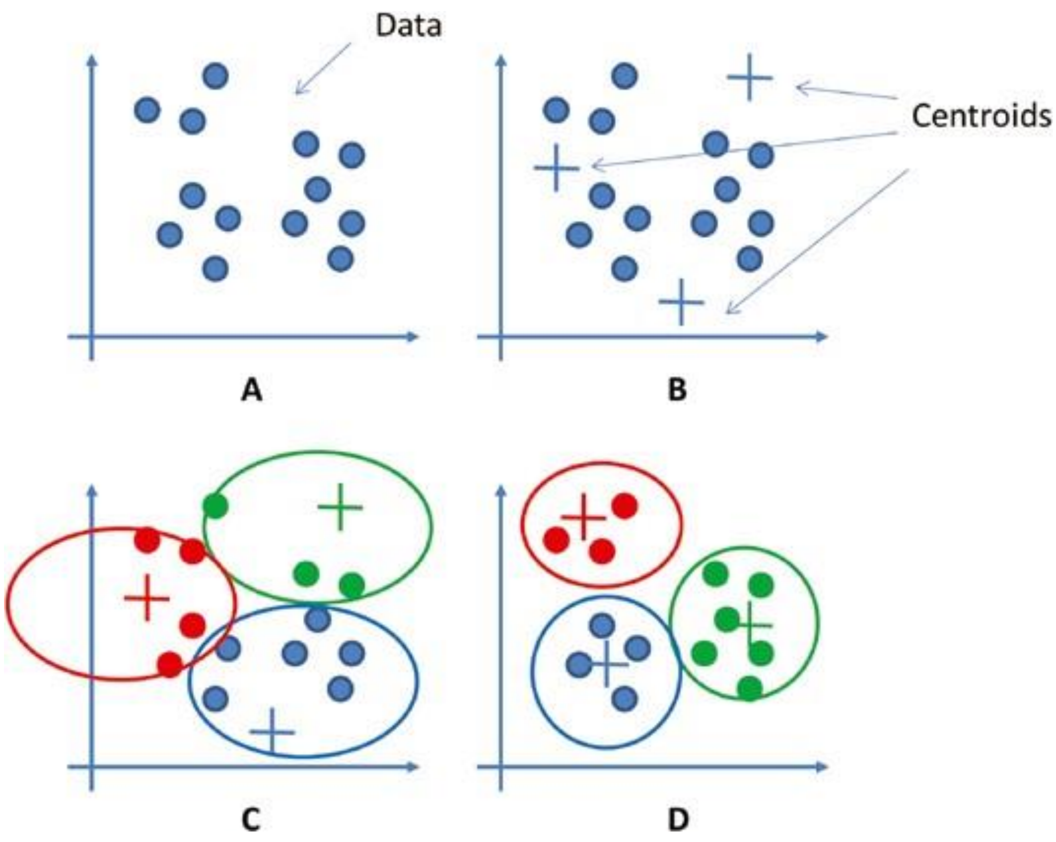
- **Customer Segmentation**: Group customers with similar characteristics for targeted marketing and personalized recommendations.
- **Image Recognition**: Identify objects and scenes in images without labeled training data.
- **Recommendation Systems**: Suggest relevant products, movies, or content based on user preferences and behavior.
- **Anomaly Detection**: Flag suspicious activity or fraud by identifying data points that deviate from normal patterns.
- **Document Clustering**: Organize large collections of documents based on their content and keywords.
- **Social Network Analysis**: Identify communities and influencers within social networks.
- **Fraud Detection**: Analyze financial transactions to detect fraudulent patterns.
- **Scientific Discovery**: Uncover hidden relationships and patterns in scientific data.

K-means Clustering

- Iterative algorithm that groups data points into K predefined clusters
- Steps:
  - Randomly select K cluster centers (centroids)
  - Assign each data point to the closest centroid
  - Recalculate centroids based on the assigned data points
  - Repeat steps 2 and 3 until convergence
- Advantages:
  - Simple and efficient
  - Easy to interpret
- Disadvantages:
  - Requires specifying the number of clusters (K)
  - Sensitive to outliers

K-means Clustering (cont…)

- Steps (A-D) of K-Means clustering



Example of K-means Clustering

jupyter KMEANS Last Checkpoint: 2 hours ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

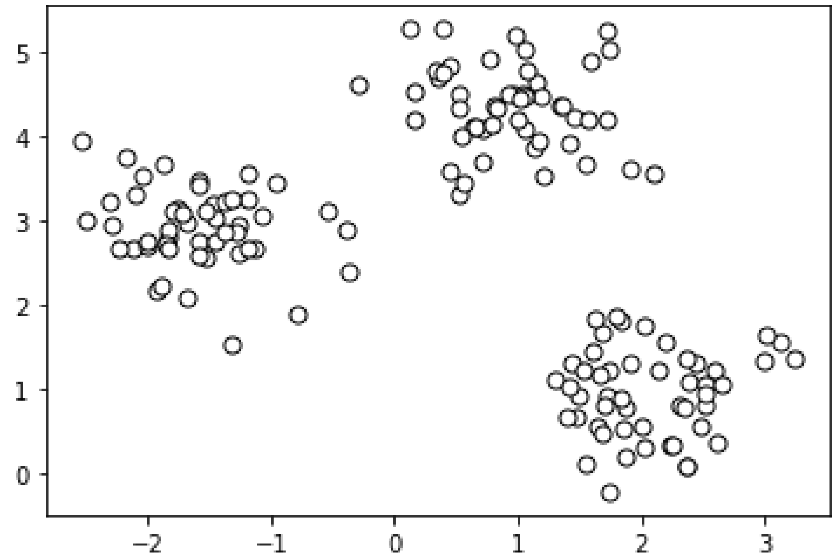
Trusted | Python 3 (ipykernel)

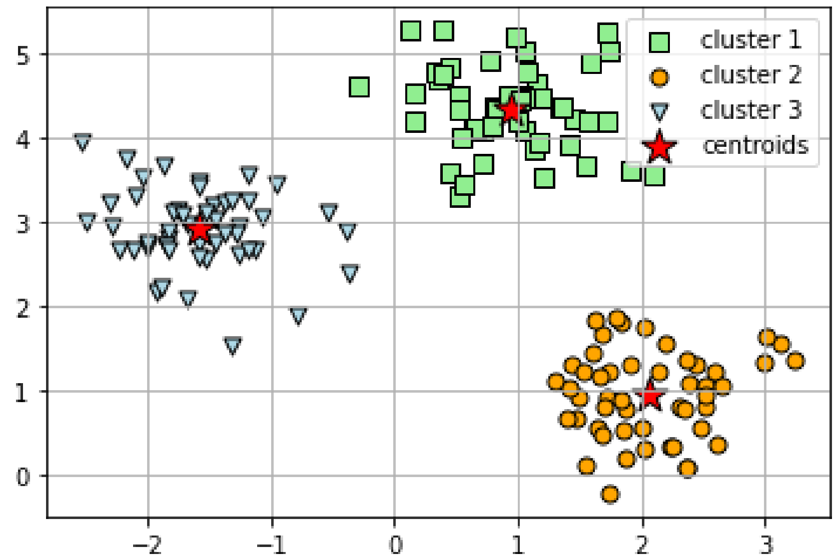
Run | Code

In [ ]:

```python
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# create dataset
X, y = make_blobs(
   n_samples=150, n_features=2,
   centers=3, cluster_std=0.5,
   shuffle=True, random_state=0
)

# plot
plt.scatter(
   X[:, 0], X[:, 1],
   c='white', marker='o',
   edgecolor='black', s=50
)
plt.show()

km = KMeans(
    n_clusters=3, init='random',
    n_init=10, max_iter=300,
    tol=1e-04, random_state=0
)
y_km = km.fit_predict(X)

# plot the 3 clusters
plt.scatter(
    X[y_km == 0, 0], X[y_km == 0, 1],
    s=50, c='lightgreen',
    marker='s', edgecolor='black',
    label='cluster 1'
)

plt.scatter(
    X[y_km == 1, 0], X[y_km == 1, 1],
    s=50, c='orange',
    marker='o', edgecolor='black',
    label='cluster 2'
)

plt.scatter(
    X[y_km == 2, 0], X[y_km == 2, 1],
    s=50, c='lightblue',
    marker='v', edgecolor='black',
    label='cluster 3'
)

# plot the centroids
plt.scatter(
    km.cluster_centers_[:, 0], km.cluster_centers_[:, 1],
    s=250, marker='*',
    c='red', edgecolor='black',
    label='centroids'
)
plt.legend(scatterpoints=1)
plt.grid()
plt.show()
```

K-means Clustering Example (Output)





Explanation of Python code for K-Means Clustering

1. Import Libraries:

- matplotlib.pyplot as plt: Used for creating visualizations (plots).
- from sklearn.datasets import make_blobs: Imports the make_blobs function for generating sample data with clusters.
- from sklearn.cluster import KMeans: Imports the KMeans class for performing KMeans clustering.

2. Create Sample Dataset:

- X, y = make_blobs(...): This line generates a sample dataset using the make_blobs function. Here's what the parameters control:
  - n_samples=150: Creates 150 data points.
  - n_features=2: Each data point will have 2 features (think of X and Y coordinates).
  - centers=3: Creates 3 clusters in the data.
  - cluster_std=0.5: Controls the spread of data points within each cluster (higher value increases spread).
  - shuffle=True: Randomly shuffles the data points.
  - random_state=0: Sets a seed for reproducibility (ensures the same data generation each time).

3. Visualize Dataset:

- plt.scatter(...): Creates a scatter plot of the generated data points.
  - X[:, 0]: Selects the first feature (X-coordinate) from all data points (represented by ':').
  - X[:, 1]: Selects the second feature (Y-coordinate) from all data points.
  - c='white': Sets the marker color to white.
  - marker='o': Sets the marker shape to circles.
  - edgecolor='black': Sets the edge color of the markers to black.
  - s=50: Sets the size of the markers to 50 points.
  - plt.show(): Displays the generated scatter plot.

Explanation of Python code for K-Means Clustering (cont…)

4. KMeans Clustering:

- km = KMeans(...): Creates a KMeans object with the following parameters:
  - n_clusters=3: Specifies the number of clusters to find (matches the number of centers in the data).
  - init='random': Initializes the centroids (cluster centers) randomly.
  - n_init=10: Runs the KMeans algorithm 10 times with different random initializations (helps find a better solution).
  - max_iter=300: Sets the maximum number of iterations allowed for the algorithm.
  - tol=1e-04: Sets the tolerance level for convergence (algorithm stops if changes in centroids are smaller than this value).
  - random_state=0: Sets a seed for reproducibility (ensures consistent cluster assignments).
- y_km = km.fit_predict(X):
  - fit(X): Trains the KMeans model on the data X. This process involves assigning data points to their closest centroids and iteratively updating the centroids based on these assignments.
  - predict(X): Predicts the cluster labels for each data point in X. The output (y_km) is an array where each element represents the cluster number (0, 1, or 2) assigned to the corresponding data point in X.

5. Visualize Clusters and Centroids:

- Three plt.scatter calls: These create scatter plots for each cluster, differentiated by color, marker shape, and label. The code uses conditional indexing (e.g., X[y_km == 0, 0]) to select data points belonging to each cluster based on their predicted labels (y_km).
- plt.scatter: This creates a scatter plot for the centroids (cluster centers) identified by the KMeans algorithm.

6. Display the plot:

- plt.legend(scatterpoints=1): Adds a legend to the plot, including the markers for clusters and centroids.
- plt.grid(): Adds a grid to the plot for better visualization.
- plt.show(): Displays the final plot showing the data points colored by their assigned clusters and the centroids marked with stars.

THANK YOU

TIME FOR DISCUSSION & QUESTIONS