

ML practical 2

This is a copy of a conversation between ChatGPT & Anonymous.



I'll share a ML notebook snippets to you please explain them .Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.
This is the problem statement

Perfect 👍 — that's a **Machine Learning practical/problem statement** on **email spam classification using binary classification techniques** — specifically **K-Nearest Neighbors (KNN)** and **Support Vector Machine (SVM)**.

Before you share the notebook snippets, let's quickly break down **what this problem means**, so you'll easily understand each code section when we go through it:

Problem Understanding

We are performing **binary classification**, where:

- **Input (X):** Features extracted from emails (e.g., word frequencies, presence of certain terms, etc.)
- **Output (y):** Label
 - 0 → Not Spam (Normal State)
 - 1 → Spam (Abnormal State)


Steps Involved

Your notebook will likely follow these major steps:

1. Import Libraries

- Packages like `



```
df = pd.read_csv("
/content/emails.csv")
print("

Dataset Loaded Successfully!")
print("Shape:", df.shape)
print(df.head())
```

Great — let's go through this snippet step by step 🙋

Code Explanation

```
python

df = pd.read_csv("/content/emails.csv")
print("✅ Dataset Loaded Successfully!")
print("Shape:", df.shape)
print(df.head())
```

Line-by-Line Explanation

1. `df = pd.read_csv("/content/emails.csv")`

- This line loads the **emails dataset** (a CSV file) into a **Pandas DataFrame** named **df**.
- The file path `/content/emails.csv` indicates it's likely being run in **Google Colab** (since Colab stores files in `/content/`).
- Each **row** in the dataset represents one email.
- The dataset typically contains:
 - Text of the email
 - A label (e.g., **spam** or **ham** / **0** or **1**)

2. `print("✅ Dataset Loaded Successfully!")`

- Just a confirmation message to indicate the dataset loaded without errors.

3. `print("Shape:", df.shape)`

- Displays the **dimensions** of the dataset:
 - `df.shape[0]` → Number of rows (emails)
 - `df.shape[1]` → Number of columns (features + label)

Example output:

```
makefile

Shape: (5172, 2)
```

meaning 5172 emails and 2 columns (perhaps "text" and "spam").

4. `print(df.head())`

- Prints the **first 5 rows** of the dataset for a quick look.
- Helps confirm column names and sample data.

Example output:

vbnet

```
0 Subject: Get cheap meds online now...
1 Hello friend, long time no see!
...
```

text spam
1
0

✓ Summary

Step	Purpose
Load CSV	Bring the dataset into memory
Check Shape	Know how big it is
Inspect Data	Verify column names & sample content

If you share the next code block, I'll explain how the preprocessing, feature extraction (like TF-IDF), and model training (KNN, SVM) steps work next.



```
X = df.iloc[:, 1:-1] # skip Email_Name column
y = df.iloc[:, -1]
imputer = SimpleImputer(strategy='mean') # replaces NaN with mean of column
X = imputer.fit_transform(X)
```

Excellent — now we're moving into the **data preprocessing phase**, where we separate features and labels and handle missing values.

Let's break it down carefully 🙋

🧩 Code Block

python

```
X = df.iloc[:, 1:-1] # skip Email_Name column
y = df.iloc[:, -1]
imputer = SimpleImputer(strategy='mean') # replaces NaN with mean of column
X = imputer.fit_transform(X)
```

🧠 Explanation (Line by Line)

1 `X = df.iloc[:, 1:-1]`

- `.iloc` means **index-based selection** in pandas.
- `[:, 1:-1]` means:
 - `:` → take all rows
 - `1:-1` → take all columns starting from index 1 **up to** (but not including) the last column

- So this **skips the first column** (which might be something like "Email_Name" or "Text") and also skips the **last column** (which is usually the target label).
- Hence, **x = all input features (independent variables)** used for classification.

2 `y = df.iloc[:, -1]`

- -1 means **the last column**.
- This is typically the **label** (target output) — in this case:
 - 0 → Not Spam
 - 1 → Spam
- So, **y = target (dependent variable)**.

3 `imputer = SimpleImputer(strategy='mean')`

- Creates an **imputer object** from scikit-learn.
- Purpose: Handle **missing (NaN)** values.
- `strategy='mean'` means it replaces missing values in each column with that column's **mean value**.
- Other strategies can be 'median', 'most_frequent', or 'constant'.

4 `X = imputer.fit_transform(X)`

- Fits the imputer on the dataset x (calculates the mean of each column)
→ then **transforms** it by replacing any NaN values with those mean values.
- The result is a **cleaned numerical feature matrix** (without missing data).

✓ Summary Table

Line	Purpose
<code>X = df.iloc[:, 1:-1]</code>	Select all feature columns (independent variables)
<code>y = df.iloc[:, -1]</code>	Select the target column (spam or not)
<code>SimpleImputer(strategy='mean')</code>	Create an imputer to fill missing data
<code>fit_transform(X)</code>	Replace NaN values with column means

🧩 In short:

This snippet extracts features and labels from the dataset, then ensures there are no missing values before feeding the data into ML models (KNN or SVM).

Would you like to share the next snippet? It's likely the **train-test split** or **scaling/vectorization** part.



```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

Perfect — now we're at the **model training and prediction stage** using **K-Nearest Neighbors (KNN)** 🔍

Let's go through it line by line 📌

Code Block

python

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

Explanation (Step by Step)

1 `knn = KNeighborsClassifier(n_neighbors=5)`

- Creates a **K-Nearest Neighbors classifier** from `sklearn.neighbors`.
- `n_neighbors=5` → means $K = 5$
→ The model will look at the **5 nearest data points** (neighbors) in the training set to predict the class of a new email.
- KNN is a **lazy learning algorithm**, meaning it doesn't build an explicit model — instead, it memorizes the training data and classifies new points based on distance (usually **Euclidean distance**) to known points.

Concept:

If the majority of the 5 nearest neighbors of an email are spam, it predicts "spam."
Otherwise, it predicts "not spam."

2 `knn.fit(X_train, y_train)`

- This **"fits"** (trains) the model on the training dataset.
- For KNN, "fit" mainly means storing the feature vectors (`X_train`) and labels (`y_train`), because KNN doesn't learn coefficients or weights — it just stores data for comparison later.

3 `y_pred_knn = knn.predict(X_test)`

- Uses the trained KNN classifier to **predict** the output (spam or not spam) for all test data points.
- `y_pred_knn` will contain predicted values like `[0, 1, 0, 0, 1, ...]`.

✓ Summary Table

Step	Code	Purpose
1	<code>knn = KNeighborsClassifier(n_neighbors=5)</code>	Create KNN model with K=5
2	<code>knn.fit(X_train, y_train)</code>	Train the model on training data
3	<code>knn.predict(X_test)</code>	Predict labels for test data

Next Step

After this, you'll usually see **performance evaluation**, such as:

```
python
```

```
print(classification_report(y_test, y_pred_knn))
print(confusion_matrix(y_test, y_pred_knn))
print(accuracy_score(y_test, y_pred_knn))
```

That shows how well KNN performed in detecting spam vs not spam.

Would you like to share the next part (probably the SVM model training) so I can explain how it compares with KNN?



```
print("Accuracy :", round(accuracy_score(y_true, y_pred), 3))
print("Precision:", round(precision_score(y_true, y_pred), 3))
print("Recall  :", round(recall_score(y_true, y_pred), 3))
print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
print("Classification Report:\n", classification_report(y_true, y_pred))
evaluate_model("K-Nearest Neighbors", y_test, y_pred_knn)
evaluate_model("Support Vector Machine", y_test, y_pred_svm)
```

Excellent 👍 — now we're at the **model evaluation** stage, where we measure how well our classifiers (KNN and SVM) performed in detecting spam emails.

Let's break this snippet down step by step 📌

Code Block

```
python
```

```
print("Accuracy :", round(accuracy_score(y_true, y_pred), 3))
print("Precision:", round(precision_score(y_true, y_pred), 3))
print("Recall  :", round(recall_score(y_true, y_pred), 3))
print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
print("Classification Report:\n", classification_report(y_true, y_pred))

evaluate_model("K-Nearest Neighbors", y_test, y_pred_knn)
evaluate_model("Support Vector Machine", y_test, y_pred_svm)
```

Explanation (Step-by-Step)

Metrics Used:

Each of these lines helps analyze the model's **performance** in terms of correctness and reliability.

Accuracy

```
python  
  
accuracy_score(y_true, y_pred)
```

- Measures the **percentage of correctly predicted emails**.
- Formula:

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

- Example: If 90 out of 100 emails are correctly classified → Accuracy = 0.9 or 90%.

Precision

```
python  
  
precision_score(y_true, y_pred)
```

- Measures **how many emails predicted as spam were actually spam**.
- Formula:

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

- High precision means **few false positives** (i.e., not many normal emails were marked as spam).

Recall

```
python  
  
recall_score(y_true, y_pred)
```

- Measures **how many actual spam emails were correctly detected**.
- Formula:

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

- High recall means the model **missed very few spam emails**.

4 Confusion Matrix

```
python

confusion_matrix(y_true, y_pred)
```

- Displays results in a 2x2 table:

	Predicted Spam	Predicted Not Spam
Actual Spam	True Positive (TP)	False Negative (FN)
Actual Not Spam	False Positive (FP)	True Negative (TN)

- Helps visualize **types of errors**.

5 Classification Report

```
python

classification_report(y_true, y_pred)
```



- Provides **Precision, Recall, F1-score, and Support** for each class (Spam and Not Spam).
- Example:

```
scss

precision recall f1-score support
0 (Not Spam)    0.98    0.97    0.97    1500
1 (Spam)        0.95    0.96    0.95     500
```

6 evaluate_model("K-Nearest Neighbors", y_test, y_pred_knn)

7 evaluate_model("Support Vector Machine", y_test, y_pred_svm)

- These are **function calls** (to a custom function named `evaluate_model`).
- That function likely wraps all the above metrics, printing results for each model.
- It compares:
 -  K-Nearest Neighbors (KNN)
 -  Support Vector Machine (SVM)

So you can directly see which model performs better.

✓ Summary Table

Metric	Meaning	Good For
Accuracy	% of correct predictions	Overall correctness

Metric	Meaning	Good For
Precision	% of predicted spam that was truly spam	Avoiding false alarms
Recall	% of actual spam correctly caught	Avoiding missed spam
Confusion Matrix	Detailed outcome counts	Visualizing errors
Classification Report	Full performance summary	Comparing models

Final Step

After running both:

```
python

evaluate_model("KNN", y_test, y_pred_knn)
evaluate_model("SVM", y_test, y_pred_svm)
```

You'll compare which one gives:

- **Higher accuracy**
- **Better precision/recall balance**
- **Fewer misclassifications**

Often, **SVM** performs slightly better for text classification problems like spam detection, since it handles high-dimensional feature spaces (like TF-IDF) efficiently.

Would you like me to explain what happens **inside the `evaluate_model()` function** next? (It'll help you interpret the results better.)



Classify the email using the binary classification method. Email Spam detection has two states:

a)

Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector

Machine for classification. Analyze their performance.

Give viva questions for this with answers

Excellent 🌟 — you've done the practical implementation part!

Now let's prepare **viva questions and answers** based on your **Email Spam Detection using KNN and SVM** project.

Below are **important viva questions (with concise answers)** — ideal for 4–5 mark oral responses or quick explanations 🙋

Viva Questions and Answers

◆ 1. What is the objective of your experiment?

Answer:

The objective is to classify emails into two categories — **Spam** and **Not Spam** — using **binary classification** methods like **K-Nearest Neighbors (KNN)** and **Support Vector Machine (SVM)**, and to compare their performance.

◆ 2. What is binary classification?

Answer:

Binary classification is a type of machine learning problem where the model predicts one of two possible outcomes.

In this case:

- 0 → Not Spam (Normal)
 - 1 → Spam (Abnormal)
-

◆ 3. What dataset did you use?

Answer:

We used the **emails.csv** dataset, which contains email text and a label indicating whether the email is spam or not spam.

◆ 4. What preprocessing steps did you perform?

Answer:

1. Loaded the dataset using Pandas.
 2. Separated features (x) and labels (y).
 3. Handled missing values using **SimpleImputer**.
 4. Converted textual data into numerical form (e.g., using TF-IDF or count vectorization if applied).
 5. Split the data into **training** and **testing** sets.
-

◆ 5. Explain K-Nearest Neighbors (KNN) algorithm.

Answer:

KNN is a **supervised learning algorithm** that classifies a new data point based on the **majority class** of its **K nearest neighbors**.

- It uses distance metrics like **Euclidean distance** to find neighbors.
 - It's a **lazy learner** because it doesn't learn a model during training.
-

◆ 6. What does K=5 mean in KNN?

Answer:

It means the algorithm will look at the **5 nearest neighbors** of a test data point and classify it based on the **majority class** among those neighbors.

♦ 7. Explain the Support Vector Machine (SVM) algorithm.

Answer:

SVM is a **supervised classification algorithm** that finds the **best separating hyperplane** between two classes by **maximizing the margin** between them.

It can use **kernel functions** to handle non-linear separations.

♦ 8. Why did you choose KNN and SVM for this experiment?

Answer:

Because both are widely used and effective for **binary classification** problems.

- KNN is simple and easy to implement.
- SVM performs well with **high-dimensional data** like text (emails) and gives good generalization.

♦ 9. What performance metrics did you use to evaluate the models?

Answer:

We used:

- **Accuracy**
- **Precision**
- **Recall**
- **F1-score**
- **Confusion Matrix**

These metrics were used to compare KNN and SVM results.

♦ 10. What does a confusion matrix show?

Answer:

It shows how many predictions were correct or incorrect:

Actual / Predicted	Spam	Not Spam
Spam	True Positive (TP)	False Negative (FN)
Not Spam	False Positive (FP)	True Negative (TN)

It helps understand the types of classification errors.

♦ 11. What is precision and recall in this context?

Answer:

- **Precision:** Of all emails predicted as spam, how many were truly spam.
- **Recall:** Of all actual spam emails, how many were correctly identified as spam.

High precision reduces false alarms, while high recall ensures fewer spam emails are missed.

◆ 12. Which algorithm performed better and why?

Answer:

Usually, **SVM** performs slightly better than KNN for text classification because:

- It handles high-dimensional data effectively.
- It generalizes better and avoids overfitting.

However, performance depends on the dataset and parameter tuning.

◆ 13. What challenges did you face during the experiment?

Answer:

- Handling **text data** and converting it into numerical features.
 - Choosing the optimal **K value** for KNN.
 - Tuning SVM parameters like **kernel type** and **C value** for better accuracy.
-

◆ 14. What is the importance of data preprocessing?

Answer:

Data preprocessing removes noise, fills missing values, and converts raw text into numerical features — ensuring better model accuracy and reliable results.

◆ 15. What conclusions did you draw from this experiment?

Answer:

Both algorithms successfully classified spam and non-spam emails.

However, **SVM achieved higher accuracy and better precision/recall** compared to KNN, making it more suitable for email spam detection.

◆ 16. What is the role of feature extraction in this project?

Answer:

Since emails are text, we convert them into numeric vectors using techniques like **TF-IDF** or **CountVectorizer**, enabling ML models to process them mathematically.

◆ 17. What type of learning is used here?

Answer:

This is a **Supervised Learning** problem because the model learns from **labeled data** (spam or not spam).

♦ 18. What are true positives and false negatives in spam detection?

Answer:

- **True Positive (TP):** Spam correctly identified as spam.
- **False Negative (FN):** Spam incorrectly identified as not spam.

♦ 19. What library did you use for implementing KNN and SVM?

Answer:

We used **scikit-learn (sklearn)** library, which provides easy-to-use classes like:

- `KNeighborsClassifier()` for KNN
- `SVC()` for SVM

♦ 20. What could you do to further improve accuracy?

Answer:

- Use better feature extraction (TF-IDF with n-grams).
- Apply **hyperparameter tuning** (GridSearchCV).
- Try ensemble models like **Random Forest** or **XGBoost**.
- Increase dataset size for better generalization.

Would you like me to give you **a short summary paragraph** you can speak at the end of your viva (like a conclusion statement)? It makes a great final impression.



Explain knn with example

Sure 👍 — here's a **clear and simple explanation** of the **K-Nearest Neighbors (KNN)** algorithm with an **example** that you can easily use in a viva or report.

K-Nearest Neighbors (KNN) Algorithm

♦ Definition

K-Nearest Neighbors (KNN) is a **supervised machine learning algorithm** used for **classification** and **regression** tasks.

In classification, it predicts the class of a new data point based on the **majority class of its K nearest neighbors** in the training dataset.

It's a **non-parametric** and **lazy learning** algorithm:

- **Non-parametric:** It doesn't make any assumptions about data distribution.
- **Lazy learning:** It doesn't learn a model during training — it stores all training data and makes decisions at prediction time.

Working Steps

1. Choose the number of neighbors (**K**).
2. For a new data point:
 - Calculate the **distance** (usually Euclidean distance) from all points in the training data.
3. Identify the **K closest neighbors**.
4. Take a **majority vote** among these K neighbors.
5. Assign the class with the highest vote to the new data point.

Example

Let's say we want to classify whether a new email is **Spam (1)** or **Not Spam (0)** using KNN.

Email	Keyword Count	Label
A	3	0 (Not Spam)
B	4	0 (Not Spam)
C	5	1 (Spam)
D	7	1 (Spam)

Now we receive a **new email (E)** with **keyword count = 5.5**.

Step : Choose K = 3

We will look at the **3 nearest neighbors**.

Step : Compute distances

Neighbor	Keyword Count	Distance from 5.5	Label
A	3	2.5	0
B	4	1.5	0
C	5	0.5	1
D	7	1.5	1

Step : Pick 3 nearest neighbors

C (0.5), B (1.5), and D (1.5)

Their labels are: **1 (Spam), 0 (Not Spam), 1 (Spam)**

Step 4: Take majority vote

Spam (1) appears **2 times**, Not Spam (0) appears **1 time**.

✓ Therefore, **the new email is classified as Spam.**



Advantages

- Simple and easy to understand.
 - Works well with small datasets.
 - No training phase — all computation happens at prediction time.
-



Disadvantages

- Slow for large datasets (since it compares every point).
 - Sensitive to irrelevant features and scaling.
 - Choice of K and distance metric affects performance.
-



Summary

KNN classifies a new point based on the majority class of its closest K neighbors.

Example: For K=3, if among 3 nearest emails, 2 are spam → the new email is predicted as spam.

Would you like me to also explain **how to choose the best value of K** (for viva or report)? It's a common follow-up question.



Svm in simple words

Sure 😊 — here's a **simple explanation of Support Vector Machine (SVM)** that's easy to understand and perfect for a viva:



Support Vector Machine (SVM) – Simple Explanation

SVM is a **supervised machine learning algorithm** used for **classification** and **regression**, but it is mainly used for **classification problems** like spam detection.

◆ Basic Idea

SVM tries to **find the best boundary (line or plane)** that separates data points of different classes.

- This boundary is called a **hyperplane**.
- The goal of SVM is to find the **hyperplane with the largest margin**, meaning it is as far as possible from the nearest data points of both classes.

Those closest points are called **support vectors**, and they “support” or define the boundary.



Example (Simple 2D case)

Imagine you plot emails on a graph:

- On one side: **Spam emails**
- On the other side: **Not Spam emails**

SVM will draw a **line** (or plane in higher dimensions) that best separates these two groups.

If there are many possible lines, SVM chooses the one that **maximizes the distance** from both classes — so it's the **most confident** in classification.



How it Works

1. SVM finds a **hyperplane** that separates the classes.
2. It tries to **maximize the margin** between the classes.
3. The **support vectors** are the points closest to the hyperplane.
4. For non-linear data, SVM uses a **kernel function** to project the data into a higher dimension where separation is possible.



Simple Analogy

Think of separating apples and oranges on a table with a ruler:

- You place the ruler (decision boundary) so that it divides them clearly.
- You adjust the ruler so it's **as far as possible** from the nearest apple and orange — that's what SVM does.



Advantages

- Works well with **high-dimensional data** (like text or emails).
- Gives **high accuracy**.
- Effective even when the number of features is greater than the number of samples.



Disadvantages

- Slow for very large datasets.
 - Choosing the right **kernel** and parameters can be tricky.
-

In Short:

SVM finds the best possible line (or plane) that separates two classes with the **maximum margin**.

In spam detection, SVM separates **spam** and **not spam** emails based on their features.

Would you like me to also include a **short example** (like with spam and not spam points) to make it easier to visualize for your viva?