# Artificial Intelligence II/DL for NLP - 2025
# Homework 1

- Eclass
- Site
- Piazza
- Kaggle

B.Sc. Informatics & Telecommunications
M.Sc. Data Science & Information Technologies

Department of Informatics & Telecommunications
National & Kapodistrian University of Athens

Yorgos Pantis - pantisyorgos@gmail.com

Aiteam
NATIONAL & KAPODISTRIAN
UNIVERSITY OF ATHENS

# Outline

- Data
- Feature Extraction: TF-IDF Method
- Classifier: Logistic Regression
- Evaluation Metrics
- Learning Curves
- Python
- Kaggle Competition
- Report
- Grading
- Questions and Answers

## (1/3) Data *Formats*

- Common text data sources: Documents, Social Media, Web Pages, Emails
- Formats: CSV, JSON, XML, TXT
- Structured vs Unstructured text data

# (2/3) Data *Preprocessing*

- Tokenization: Splitting text into words or phrases
- Lowercasing: Normalizing words to lowercase
- Stopword Removal: Eliminating common words (e.g., "the", "is")
- Stemming & Lemmatization: Reducing words to their base forms
- Removing special characters and punctuation
- Checking if anonymization is needed to ensure privacy and compliance with data protection regulations

# (3/3) Data *Example*

**Original**

4all u guyz out there!!! Did u knw that AI is changin' da world??

BTW, my email is johndoe123@gmail.com

**Preprocessed**

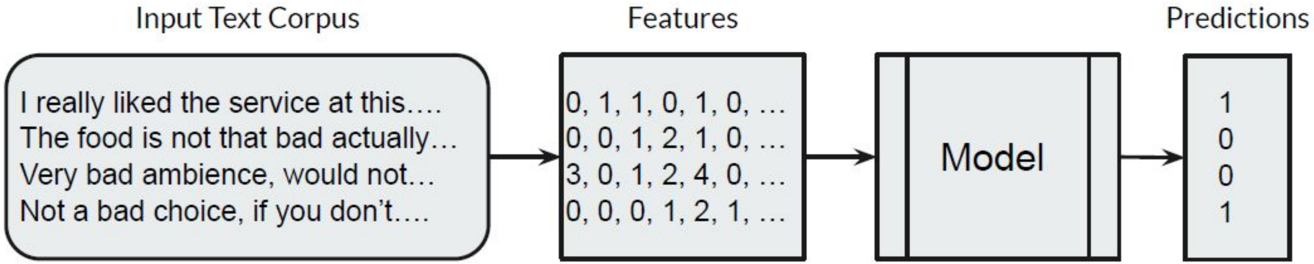
for all you guys out there!!! did you know that ai is changing the world??

btw, my email is XXX@email.com

## (1/9) Feature Extraction - TF-IDF Method *What is it*

- Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical measure used to evaluate word importance
- Key idea: Words that appear frequently in a document but rarely in other documents are important
- TF-IDF reduces the weight of common words (e.g. “the”, “is”) while emphasizing unique terms in a document

Generally, **transforms text into numbers**, where later can be fitted into a model:



## (2/9) Feature Extraction - TF-IDF Method *How it works*

**Term Frequency (TF)**: Measures how often a word appears in a document

$$TF\text{-}IDF(t, d) = TF(t, d) \times IDF(t)$$

TF(t, d) = Number of times a term appears in a document

**Inverse Document Frequency (IDF)**: Measures how important a word is across multiple documents

$$IDF(t) = \log(N / DF(t))$$

N = total number of documents

DF(t) = number of documents containing term

# (3/9) Feature Extraction - TF-IDF Method *Example*

We have a collection of **5 documents** about technology and AI:

1. AI ai is transforming the world of technology
2. Big tech companies invest heavily in AI research
3. AI and machine learning are the future of automation
4. AI is a major trend in the tech industry today
5. Understanding AI and deep learning is essential for innovation

## (4/9) Feature Extraction - TF-IDF Method *Example*

Let's analyse the importance of words **AI** and **automation**:

N = 5 documents

| | **DF** | **IDF = log(N / DF)** |
|---|---|---|
| AI | 5 | log(5 / 5) = log(1) = 0 |
| automation | 1 | log(5 / 1) = log(5) = 0.7 |

**Results:**

> AI is not useful for distinguishing the documents

> automation is more valuable to identify a text

# (5/9) Feature Extraction - TF-IDF Method

## *Why we need log?*

Question: Why we need log?

- Using log ensures that very frequent words don't dominate and rare words aren't overemphasized
- A logarithm function **increases much slower** than a linear function

| | **N / DF** | **log(N / DF)** |
|---|---|---|
| AI | 5 / 5 = 1 | log(1) = 0 |
| automation | 5 / 1 = 5 | log(5) = 0.7 |

**Result:** smoother scaling. Instead of 5 vs 1, we have 0 vs 0.7

## (6/9) Feature Extraction - TF-IDF Method

## *Why we need log?*

**Without log:** A term appearing in **10 documents vs 1 document** would have a **10x difference** in IDF

**With log:** The difference is much smaller, making scoring more stable.

IDF = log(1,000,000 / 1) = log(1,000,000) = 6

IDF = log(1,000,000 / 10) = log(100,000) = 5

## (7/9) Feature Extraction - TF-IDF Method *Drawbacks*

- **Doesn’t capture meaning or context** (e.g., synonyms like "happy" and "joy" are treated differently)
- **Treats words independently**, missing phrases or word order (e.g. "machine learning" is more meaningful than "machine" and "learning" separately
- **Assumes word importance is independent of position or sentence structure**
- **Sensitive to rare words** (terms appearing in very few documents get high scores even if they’re irrelevant)
- **Misspelled Words:** “data science” vs. “dta science” → spelling errors break recognition
- **Context Sensitivity:** “Apple” (fruit) vs. “Apple” (company) → no distinction

## (8/9) Feature Extraction - TF-IDF Method *Improvements*

> Preprocessing

Normalization

- **Problem:** "Data" and "data" are treated as different words
- **Solution:** Convert all text to **lowercase**

Stopwords

- **Problem:** Words like **"is", "the", "and"** appear frequently but carry little meaning
- **Solution:** Remove **stopwords**

c) Spelling Correction

- **Problem:** Misspellings distort TF-IDF results
- **Solution:** Try to fix them **manually** or use spell checkers like **TextBlob** or **SymSpell** to correct mistakes

## (9/9) Feature Extraction - TF-IDF Method *Improvements*

> N-grams (A sequence of N words treated as a single unit in the model)

**Document:** "New York is a big city"

**Unigrams:** ["New", "York", "is", "a", "big", "city"]

**Bigrams:** ["New York", "York is", "is a", "a big", "big city"]

**Trigrams:** ["New York is", "York is a", "is a big", "a big city"]

## (1/3) Classifier - Logistic Regression *Formulation*

**Question:** Given a dataset X = ($X_1$, $X_2$, …, $X_n$) and labels Y = {0, 1}, can we find a function F: X → Y?

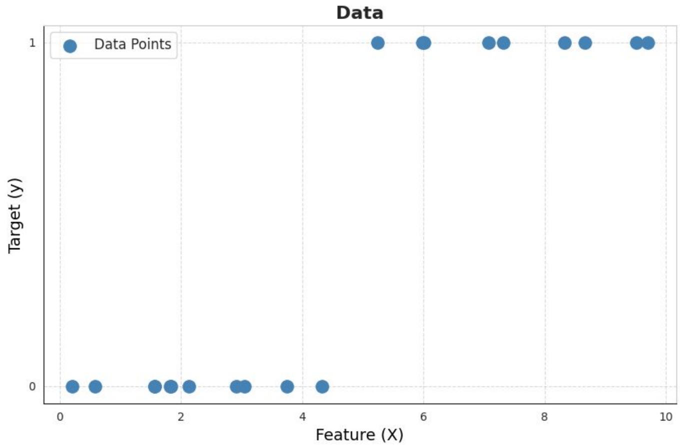
**Thoughts:** A similar problem arises in Linear Regression. Can we adapt it to handle classification tasks?

**Answer:** This adaptation is called **Logistic Regression**

Logistic Regression is used for **binary classification**:

- It models the probability that a given observation belongs to a particular category

$$P(Y=1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\ldots+\beta_n X_n)}}$$



## (2/3) Classifier - Logistic Regression *Connection with LR*

**Linear Regression:** model F: X → Y, where data X = ($X_1$, $X_2$, …, $X_n$) ∈ $R^N$ and Y ∈ R

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$$

- Coefficients $\beta_0$,$\beta_1$, …, $\beta_n$ are estimated by **Mean Squared Error**

**Logistic Regression:** model F: X → Y, where data X = ($X_1$, $X_2$, …, $X_n$) ∈ $R^N$ and Y ∈ {0, 1}

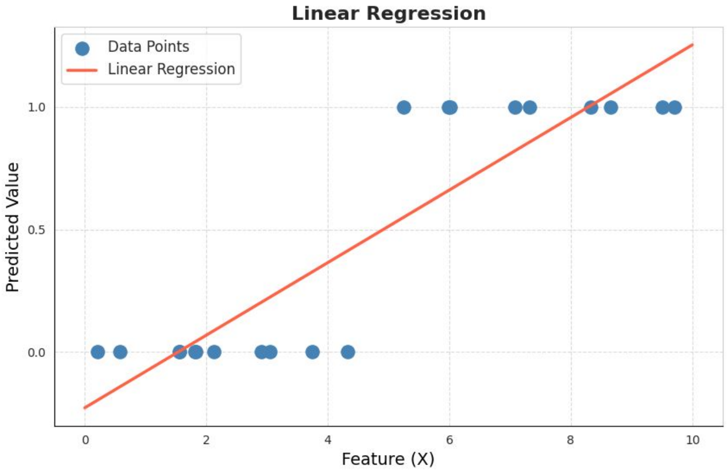
$$\log\left(\frac{P(Y=1|X)}{1-P(Y=1|X)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$$

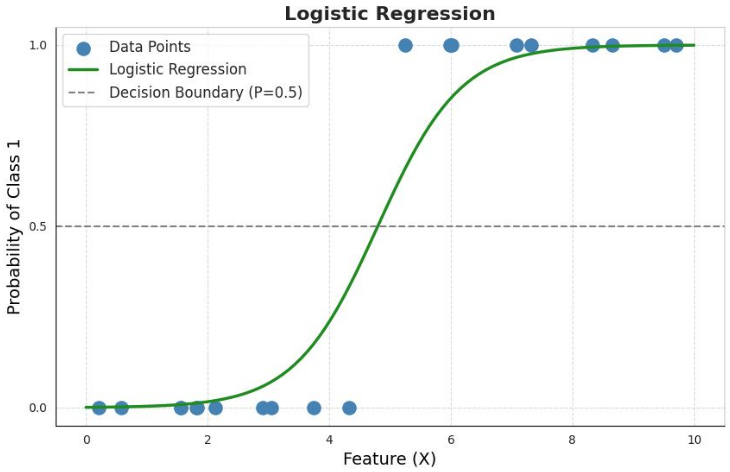
- Coefficients $\beta_0$, $\beta_1$, …, $\beta_n$ are estimated by **Log-likelihood Estimation**

**Note:** Logistic regression is a special case of Generalized Linear Models

# (3/3) Classifier - Logistic Regression *Connection with LR*

| Linear Regression | Logistic Regression |
|---|---|
| • Based on a straight line<br>• Unable to fit binary data properly | • Based on sigmoid function<br>• Able to fit binary data |



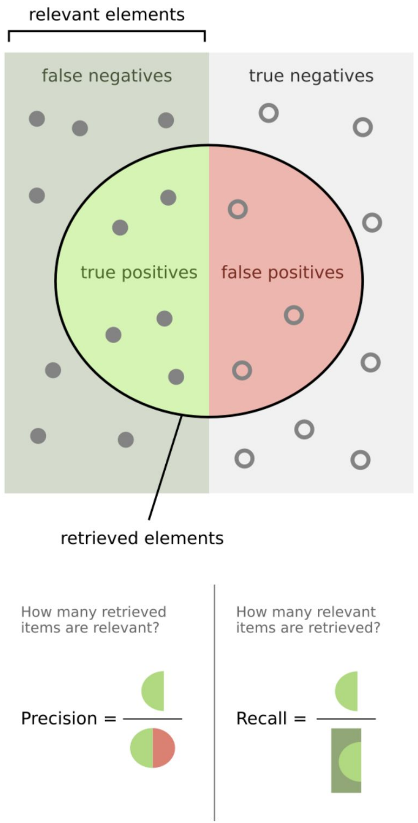
# (1/5) Evaluation Metrics

**Metrics:**

- Accuracy
- Precision
- Recall
- $F_1$ score

Let's denote:

- TP = True Positives (correctly predicted positive instances)
- TN = True Negatives (correctly predicted negative instances)
- FP = False Positives (incorrectly predicted as positive)
- FN = False Negatives (incorrectly predicted as negative)



# (2/5) Evaluation Metrics *Accuracy*

**Accuracy:**

- **Definition:** Measures the proportion of correctly classified instances out of the total instances
- **Formula:**

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

- **Range:** 0 ≤ Accuracy ≤ 1
- **Interpretation:** High accuracy indicates good performance, but it might be misleading for imbalanced datasets

# (3/5) Evaluation Metrics *Precision*

**Precision:**

- **Definition:** Measures the accuracy of positive predictions
- **Formula:**

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

- **Range:** 0 ≤ Precision ≤ 1
- **Interpretation:** High precision means fewer false positives; useful when false positives are costly (e.g., spam detection)

# (4/5) Evaluation Metrics *Recall*

**Recall:**

- **Definition:** Measures the ability to capture all relevant instances
- **Formula:**

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

- **Range:** 0 ≤ Recall ≤ 1
- **Interpretation:** High recall means fewer false negatives; important in scenarios like medical diagnosis

# (5/5) Evaluation Metrics

## *$F_1$ Score*

**$F_1$ Score:**

- **Definition:** Balances precision and recall into a single metric
- **Formula:**

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Range:** $0 \leq F_1$ Score $\leq 1$
- **Interpretation:** The $F_1$ Score is useful when both false positives and false negatives are significant. High useful in unbalanced datasets

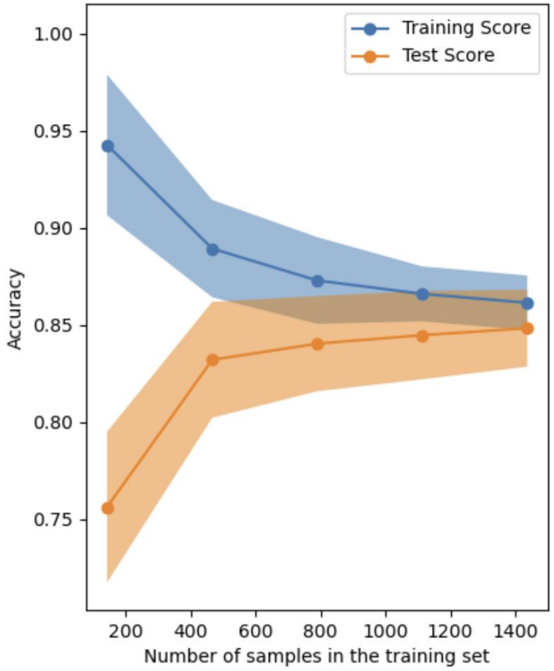
# (1/5) Learning Curves

## *Why we need them*

- Learning curves show model performance over training
- Plot of performance (e.g., general metric, such as **accuracy** or **loss**) against training size or epochs

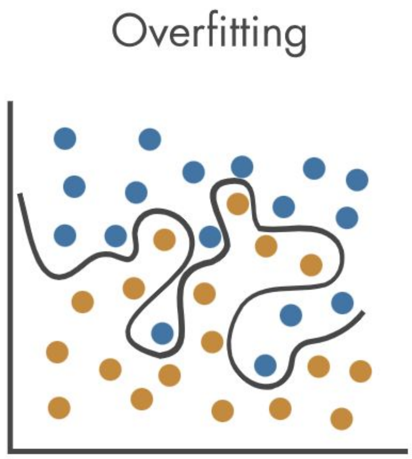
Two curves:

- **Training performance** (on training data)
- **Testing performance** (on unseen data)



## (2/5) Learning Curves *Examples*

**Fitted model for the classification task:**



# (3/5) Learning Curves *Underfitting*

**Underfitting:**

- Model is too simple to capture the underlying pattern
- Low training and validation error

- **Characteristics:** both training and validation curves show low performance
- **Causes:**  when a model is too simple, has insufficient training time, or lacks relevant features
- **Solution:** Increase model complexity, more training time, add relevant features

# (4/5) Learning Curves

## *Overfitting*

**Overfitting:**

- Model learns noise along with the pattern
- Low training error, high validation error

- **Characteristics:** low validation performance with high training performance
- **Causes:**  model complexity, insufficient data, lack of regularization
- **Solution:** regularization, more data, simpler model

**Note:** In some cases, overfitting does not follow this pattern—for instance, overparameterized neural networks can have more parameters than data points yet still achieve near-perfect performance on both training and validation sets

# (5/5) Learning Curves

## *Ideal Plots*

**Ideal Learning Curve:**

- Balanced performance on training and validation sets
- Training curve: low error, stable after some epochs
- Validation curve: approaches training curve, without large gap
- Achieved by appropriate model complexity, sufficient data, and regularization

# (1/7) Python

## *Scikit-learn Libraries*

**Packages:**

- Regular Expression for data cleaning
- TF-IDF Method for feature extraction
- Logistic Regression for fitting the model
- Metrics to evaluate the model's performance

**Online Python:**

- Google Colab

## (2/7) Python

***Code Example***

**Import libraries**

```
# Import basic libraries
import pandas as pd
import numpy as np
import re
import string
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

# (3/7) Python *Code Example*

**Import the data**

```
# Sample dataset with mistakes
data = {
    "text": [
        "I luv this movie, it's amzing!",  # luv -> love, amzing -> amazing
        "Terible film. Waste of time.",  # Terible -> Terrible
        "An excelent performnce by the lead actor!",  # excelent -> excellent, performnce -> performance
        "Not gud, vry boring and slow.",  # gud -> good, vry -> very
        "Fantstic direction and gr8 storytelling.",  # Fantstic -> Fantastic, gr8 -> great
        "Horrble acting and bad script."  # Horrble -> Horrible
    ],
    "label": [1, 0, 1, 0, 1, 0]  # 1 = Positive, 0 = Negative
}

df = pd.DataFrame(data)
```

## (4/7) Python *Code Example*

### Clean the data

```python
# Preprocessing function
def preprocess_text(text):
    text = text.lower()  # Convert to lowercase

    # Correct the spelling mistakes
    text = re.sub(r"\b(luv)\b", "love", text)
    text = re.sub(r"\b(amzing)\b", "amazing", text)
    text = re.sub(r"\b(terible)\b", "terrible", text)
    text = re.sub(r"\b(excelent)\b", "excellent", text)
    text = re.sub(r"\b(performnce)\b", "performance", text)
    text = re.sub(r"\b(gud)\b", "good", text)
    text = re.sub(r"\b(vry)\b", "very", text)
    text = re.sub(r"\b(fantstic)\b", "fantastic", text)
    text = re.sub(r"\b(gr8)\b", "great", text)
    text = re.sub(r"\b(horrble)\b", "horrible", text)
    return text

df["text"] = df["text"].apply(preprocess_text)
```

# (5/7) Python *Code Example*

**Split the data into train/test sets and fit the TF-IDF method**

```python
# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(df["text"], df["label"], test_size=0.2, random_state=42)

# TF-IDF Vectorization
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

# (6/7) Python *Code Example*

**Fit Logistic Regression and print the metrics**

```
# Train Logistic Regression model
model = LogisticRegression()
model.fit(X_train_tfidf, y_train)

# Predictions
y_pred = model.predict(X_test_tfidf)

# Evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report(y_test, y_pred))
```

# (7/7) Python *Code Example*

## Learning curves to check overfitting and underfitting

```
# Generate learning curve
train_sizes, train_scores, test_scores = learning_curve(
    LogisticRegression(), X_train_tfidf, y_train, cv=2, scoring="accuracy", train_sizes=np.linspace(0.1, 1.0, 10)
)

# Compute mean and std of accuracy
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot learning curve
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, label="Training Score", color="blue", marker="o")
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.1, color="blue")

plt.plot(train_sizes, test_mean, label="Validation Score", color="red", marker="s")
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.1, color="red")

plt.xlabel("Training Size")
plt.ylabel("Accuracy")
plt.title("Learning Curve for Logistic Regression")
plt.legend()
plt.show()
```

# (1/3) Kaggle Competition *Rules*

**Rules:**

- Team name: **academic identification number** (sdiXXYYYYY)
- Submission of solutions: file name **submission.csv.** See the **sample_submission.csv**
- Notebook Name: **academic identification number** (sdiXXYYYYY)
- Share your Python Notebook **only with me**

# (2/3) Kaggle Competition *Task*

**Task:** develop a sentiment classifier using **only Logistic Regression** and **only TF-IDF** in Python on a given English-language Twitter dataset

- Exploratory Data Analysis
- Data cleaning
- Fit the TF-IDF method
- Fit the logistic regression model
- Evaluation metric for the model. **Only accuracy** in Kaggle competition and **accuracy, precision, recall, $F_1$ score** in the report
- Plots of the model

# (3/3) Kaggle Competition *Dataset*

**Dataset description:**

- **train_dataset** for training the model
- **val_dataset** for validation the model
- **test_dataset** for testing the model in Kaggle competition

Each dataset consists of three columns:

- **ID**: A unique identifier for each tweet
- **Text**: The content of the tweet
- **Label**: A binary value where **0** represents a **negative** sentiment and **1** represents a **positive** sentiment

**Note:** There are **no labels** in test_dataset

# Report

**Report requirements:**

- Clearly explain your thought process and reasoning
- Provide a detailed description of your methodology
- Justify your choices for model parameters, explaining their impact on performance

**Submission guideline:**

- You **must** follow the provided template for your report using Latex or Word
- Reports must be in English for Data Science & IT master's students; others may choose any language
- Submit your report in **.pdf** format via **only** e-Class
- Name your report as **[full-id].pdf** (e.g., **ZZZZZZXXYYYYY.pdf** if you are a bachelor student in this department)
- You are encouraged to mention in the appendices of your report any other approaches you explored that did not improve the model's performance

# Grading

| | |
|---|---|
| ● **Implementation:** code and Kaggle competition | **Total 70%** |
| ○ EDA and data processing | **10%** |
| ○ Model creation | **20%** |
| ○ Experiments | **30%** |
| ○ Fine Tuning and Optimization | **10%** |
| ● **Report:** analysis and presentation | **Total 30%** |
| ○ Experiments | **10%** |
| ○ Analysis | **15%** |
| ○ Plots | **5%** |

## (1/4) Questions and Answers

**Q1: How to balance preprocessing and accuracy, and the role of EDA in the dataset?**

**A1:**

- Data preprocessing serves multiple purposes—enhancing privacy, reducing overfitting, and improving model performance
- Stop words aren't always a problem. If your model is robust, free from bias toward specific words, and not overfitting, keeping stop words may actually improve accuracy
- Start with EDA: Examine the original dataset and its key statistics to identify potential improvements
- Apply preprocessing and then analyze how these modifications impact the dataset's structure and distribution
- Support your choices with appropriate plots

| | **Format** | **Statistics** |
|---|---|---|
| Original data | - | - |
| Preprocessed data | - | - |

# (2/4) Questions and Answers

**Q2: What is considered as "good" accuracy?**

**A2:**

- Start by considering the worst accuracy you can achieve without using any model at all—this serves as your baseline
- Next, test a simple model without preprocessing or complex architecture to establish an initial benchmark
- Experiment with improvements such as preprocessing, feature engineering, or using a more advanced model
- Perform hyperparameter tuning to optimize performance
- Always watch out for overfitting and underfitting
- Support your choices with appropriate plots

| **Models** | **Scores (train / val / test)** |
|---|---|
| Random | - |
| Baseline | - |
| $Advanced_1$ | - |
| $Advanced_n$ | - |

## (3/4) Questions and Answers

**Q3: What is an "acceptable" level of overfitting for the model?**

**A3:**

- There is no single numerical threshold—context matters. Consider a model with 0% training accuracy and 7% testing accuracy
- Does the model perform very well on the training set but poorly on the validation and test sets? Or does it show similar behavior on validation and test sets?
- Examine the learning curves. How does the model's performance change with increasing data in the training and validation sets?
- Would adding regularization improve generalization?
- Does the model have too many parameters? Reducing them might help strike a balance between complexity and performance
- Support your choices with appropriate plots

# (4/4) Questions and Answers

Any other questions? Feel free to reach out on Piazza

**Good luck!**