

Overview

- 1 Feature Engineering
- 2 Dimensionality Reduction
 - Principal Component Analysis
 - t-Distributed Stochastic Neighbor Embedding (t-SNE)
- 3 Working with Language
 - NLP, Tokens, Tokenization, Encoding, Embeddings
 - Bag-of-Words, TF-IDF and Word2Vec
- 4 Appendix A: Data Organization, Libraries and Frameworks
- 5 Appendix B: Sentiment Analysis of Movie Reviews
- 6 Appendix C: Maximum Likelihood Estimation (MLE)
- 7 Appendix D: Cheatsheets

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Data-Driven Infrastructure Development

From an infrastructure standpoint, we seek:

- Data-driven approaches to measurement of performance in the building environment and identification of trends and patterns in behavior.
- Solutions that account for unique physical, economic, social and cultural characteristics of individual cities.

Sources of Complication:

- Multiple domains; multiple types of data and information.
- Network structures that are spatial and interwoven.
- Behaviors that are distributed and concurrent.
- Many interdependencies among coupled urban subsystems.

- Study of data to extract meaningful insights relevant to an application domain.
- End-to-end process of going from messy data to knowledge and actionable insights.
- Data science combines domain expertise, programming skills and knowledge of mathematics and statistics to extract meaningful insight from noisy, structured and unstructured data.
- Data science is the process of using advanced analytics to extract valuable information from data for decision making, planning and improvement of operations.

Core Data Science Activities

Core Activities:

- **Collection:** Gathering data from various sources and formats (e.g., csv, txt, xlsx, html, xml).
- **Cleaning:** Identifying and correcting errors, inconsistencies, and missing values in a dataset.
- **Exploratory Analysis:** Investigating datasets to summarize their main characteristics, often with visual methods.
- **Feature Engineering:** Create new features from existing data to improve model performance.
- **Model Building:** Developing and training machine learning and statistical models to make predictions or classify data.

Core Data Science Activities

- **Model Evaluation:** Assessing the performance of a model to ensure it meets the project goals.
- **Data Visualization:** Creating charts and graphs to communicate findings and insights.
- **Deployment:** Implement a model so it can be used in a real-world application.

Challenges:

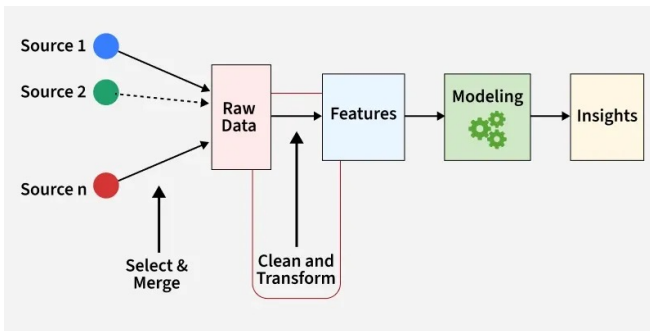
- Traditional databases are being replaced by cloud computing (i.e., large-scale distributed computing).

- Feature Engineering
 - Extract-Transform-Load (ETL) Processes
- Working with Language
 - Tokens and Word Tokens
 - One-Hot and Byte Pair Encoding Techniques
 - Embedding Models/Vectors.
- Dimensionality Reduction
 - Principal Component Analysis (PCA)
 - t-Distributed Stochastic Neighbor Embedding (t-SNE)
- Data Organization
 - Sample, Batch size, and Epochs
- Natural Language Processing
 - Bag of Words, TF-IDF and Word2Vec

Feature Engineering

Feature Engineering

Feature Engineering is the process of transforming raw data into meaningful inputs (features) that machine learning models can understand and make predictions.



ETL Data Transformation Operations

Data Transformation Operations

- Read data in a variety of formats (e.g., csv, text).
- Find and remove duplicate values.
- Remove unnecessary columns; rename columns.
- Filter data to keep only specific values (e.g., "MD" or "VA").
- Conditionally replace invalid values with new values computed by an external function.
- Convert categorical data into integers and one-hot encodings.
- Extract lower-level detail (e.g., day, hr, min) from string.
- Transfer data to dataset with a function/mapping.

ETL in Pandas

Example 1:

• • • •

Output:

• • • •

Source Code: See: [python-code/pandas/](#)

Example 2:

• • • •

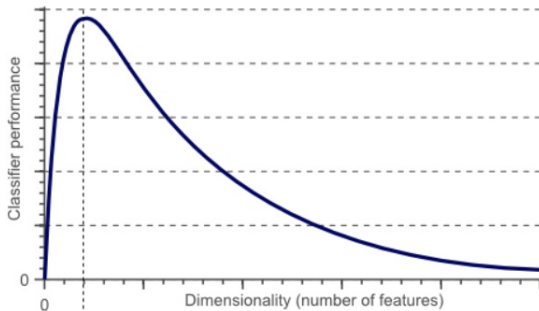
Output:

• • • •

Dimensionality Reduction

Curse of Dimensionality

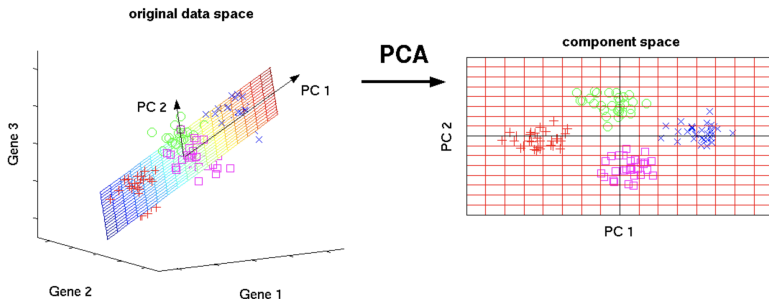
Machine learning problems are inherently statistical and involve high-dimensional data. **Increases** in the **problem dimensionality**, **decrease** the number of data points available for **classification** in each **dimension**.



Principal Component Analysis

Principal Component Analysis

Principal component analysis is an orthogonal linear transformation that transforms mean-centered data into a new coordinate system such that the variance of the projected data is maximized along the new axes – the latter is called the first principal component.



1. *Journal of Management Studies*, 1990, 27, 1, 1-14.

- The **first principal component** can be defined as the direction that **maximizes** the **variance** of the projected data.
- The **i-th principal component** is a direction that **maximizes** the **variance** in the projected data and is **orthogonal** to the **first i-1 principal components**.

- Exploratory data analysis.
- Dimensionality reduction.

Note: In general, dimensionality reduction loses information.
PCA-based reduction procedures tend to minimize information loss.

Principal Component Analysis

Mathematical Procedure: Suppose that our dataset comprises n m -dimensional data points:

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \quad (1)$$

Step 1. Compute the mean value for each dimension in the dataset:

$$\bar{X} = \begin{bmatrix} \bar{x}_1 & \bar{x}_2 & \cdots & \bar{x}_m \end{bmatrix} \quad (2)$$

$$\text{Cov} = \begin{bmatrix} \text{COV}_{11} & \text{COV}_{12} & \cdots & \text{COV}_{1m} \\ \text{COV}_{21} & \text{COV}_{22} & \cdots & \text{COV}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \text{COV}_{m1} & \text{COV}_{m2} & \cdots & \text{COV}_{mm} \end{bmatrix} \quad (3)$$

$$cov_{ij} = \frac{1}{n-1} \sum_{k=1}^n (x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j) \quad (4)$$

1. *Journal of the American Medical Association*, 2000; 283: 2689-2696.

$$[\text{Cov}] W = \lambda W. \quad (5)$$

Principal Component Analysis

Example 1: Define straight line segment + noisy data:

```
def NoisyLineFunction (a,b,x):
    return a + b*x + 5*(random.random() - 1.0)
```

Generate and Plot Raw Data: (x,y) coordinates:

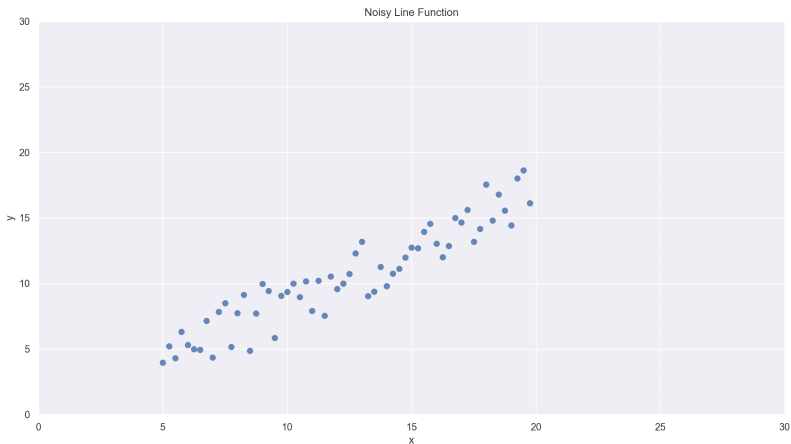
5.00	3.96
5.25	5.22
5.50	4.30
5.75	6.32

... data values removed ...

19.00	14.44
19.25	18.02
19.50	18.62
19.75	16.12

Principal Component Analysis

Two-dimensional plot of raw data:



Principal Component Analysis

Compute Principal Components (No components = 2):

```
pca = PCA(n_components=2)
pca.fit(X)
```

Mean values; Eigenvalue and Eigenvectors

```
--- Print mean ...
```

[12.375 10.5313869]

```
--- Print components (first and second eigenvectors) ...
```

```
[ [ 0.75668904  0.65377496]      <-- first eigenvector ...
  [-0.65377496  0.75668904] ]    <-- second eigenvector ...
```

```
--- Print variance ...
```

```
[32.50694888  1.05218475]
```

Principal Component Analysis

Principal Components in Two Dimensions:



Principal Component Analysis

Dimensionality Reduction Platform (one dimension)

```
pca = PCA(n_components=1)
pca.fit(X)
X_pca = pca.transform(X)

print("--- original shape:  ", X.shape)
print("--- transformed shape:", X_pca.shape)

# Compute inverse transform on reduced data ...

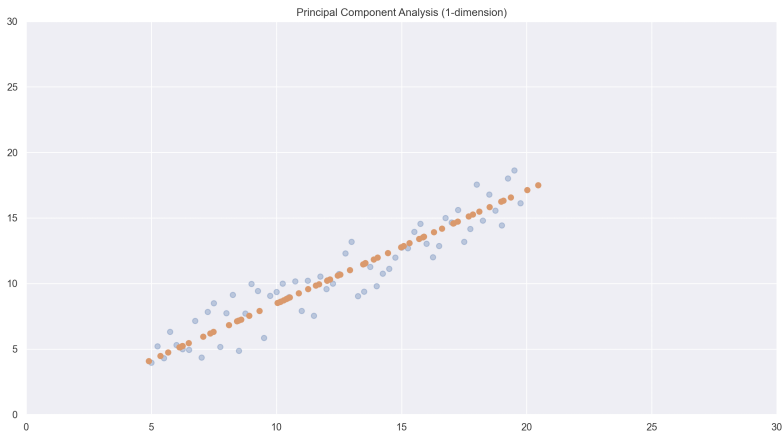
X_new = pca.inverse_transform(X_pca)
```

Shape of Original and Transformed Data

```
--- original shape: (60, 2)
--- transformed shape: (60, 1)
```

Principal Component Analysis

Principal Component Analysis in One Dimension:



Principal Component Analysis

Components and Variance (as above)

```
--- Print components ...  
[[0.75668904 0.65377496]]
```

```
--- Print variance ...  
[32.50694888]
```

Side-by-Side Comparison of Coordinates

2D Coords (x,y) --> 1D Coord System --> Inverse Transform Coords

=====					
5.00	3.96	-9.88	4.90	4.07	
5.25	5.22	-8.87	5.67	4.74	
5.50	4.30	-9.27	5.36	4.47	
....					
19.00	14.44	7.57	18.10	15.48	
19.25	18.02	10.10	20.02	17.13	
19.50	18.62	10.68	20.46	17.51	
19.75	16.12	9.23	19.36	16.57	
=====					

Principal Component Analysis

Strengths:

- Principal component analysis learns the **linear transformation** of data.
- Principal component analysis represents data in lower dimensions via an **optimal orthogonal transformation**.
- Implementations are fast.

Weaknesses:

- Principal component analysis learns the **linear transformation** of data.
- As the number of features increases, the chances of overfitting of the model decreases significantly.

Source Code: See: python-code.d/reduction/

t-Distributed Stochastic Neighbor Embedding (t-SNE)

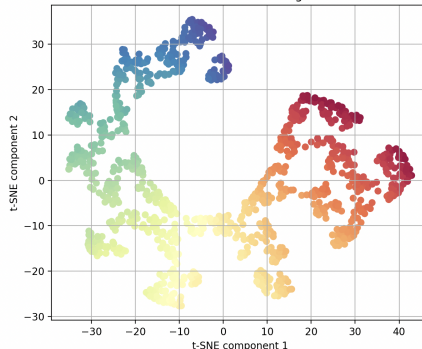
t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-distributed stochastic neighbor embedding (t-SNE) is a non-linear dimensionality reduction technique primarily used for visualizing high-dimensional datasets in 2D or 3D space.

Unlike linear techniques like PCA, t-SNE excels at **preserving the local structure of data**, meaning it keeps similar data points close together in the lower-dimensional representation while ensuring **dissimilar points are far apart**.

Example 1: Swiss Roll

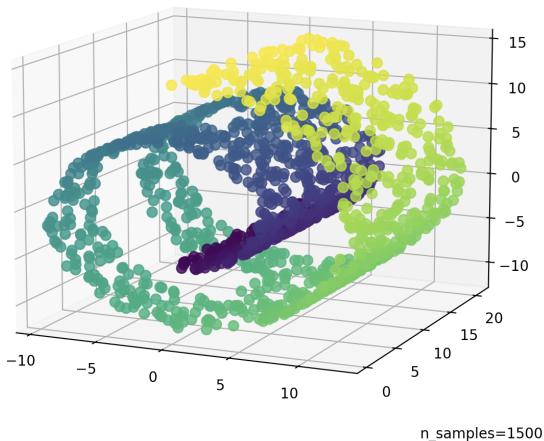
t-SNE 2D Embedding



Source: python-code-ai.d/reduction/TestTSNE-SwissRollExample01.py

t-Distributed Stochastic Neighbor Embedding (t-SNE)

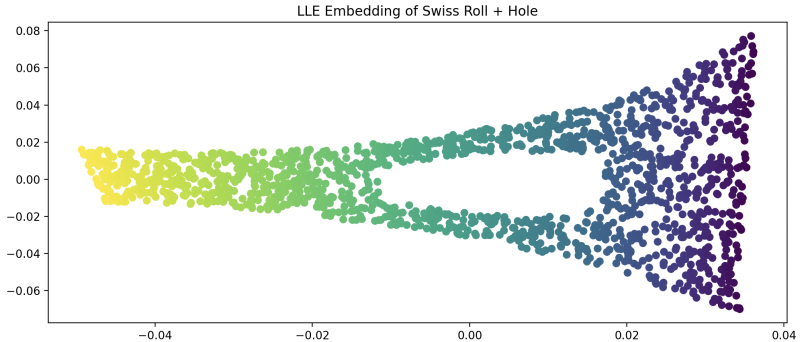
Example 2: Swiss Roll + Hole



Source: [python-code-ai.d/reduction/TestTSNE-SwissRollExample02.py](https://github.com/python-code-ai/d/reduction/TestTSNE-SwissRollExample02.py)

t-Distributed Stochastic Neighbor Embedding (t-SNE)

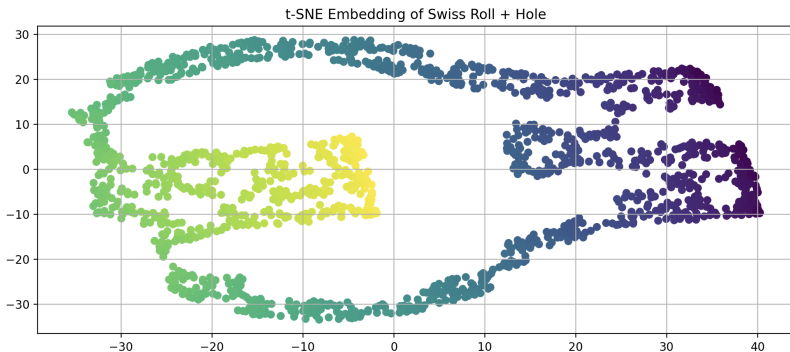
Example 2: Continued ...



Source: python-code-ai.d/reduction/TestTSNE-SwissRollExample02.py

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Example 2: Continued ...



Source: python-code-ai.d/reduction/TestTSNE-SwissRollExample02.py

Language

(Tokens, Tokenization, Encoding, Embeddings)

Natural Language Processing

Natural Language Processing

Natural Language Processing (NLP) is a branch of AI that helps computers understand, process and generate human language (text or speech).

NLP combines linguistics, machine learning, and deep learning to work with natural language.

Importance of NLP:

- Helps machines understand human communication.
- Used in many real-world applications.
- Makes interaction with machines more natural.

Natural Language Processing

Basic Flow:

- **Input:** Text and speech.
- **Text Preprocessing:** Clean text, tokenize, remove stop words, convert to features.
- **Model:** Machine learning, deep learning, transformers and attention.
- **Output:** Sentiment (positive or negative).
- **Applications:** Classification, translations, chatbots, question/answering, story generation.

Tokens

Token

In AI, a **token** is a **fundamental unit of data** processed by language models, such as ChatGPT. Instead of reading whole words, AI breaks text down into small units (e.g., an entire word, part of a word, punctuation, a single character).

Why Tokens Matter:

- Businesses pay for AI usage based on input/output tokens.
- AI models can only remember or process a certain amount of information at once (also called the context window). This limit is measure in terms of tokens, not words.
- The output speed of AI is measured in terms of the number of tokens it can generate per second.

Tokens

Simple Example:

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ▶ ↺ 🔍 ↻

One-Hot Encoding

- One hot encoding is one method of converting data to prepare it for an algorithm and get a better prediction.
- Each categorical value is converted into a new categorical column and assign a binary value of 1 or 0 to those columns.
- Each integer value is represented as a binary vector.

Example 1:

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Source: datascience.com

One-Hot Encoding

Example 2: Onehot encoding for colors ...

The diagram is divided into three vertical sections by two vertical lines:

- Section (a): Colors in our data**
 - red
 - yellow
 - blue
 - green
 - orange
 - brown
 - purple
 - black
- Section (b): Assignment of a number to each value**
 - red $\rightarrow$ 0
 - yellow $\rightarrow$ 1
 - blue $\rightarrow$ 2
 - green $\rightarrow$ 3
 - orange $\rightarrow$ 4
 - brown $\rightarrow$ 5
 - purple $\rightarrow$ 6
 - black $\rightarrow$ 7
- Section (c): One-hot encoding of each color**
 - red $\rightarrow$ [1, 0, 0, 0, 0, 0, 0, 0]
 - yellow $\rightarrow$ [0, 1, 0, 0, 0, 0, 0, 0]
 - blue $\rightarrow$ [0, 0, 1, 0, 0, 0, 0, 0]
 - green $\rightarrow$ [0, 0, 0, 1, 0, 0, 0, 0]
 - orange $\rightarrow$ [0, 0, 0, 0, 1, 0, 0, 0]
 - brown $\rightarrow$ [0, 0, 0, 0, 0, 1, 0, 0]
 - purple $\rightarrow$ [0, 0, 0, 0, 0, 0, 1, 0]
 - black $\rightarrow$ [0, 0, 0, 0, 0, 0, 0, 1]

Source: Chapter 10, Glassner, 2021.

1

```

1  # =====
2  # TestEncoder01.py: Manual one hot encoding with numpy
3  # =====
4
5  from numpy import argmax
6
7  print("TestEncoder01.py ... ")
8  print("=====")
9
10 # define input string
11
12 data = 'hello world'
13 print("--- Input string: %s" %(data))
14
15 # define universe of possible input values
16
17 alphabet = 'abcdefghijklmnopqrstuvwxyz '
18 print("--- Alphabet of input values: %s" %(alphabet))
19
20 # define a mapping of chars to integers
21
22 char_to_int = dict((c, i) for i, c in enumerate(alphabet))
23 int_to_char = dict((i, c) for i, c in enumerate(alphabet))
24
25 # integer encode input data
26
27 integer_encoded = [char_to_int[char] for char in data]
28 print("--- Integer encoded data: %s" %(integer_encoded))

```

One-Hot Encoding

Example 1: continued ...

```

29
30 # one hot encode
31
32 print("--- One hot encode ...")
33
34 onehot_encoded = list()
35 for value in integer_encoded:
36     letter = [0 for _ in range(len(alphabet))]
37     letter[value] = 1
38     onehot_encoded.append(letter)
39 print(onehot_encoded)
40
41 # invert encoding
42
43 print("--- Invert encoding ...")
44
45 print("--- Encoding[ 0] : %s" %( int_to_char[argmax(onehot_encoded[0])] ))
46 print("--- Encoding[ 1] : %s" %( int_to_char[argmax(onehot_encoded[1])] ))
47
48 .... lines of code deleted ...
49
50 print("--- Encoding[ 8] : %s" %( int_to_char[argmax(onehot_encoded[8])] ))
51 print("--- Encoding[ 9] : %s" %( int_to_char[argmax(onehot_encoded[9])] ))
52 print("--- Encoding[10] : %s" %( int_to_char[argmax(onehot_encoded[10])] ))
53
54 print("=====")
55 print("Finished !! ")

```

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

One-Hot Encoding

Example 2: One-Hot Encoding in Python

```

1 # =====
2 # TestEncoder02.py: One hot encoding with sklearn ...
3 #
4 # Modified by: Mark Austin                                December 2025
5 # =====
6
7 from numpy import array
8 from numpy import argmax
9 from sklearn.preprocessing import LabelEncoder
10 from sklearn.preprocessing import OneHotEncoder
11
12 # =====
13 # Main function ...
14 # =====
15
16 def main():
17     print("--- Enter TestEncoder02.main()                                ... ");
18     print("--- ===== ... ");
19     print("");
20
21     # Input string of temperature levels ...
22
23     data = ['freeze', 'cold', 'warm', 'cold', 'hot', 'burn',
24             'warm', 'burn', 'cold', 'warm', 'hot']
25
26     values = array(data)

```


1. **Introduction**

```

56     inverted00 = label_encoder.inverse_transform([argmax(onehot_encoded[0, :])])
57     inverted01 = label_encoder.inverse_transform([argmax(onehot_encoded[1, :])])
58     inverted02 = label_encoder.inverse_transform([argmax(onehot_encoded[2, :])])
59     inverted03 = label_encoder.inverse_transform([argmax(onehot_encoded[3, :])])
60
61     print("--- Inverted encoding [0]: {s} ...".format(str(inverted00)) )
62     print("--- Inverted encoding [1]: {s} ...".format(str(inverted01)) )
63     print("--- Inverted encoding [2]: {s} ...".format(str(inverted02)) )
64     print("--- Inverted encoding [3]: {s} ...".format(str(inverted03)) )
65
66     print("=====")
67     print("Finished !! ")
68
69     # call the main method ...
70
71     if __name__ == "__main__":
72         main()

```

Source Code: See: python-code-ai.d/encoder/

Abbreviated Output: ...

```
--- Part 01: Input values:
```

['freeze' 'cold' 'warm' 'cold' 'hot' 'burn' 'warm' 'burn' 'cold' 'warm' 'hot']

```
--- Part 02: Integer encoded ...
```

[2 1 4 1 3 0 4 0 1 4 3]

--- Part 03: Onehot binary encoding ...

[[0, 0, 1, 0, 0]]

[0. 1. 0. 0. 0.]

[0. 0. 0. 0. 1.]

[0. 1. 0. 0. 0.]

[0. 0. 0. 1. 0.]

```
[1.  0.  0.  0.  0.]
```

```
[0. 0. 0. 0. 1.]
```

```
[1. 0. 0. 0. 0.]
```

```
[0.  1.  0.  0.  0.]
```

```
[0. 0. 0. 0. 1.]
```

```
[0. 0. 0. 1. 0.]
```

--- Part 04: Retrieve and Invert Individual Encodings ...

```
--- Inverted encoding [0]: ['freeze'] ...
```

```
--- Inverted encoding [1]: ['cold'] ...
```

```
--- Inverted encoding [2]: ['warm'] ...
```

```
--- Inverted encoding [3]: ['cold'] ...
```

One-Hot Encoding

Example 3: Dataframe with one date column

Source Code:

Output:

```
data01 = { 'Date': pd.to_datetime( ['2023-01-15', '2023-02-20',
                                     '2024-01-10', '2024-04-10',
                                     '2024-06-12', '2025-08-12',
                                     '2025-10-01', '2025-03-05'] )}

df01 = pd.DataFrame(data01)
```

	Date
0	2023-01-15
1	2023-02-20
2	2024-01-10
3	2024-04-10
4	2024-06-12
5	2025-08-12
6	2025-10-01
7	2025-03-05

Solution Procedure:

- Load data, extract day, month, year and quarter from date.
- Onehot encoding for month and quarter columns.
- Change column names, convert month/quarter columns to binary.

One-Hot Encoding

Extract day, month, year and quarter from date:

Source:

```
df01['DayOfWeek'] = df01['Date'].dt.day_name()
df01['Month']      = df01['Date'].dt.month_name()
df01['Year']       = df01['Date'].dt.year
df01['Quarter']    = df01['Date'].dt.quarter
```

Output :

	Date	DayOfWeek	Month	Year	Quarter
0	2023-01-15	Sunday	January	2023	1
1	2023-02-20	Monday	February	2023	1
2	2024-01-10	Wednesday	January	2024	1
3	2024-04-10	Wednesday	April	2024	2
4	2024-06-12	Wednesday	June	2024	2
5	2025-08-12	Tuesday	August	2025	3
6	2025-10-01	Wednesday	October	2025	4
7	2025-03-05	Wednesday	March	2025	1

One-Hot Encoding

Onehot encoding for month and quarter columns: ...

Source:

```
df01 = pd.get_dummies( df01, columns = ['Month', 'Quarter'])
```

Output :

	Date	DayOfWeek	Year	...	Quarter_2	Quarter_3	Quarter_4
0	2023-01-15	Sunday	2023	...	False	False	False
1	2023-02-20	Monday	2023	...	False	False	False
2	2024-01-10	Wednesday	2024	...	False	False	False
3	2024-04-10	Wednesday	2024	...	True	False	False
4	2024-06-12	Wednesday	2024	...	True	False	False
5	2025-08-12	Tuesday	2025	...	False	True	False
6	2025-10-01	Wednesday	2025	...	False	False	True
7	2025-03-05	Wednesday	2025	...	False	False	False

Drop unnecessary columns ...

=====

```
to_drop = ['DayOfWeek', 'Year' ]
df01.drop( to_drop, inplace = True, axis = 1)
print( df01.head(10) );
```

	Date	April	August	February	January	...	October	Q1	Q2	Q3	Q4
0	2023-01-15	0	0	0	1	...	0	1	0	0	0
1	2023-02-20	0	0	1	0	...	0	1	0	0	0
2	2024-01-10	0	0	0	1	...	0	1	0	0	0
3	2024-04-10	1	0	0	0	...	0	0	1	0	0
4	2024-06-12	0	0	0	0	...	0	0	1	0	0
5	2025-08-12	0	1	0	0	...	0	0	0	1	0
6	2025-10-01	0	0	0	0	...	1	0	0	0	1
7	2025-03-05	0	0	0	0	...	0	1	0	0	0

[8 rows x 12 columns]

0

- Computational elements are **binary** (instead of ordinal) and sit in an **orthogonal vector space**.

0

- Some decision tree based methods can work directly with labeled entries – no need for one-hot encoding.
- For **high cardinality** the vector space can **quickly blow up**, leading to sparse representations and the **curse of dimensionality**.

- Apply one-hot encoding then reduce problem space with PCA (linear transformation) or t-SNE (nonlinear transformation).

Byte Pair Encoding

Byte Pair Encoding

Byte Pair Encoding (BPE) is a data compression algorithm that has become the **standard tokenization method** for **large language models**.

Benefits and Usage:

- The **algorithm** iteratively finds the most frequent pair of adjacent bytes or characters in a corpus of text and replaces them with a single new unused token. (i.e., the vocabulary size is smaller than word-level tokenization).
- Widely used in large language models like GPT-2, GPT-3 and Llama-3 to handle rare words by breaking them down into familia sub-components.

Byte Pair Encoding

Procedure: We begin with a corpus of words:

- Create base vocabulary (split words into individual characters).
- Find most frequent pairs.
- Merge and update.
- Repeat until target reached.

Byte Pair Encoding

Simple Example: ...

Embedding Models

Embedding Models

Embedding Models

Pathway of Development:



Convert Raw Data into Usable Embeddings:



- Embedding model must capture semantic meaning and context of the input data.
- High quality embedding ensure that semantically similar items are close together in the vector (embedding) space.

- Embeddings should be fine-tuned for specific tasks: sentiment analysis, code understanding and/or retrieval.

- Embedding should capture specific domain knowledge and be tuned to match query formats to document fragments.

Sparse Embedding Vectors

Weaknesses: Highly rigid. Methods fail to find matches if different words are used.

Sparse and Dense Embeddings

Dense Embeddings

Dense Embedding Vectors represent the learned semantic meaning of text compressed into a **fixed lower-dimensional space** (typically 100 to 1000 dimensions) where **almost every value** is **non-zero**.

How it Works: Similar concepts (e.g., tiger and animal) map close together in vector (embedding) space.

Use Cases: Conversational search, recommendation engines and understanding overall end-user intent.

Strengths: Superb at capturing context, synonyms and overarching themes.

Weaknesses: Can *hallucinate* incorrect information.

Text Embedding Methods

Static vs Contextual Embeddings:

Static Embeddings

"I went to the bank"

Vector [0.2, -0.5, 0.8, ...]

"The river bank was muddy"

Vector [0.2, -0.5, 0.8, ...]

"Bank of America"

Vector [0.2, -0.5, 0.8, ...]

Same vector for all contexts

Contextualized Embeddings

"I went to the bank"

Vector [0.8, 0.2, -0.3, ...]

"The river bank was muddy"

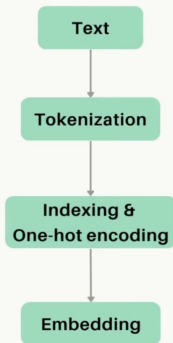
Vector [-0.3, 0.6, 0.4, ...]

"Bank of America"

Vector [0.7, -0.1, 0.5, ...]

Different vectors based on context

Lifecycle of text embeddings



["The earth is spherical.",
"The earth is a planet.",
"I like to eat at a restaurant."]

ID dictionary:

```
{'the': 1, 'earth': 2, 'is': 3, 'a': 4, 'spherical': 5, 'planet': 6,
'i': 7, 'like': 8, 'to': 9, 'eat': 10, 'at': 11, 'restaurant': 12}
```

ID sequences:

```
[[1, 2, 3, 5], [1, 2, 3, 4, 6], [7, 8, 9, 10, 11, 4, 12]]
```

One-hot encoded sequences:

```
[[0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 1. 0. 0.]]
```

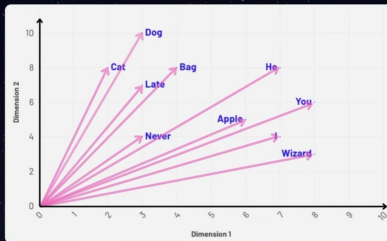
Dense Embeddings: Coffee, Tea and Pasta ...



One-Dimensional Dense Embedding:



Two-Dimensional Dense Embedding:



Three-Dimensional Dense Embedding:



Dimensionality Reduction:



One-Dimensional Dense Embedding:



One-Hot to Dense Embedding:



Dense Text Embeddings:



Sparse and Dense Embeddings

.... this apricot jam is made with sweet, ripe, juicy apricots at the height of their season.



Continuous Bag-of-Words (CBOW)



Skipgram

Embedding Model Data Types

Image Embeddings:

Embedding Model Data Types

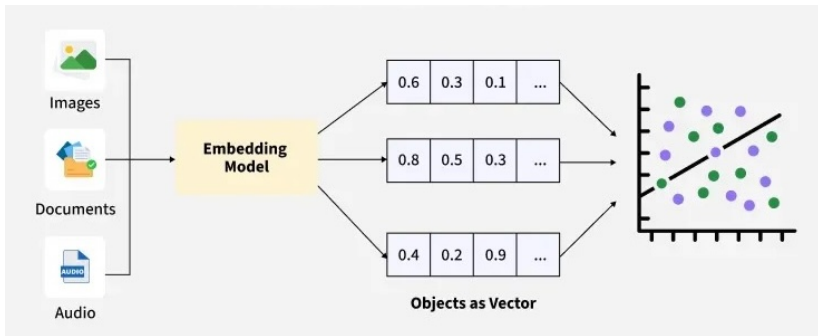
Audio/Speech Embeddings:

Embedding Model Data Types

Graph Embeddings:

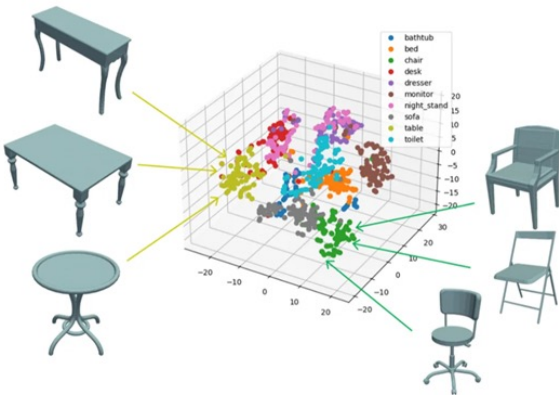
Embedding Model by Data Type

MultiModal Embedding Model:



Embedding Model Applications

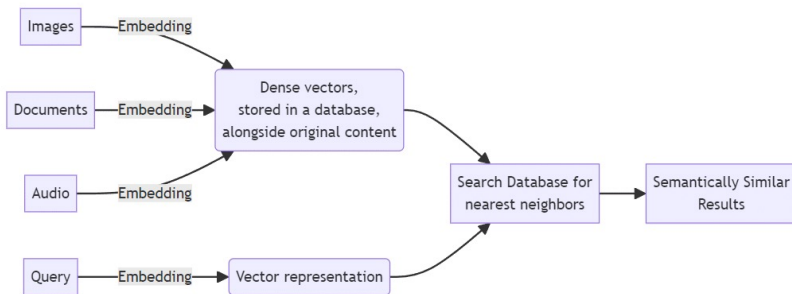
Application 1: Retrieve objects from embedding space:



Object retrieval in embedding space: similar objects are expected to have a small embedding distance.

Embedding Model Applications

Application 2: Search database for nearest neighbors:

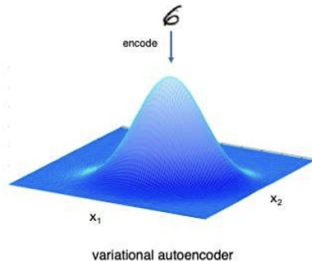
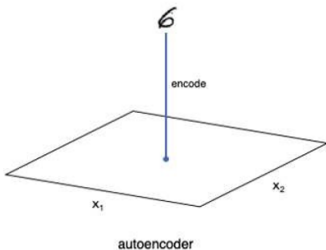


Application 2: Vector search data workflow:



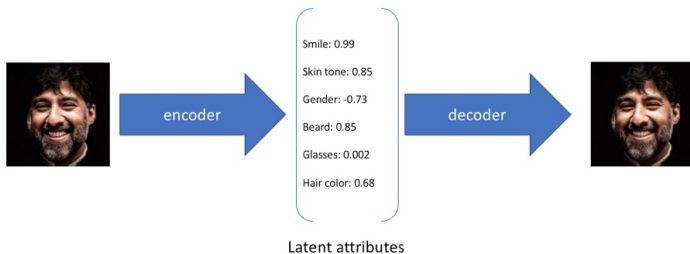
100

1. *Journal of the American Medical Association*, 1997; 278: 1039-1044.



Embedding Model Applications

Example: Single value representations for latent attributes:

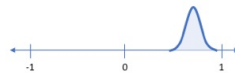
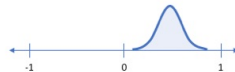


Embedding Models

Discrete Value and Probability Distribution: Representations for smile latent attribute:



VS.



Transformer-based Models:

- • •

Embedding Model Architecture

Sentence/Document Transformers:

- • •

Embedding Model Architecture

Static WordEmbedding Models:

- • •

Embedding Model Process

Embedding Vector Design Strategy:

- . . .
- . . .
- . . .

Bag-of-Words and TF-IDF

Bag-of-Words

A NLP technique that converts text into numerical vectors by counting word frequencies, disregarding order and context. The result is a vocabulary of unique words from a document set.

Applications:

- Text classification.
- Spam detection.
- Sentiment Analysis.

TF-IDF

TF-IDF (Term Frequency Inverse Document Frequency) is a numerical statistic used in NLP and information retrieval to measure a words importance in a document corpus. It increases with word frequency and decreases for words that are rarely used in the corpus.

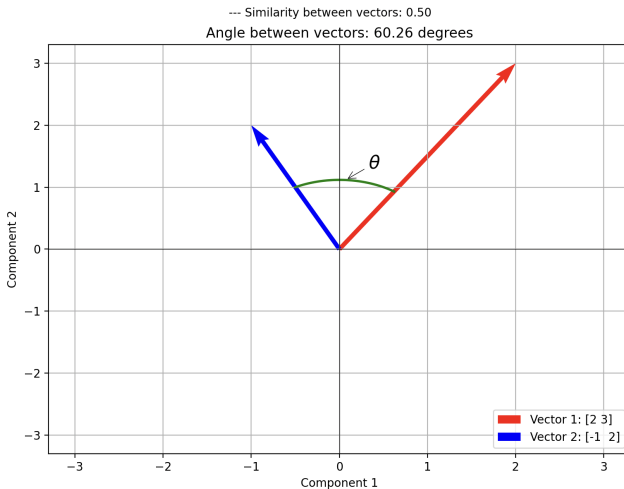
TF-IDF

Example 1: ...

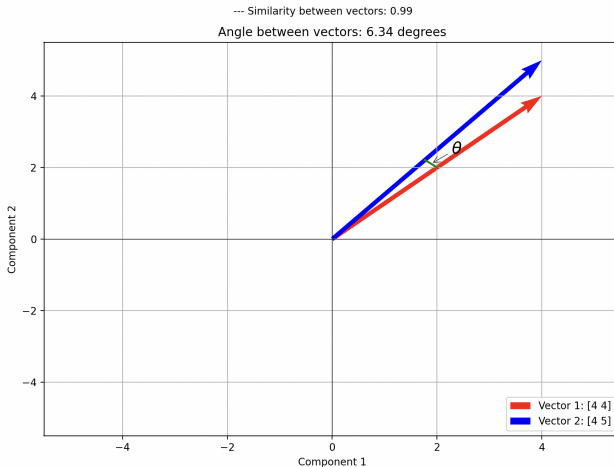
Word2Vec

Example 1: ...

Embedding Similarity



Embedding Similarity



A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

100

1. *Journal of Management Studies*, 1996, 33, 1, 1-14.

Downloaded from <http://ajph.org/> on November 10, 2015

Downloaded from <https://www.cambridge.org/core>. University of Cambridge, on 02 Jun 2019 at 12:00:00, subject to the Cambridge Core terms of use, available at <https://www.cambridge.org/core/terms>. <https://doi.org/10.1017/9781315336435.008>

Data Organization

Epochs and Batches

- Epochs are comprised of one or more batches.
- The number of epochs can be large, hundreds or even thousands.
- Some learning algorithms require that the batch size and number of epochs be specified upfront.

Common Batch Sizes in RNN

- Powers of two ...
- 32, 62, 128.

Epoch vs Batch Size

Data Science Libraries and Frameworks

Python Libraries and Frameworks:

- **Pandas:** Ideal for data manipulation and analysis, Pandas allows you to efficiently handle large datasets, perform complex transformations, and easily load data.
- **PySpark:** PySpark handles big data via an interface to Apache Spark, support for large-scale data processing across distributed computing environments.
- **BeautifulSoup:** Used for web scraping. Allows you to extract data from HTML and XML files.

Data Science Libraries and Frameworks

Java Libraries and Frameworks:

- . . .
- . . .
- . . .
- . . .

A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

Sentiment Analysis

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

1

- Create training data tensors.
- Initialize `SentimentRNN()` model.
- Plot embeddings (pre-training).
- Learning neural network (epochs = 5).
- Plot training history, confusion matrix, and embeddings (post-training).

Movie Review Dataset: Consists of 50,000 positive and negative reviews (on average, reviews are 230 words).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   review      50000 non-null   object
1   sentiment    50000 non-null   object
dtypes: object(2)
memory usage: 781.4+ KB
```

Sentiment Analysis of Movie Reviews

Processed IMDB Datfile:

```
--- X.shape: (50000,)
```

```
--- y.shape: (50000,)
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1
...	...	...
995	Nothing is sacred. Just ask Ernie Fosselius. T...	1
996	I hated it. I hate self-aware pretentious inan...	0
997	I usually try to be professional and construct...	0
998	If you like me is going to see this in a film ...	0
999	This is like a zoology textbook, given that it...	0

Split Data: Training, test and validation.

```
x_train.shape: (36000,)
```

```
x_test.shape: (10000,)
```

```
x_val.shape: (4000,)
```

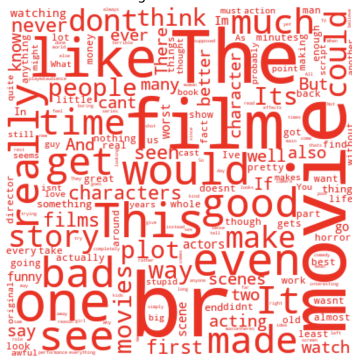
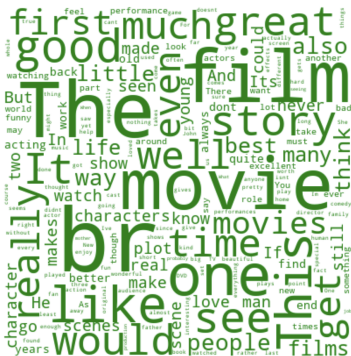
```
y_train.shape: (36000,)
```

```
y_test.shape: (10000,)
```

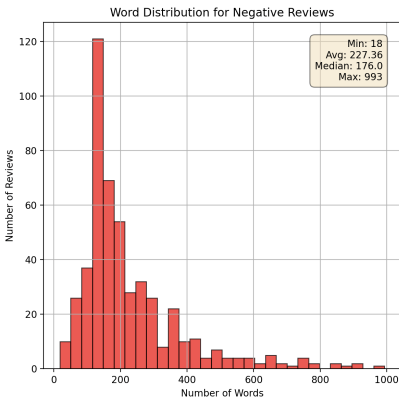
```
y_val.shape: (4000,)
```

Word Cloud (Most frequent words): Positive and negative reviews.

Negative Reviews



Distribution of Review Lengths: Positive and negative reviews ...



Distribution of Review Lengths: after tokenization ...



Pre-Training Embedding Projection: Positive/negative data points are mixed together.

1. *Journal of Management Studies*, 1997, 34, 1, 1-14.

DRAFT

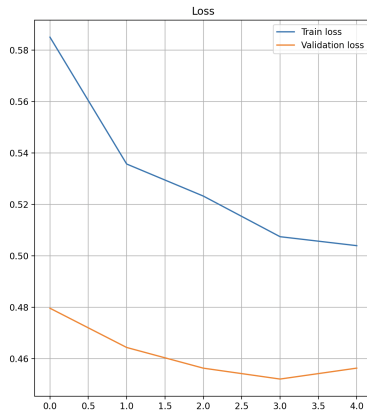
```

--- no_layers      = 3
--- embedding_dim  = 300
--- hidden_dim     = 256
--- output_dim     = 1

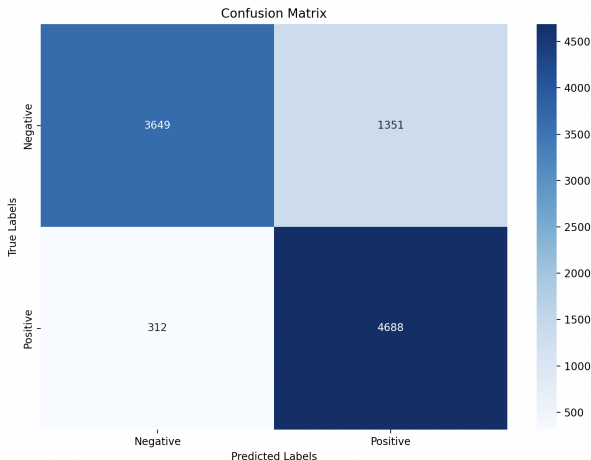
```

```
SentimentRNN(
  (embedding): Embedding(1001, 300)
  (rnn): GRU(300, 256, num_layers=3, batch_first=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=256, out_features=1, bias=True)
  (sig): Sigmoid()
)
```

Training History:



Test Set Confusion Matrix:



Post-Training Embedding Projection: Positive/negative data points are separated.



Maximum Likelihood Estimation

Maximum Likelihood Estimation

In statistics, **maximum likelihood estimation** (MLE) is a method for estimating the parameters of a probability distribution by maximizing the likelihood function, making the observed data most probable under the assumed model.

Key Concepts:

- Likelihood function $L(\theta)$: Probability of observing the given data as a function of the unknown parameter θ .
- Goal: Find the parameter value θ that maximizes $L(\theta)$.
- Log-Likelihood: Simplify calculations by working with $\ln-L(\theta)$.
- Distributions: MLE requires an assumed underlying probability distribution (e.g., normal, poisson).

Maximum Likelihood Estimation

Computational Process:

- Model Definition: Assume a probability density function for the data $f(x|\theta)$.
- Likelihood Function: Multiply individual probabilities for independent data points:

$$L(\theta) = \prod_{i=1}^n f(x_i|\theta). \quad (8)$$

- Take Logarithm:

$$\ln(L(\theta)) = \sum_{i=1}^n \ln(f(x_i|\theta)). \quad (9)$$

- Maximize: Take derivatives, maximize equation 9 wrt θ .

Maximum Likelihood Estimation

Key Characteristics:

- Consistency:
- Efficiency:
- Flexibility:

1. *Journal of Management Studies*, 1997, 34, 1, 1-14.

1. *Journal of Management Studies*, 1997, 34, 1, 1-14.

1. *Journal of the American Medical Association*, 2000; 283: 2689-2696.

©

9

[10] J. J. Moreau, "On the existence of solutions of differential inclusions," *Ann. Inst. Henri Poincaré*, vol. 2, pp. 271–310, 1986.

1000

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99

Appendix D: Cheatsheets

Data Cleaning

9 DATA CLEANING TECHNIQUES

EVERY DATA ANALYST SHOULD KNOW

1 REMOVE DUPLICATES

Identify and remove duplicate rows to keep data unique.

Example (Before)

ID	Name	Email
1	Alice	alice@bio.com
2	Bob	bob@bio.com
2	Bob	bob@bio.com
3	Carol	carol@bio.com

Example (After)

ID	Name	Email
1	Alice	alice@bio.com
2	Bob	bob@bio.com
3	Carol	carol@bio.com

```
df_clean = df.drop_duplicates()
```

Why it matters: Removes redundancy and improves data quality.

2 HANDLE MISSING VALUES

Detect and handle missing or null values appropriately.

Example (Before)

ID	Age	Salary
1	25	50000
2	NA	45000
3	30	NA
4	28	55000

Example (After)

ID	Age	Salary
1	25	50000
2	25	45000
3	30	57500
4	28	55000

```
df['Age'] = df['Age'].fillna(df['Age'].median())  
df['Salary'] = df['Salary'].fillna(df['Salary'].mean())
```

Why it matters: Prevents biased analysis and improves model performance.

3 STANDARDIZE FORMATS

Standardize data formats for consistency (dates, text, units, etc.).

Example (Before)

Date Column
01/02/2024
2024-02-01
Feb 1, 2024
02-01-24

Example (After)

Date Column
2024-02-01
2024-02-01
2024-02-01
2024-02-01

```
df['Date'] = pd.to_datetime(df['Date'])  
df['Date'] = df['Date'].dt.strftime('%Y-%m-%d')
```

Why it matters: Ensures consistency and avoids errors in analysis.

4 CORRECT INCONSISTENCIES

Fix typos, spelling errors, inconsistent text and category values.

Example (Before)

Product
iPhone
IPhone
iPhone
iPhone

Example (After)

Product
iPhone
iPhone
iPhone
iPhone

```
df['Product'] = df['Product'].str.strip()  
df['Product'] = df['Product'].str.lower().str.title()
```

Why it matters: Improves accuracy in grouping and reporting.

5 REMOVE OUTLIERS

Detect and remove outliers that can skew results and affect analysis.

Example (Before)

Salary
45000
47000
48000
1200000
49000

Example (After)

Salary
45000
47000
48000
49000

```
Q1 = df['Salary'].quantile(0.25)  
Q3 = df['Salary'].quantile(0.75)  
df_clean = df[(df['Salary'] >= Q1 - 1.5*(Q3-Q1)) &  
(df['Salary'] <= Q3 + 1.5*(Q3-Q1))]
```

Why it matters: Outliers can distort insights and lead to wrong decisions.

6 FIX DATA TYPES

Convert columns to the correct data types for accurate analysis.

Example (Before)

Column	Type	Value
Age	object	25
Salary	object	45000
Date	object	2024-02-01

Example (After)

Column	Type	Value
Age	int	25
Salary	float	45000.0
Date	datetime64[ns]	2024-02-01

```
df['Age'] = df['Age'].astype('int64')  
df['Salary'] = df['Salary'].astype('float64')  
df['Date'] = pd.to_datetime(df['Date'])
```

Why it matters: Correct types prevent errors and improve performance.

7 REMOVE UNNECESSARY COLUMNS

Drop irrelevant or redundant columns to reduce noise.

Example (Before)

ID	Name	Email	Notes	Temp
1	Alice	alice@bio.com	-	25
2	Bob	bob@bio.com	-	28
3	Carol	carol@bio.com	good	24

Example (After)

ID	Name	Email
1	Alice	alice@bio.com
2	Bob	bob@bio.com
3	Carol	carol@bio.com

```
df_clean = df.drop(columns=['Notes', 'Temp'])
```

Why it matters: Simplifies data and speeds up analysis.

8 VALIDATE & FILTER DATA

Apply validation rules and filters to keep only valid data.

Example (Before)

Age
25
30
-5
150
30
28

Example (After)

Age
25
30
28

```
df_valid = df[(df['Age'] > 0) & (df['Age'] <= 100)]  
df_valid = df_valid[df_valid['Salary'] > 0]
```

Why it matters: Ensures data reliability and business rule compliance.

9 NORMALIZE / SCALE DATA

Normalize or scale numerical data for consistency and better modeling.

Example (Before)

Salary
45000
60000
75000
100000

Example (After)

Salary_scaled
0.30
0.38
0.50
0.65

```
from sklearn.preprocessing import MinMaxScaler  
scaler = MinMaxScaler()  
df['Salary_scaled'] = scaler.fit_transform(df['Salary'])
```

Why it matters: Helps algorithms perform better and faster.

Clean data is the foundation of powerful insights.
Clean it right, analyze it right!

LINK IN BIO!

Decoding the Bag of Words (BoW) Model

Extract every distinct word across all sentences to create a master reference list.

TABLE 1. *Mean and standard deviation of the dependent variables for the three groups of subjects*

Count how many times each unique word appears within an individual sentence.

1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 2680, 26

Sentence	Vector Representation
"Cats love fish"	1 1 1 0 0 0
"Dogs love meat and cats love fish"	1 2 1 1 1 1

Convert the counts into a structured vector based on the master vocabulary size.

Blazing Fast Baseline Efficiency

Extremely simple to compute, making it an ideal starting point for spam filtering.

Complete Loss of Context

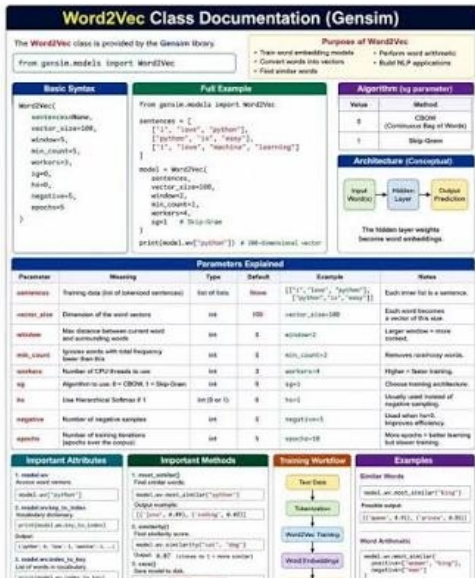
"not good" $\begin{bmatrix} 1 & 1 \end{bmatrix}$

Opposite meanings like "not good" and "good, not" produce identical numerical vectors.

The High Sparsity Problem

Large vocabularies create massive vectors full of zeros, wasting valuable computational memory.

Word2Vec



LLM-Series

techwith.ram

Embedding Space

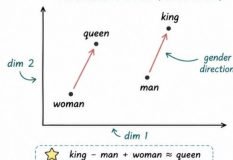
The **high-dimensional** vector space where embeddings live.
Meaningful **relationships**, such as gender or plural,
become consistent **directional offsets** that work across many word pairs.

WHAT IS IT?

- ✓ Every word/token is a point (vector) in a high-dimensional space.
- ✓ Dimensions are learned by the model.
- ✓ Semantic meaning is captured by position and direction.

💡 We can't visualize 768+ dimensions, so we project them down to 2D for intuition.

VISUALIZING IN 2D (FOR INTUITION)



EXAMPLES OF RELATIONSHIPS

- Paris = France + Italy $\approx$ Rome
 - walk + ing $\approx$ walking
 - cat + plural $\approx$ cats
 - teacher = man + woman $\approx$ teacher (female)
- } directions
encode
meaning

These aren't perfect, but they hold surprisingly well across many word pairs.

HOW IT WORKS

- 1 Words with similar meanings are close in space.
- 2 Relationships become consistent directions.
- 3 Arithmetic on vectors captures those relationships.
- 4 This generalizes to unseen word pairs.

 The model learns this space during training by predicting context.

KEY PROPERTIES

	Continuous	Small changes in position mean small semantic changes.
	Structured	Captures complex relationships (synonyms, analogies, roles).
	Universal (to some extent)	Patterns transfer across many languages and tasks.
	High-Dimensional	Typically 512, 768, 1024, ... dims (we just can't see them!)

WHY IT MATTERS

- ✓ Enables semantic search and retrieval.
- ✓ Improves generalization and reasoning.
- ✓ Foundation for everything the model does.

Analogy Example
king = man + woman = queen
If this works,
the model understands
the underlying concept
of "gender".