

# Editorial – Problema Om vs AI

Soluție realizată de Carla Crivoi (crivoicarla02@gmail.com) și Mihai Nan (mihai.nan@upb.ro)

May 16, 2025

## 1 Descrierea soluției

### 1.1 Rezolvarea subtask-ului 1

Problema este una clasică de clasificare binară: folosind date etichetate ce indică dacă un text este scris de un om sau generat de AI, putem antrena un model de învățare supervizată care poate face această distincție.

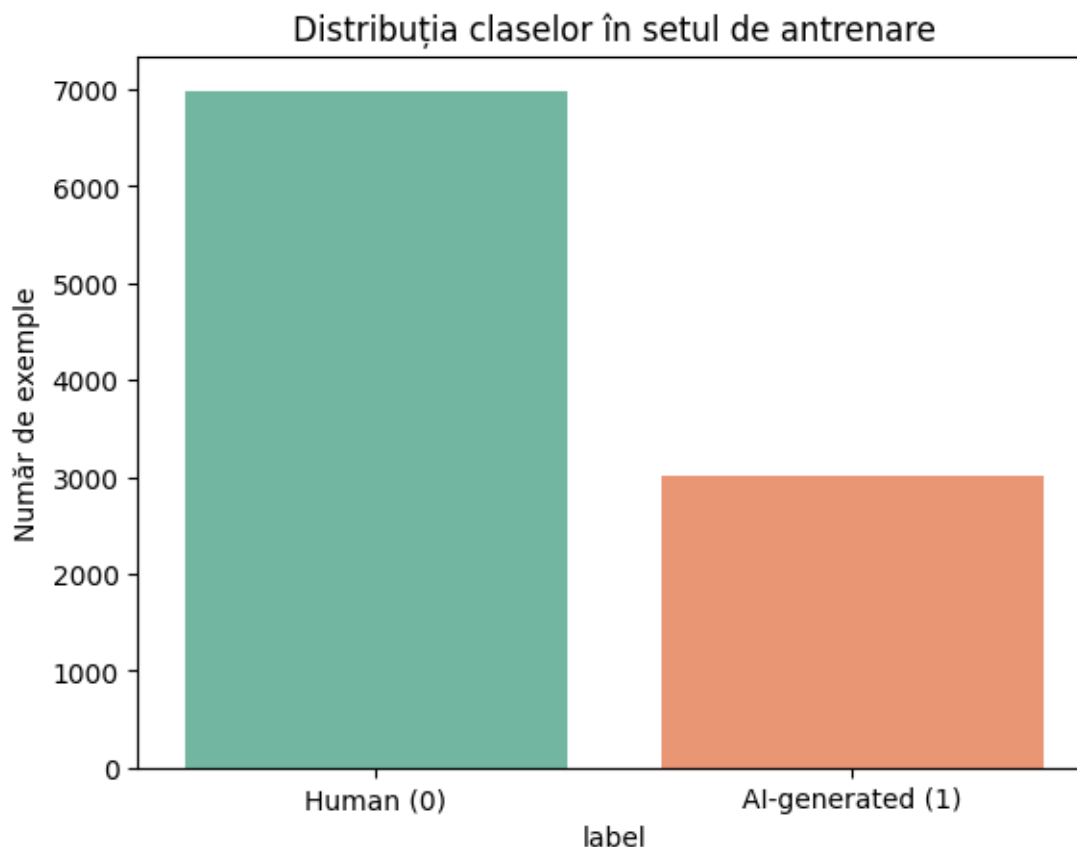
```
[1]: import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
import string
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
import matplotlib.pyplot as plt
import seaborn as sns

train_df = pd.read_csv("train_data.csv")
test_df = pd.read_csv("test_data.csv")
submission = []
```

Înainte de antrenarea modelului, este esențial să verificăm dacă avem un echilibru între clasele 0 (texte scrise de oameni) și 1 (texte generate de AI). Un dezechilibru puternic poate influența negativ performanța modelului.

```
[2]: label_counts = train_df["label"].value_counts().reset_index()
label_counts.columns = ['label', 'count']

sns.barplot(data=label_counts, x='label', y='count', hue='label',
            palette='Set2', legend=False)
plt.xticks([0, 1], ['Human (0)', 'AI-generated (1)'])
plt.ylabel("Număr de exemple")
plt.title("Distribuția claselor în setul de antrenare")
plt.show()
```



Funcția `clean_text` are rolul de a pregăti textul pentru procesarea ulterioară, astfel încât modelele să primească date curate și relevante. Pașii principali realizați de această funcție sunt:

- **Conversia în litere mici** (`text.lower()`): Pentru a evita ca același cuvânt scris cu majusculă sau minusculă să fie tratat diferit, uniformizăm textul, transformându-l pe tot în litere mici.
- **Eliminarea semnelor de punctuație** (`text.translate(...)`): Semnele de punctuație nu adaugă valoare semantică în procesul de clasificare și pot genera zgomot. De aceea, sunt eliminate.
- **Împărțirea textului în cuvinte** (`words = text.split()`): Transformăm șirul de caractere într-o listă de cuvinte.
- **Eliminarea stop words-urilor și a numerelor**: Stop words sunt cuvinte foarte frecvente și fără conținut semnificativ (ex. „and”, „the”), iar cifrele nu vor aduce informație suplimentară relevantă pentru problema noastră.
- **Reconstruirea textului curat**: Cuvintele relevante rămase sunt unite înapoi într-un șir de caractere, pregătit pentru vectorizare.

Astfel, funcția ajută la reducerea zgomotului din date și la evidențierea elementelor cu adevărat relevante pentru clasificare.

```
[3]: def clean_text(text):
    # === transformam textul in litere mici
    text = text.lower()
    # === eliminam semnele de punctuatie
    text = text.translate(str.maketrans('', '', string.punctuation))
    words = text.split()
    # eliminam stop words si numerele
    words = [word for word in words if word not in ENGLISH_STOP_WORDS and not
    ↪word.isdigit()]
    return ' '.join(words)
```

Aplicăm funcția de curățare `clean_text` atât pe setul de antrenare (`train_df`), cât și pe setul de testare (`test_df`). Acest lucru este necesar pentru a asigura consistența preprocesării între cele două seturi — modelul trebuie să vadă date curate și similare ca format atât în timpul antrenării, cât și în timpul testării.

```
[4]: train_df["text"] = train_df["text"].astype(str).apply(clean_text)
    test_df["text"] = test_df["text"].astype(str).apply(clean_text)
```

Deoarece este posibil ca unele texte să fie duplicate, ne asigurăm că ștergem textele care apar de mai multe ori în setul de antrenare.

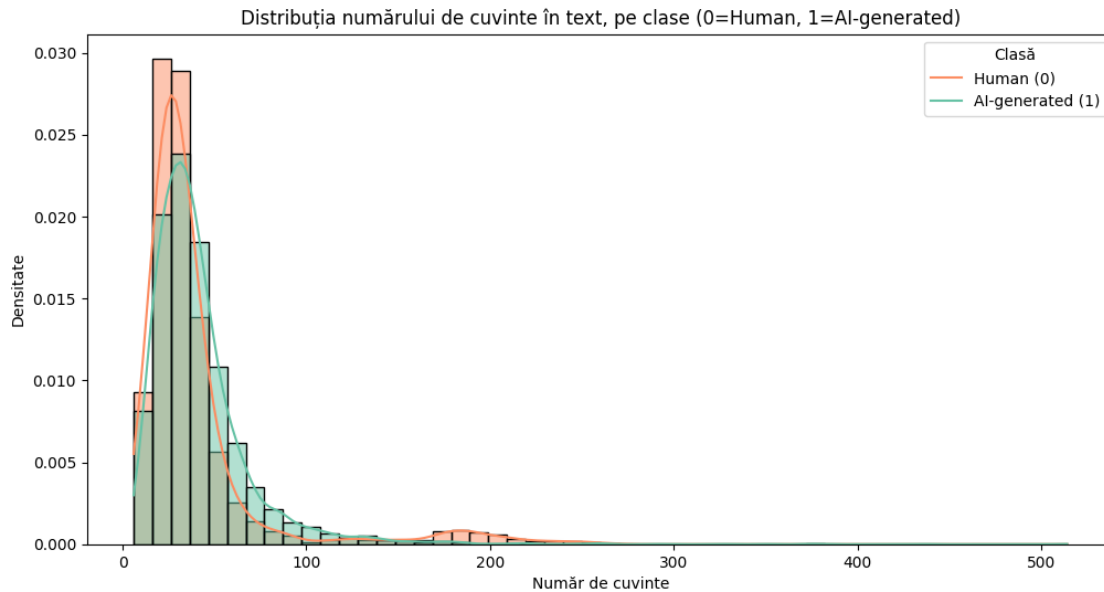
```
[5]: train_df = train_df[train_df['text'].map(train_df['text'].value_counts()) == 1]
```

```
[6]: # === Calculam numarul de cuvinte pentru fiecare text
    train_df['word_count'] = train_df['text'].apply(lambda x: len(str(x).split()))

    # === Vizualizam statisticile sumarizate pe clase
    print(train_df.groupby('label')['word_count'].describe())

    # === Plotam distributia numarului de cuvinte pentru fiecare clasa
    plt.figure(figsize=(12,6))
    sns.histplot(data=train_df, x='word_count', hue='label', bins=50, kde=True,
    ↪palette='Set2', stat='density', common_norm=False)
    plt.xlabel('Număr de cuvinte')
    plt.ylabel('Densitate')
    plt.title('Distribuția numărului de cuvinte în text, pe clase (0=Human,
    ↪1=AI-generated)')
    plt.legend(title='Clasă', labels=['Human (0)', 'AI-generated (1)'])
    plt.show()
```

	count	mean	std	min	25%	50%	75%	max
label								
0.0	6593.0	41.274230	28.142328	6.0	25.0	35.0	49.0	514.0
1.0	2920.0	39.957534	38.499310	7.0	22.0	29.0	40.0	269.0

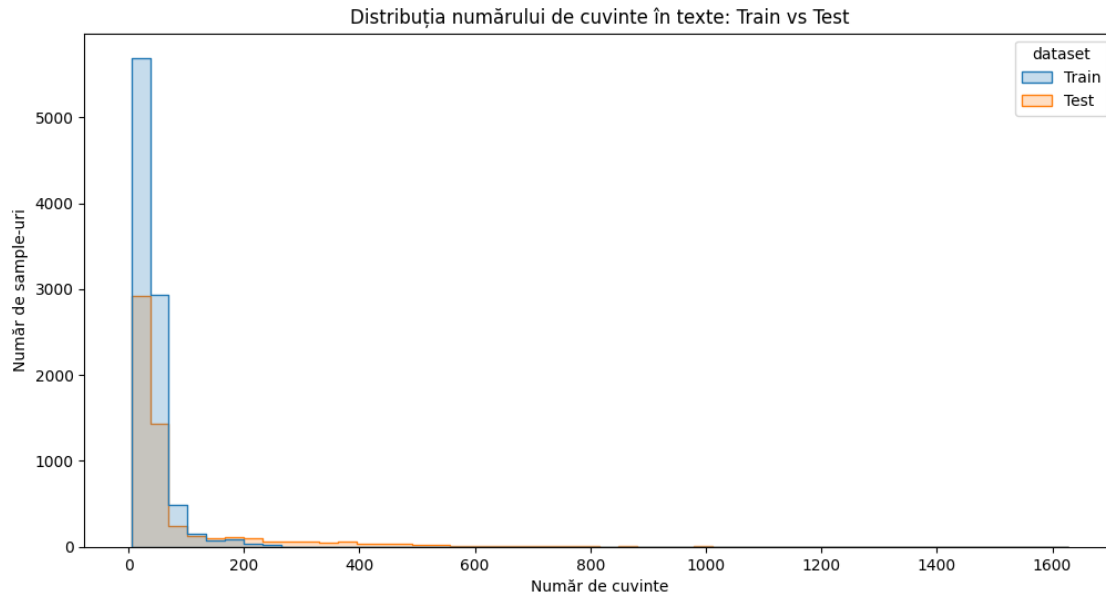


```
[7]: # === Determinam numarul de cuvinte pentru textele din test
test_df['word_count'] = test_df['text'].apply(lambda x: len(str(x).split()))

# === Adaugam o coloana pentru a marca setul (train/test)
train_df['dataset'] = 'Train'
test_df['dataset'] = 'Test'

# === Combinam ambele seturi pentru vizualizare comparativa
combined_df = pd.concat([train_df[['word_count', 'dataset']],
    ↳test_df[['word_count', 'dataset']]], ignore_index=True)

# === Histograma pentru distributia numarului de cuvinte in train vs test
plt.figure(figsize=(12,6))
sns.histplot(data=combined_df, x='word_count', hue='dataset', bins=50,
    ↳element='step', stat='count', palette=['#1f77b4', '#ff7f0e'])
plt.title('Distribuția numărului de cuvinte în texte: Train vs Test')
plt.xlabel('Număr de cuvinte')
plt.ylabel('Număr de sample-uri')
plt.show()
```



Se aplică transformarea **TF-IDF (Term Frequency-Inverse Document Frequency)** pentru a converti textele în vectori numerici.

- `max_features=5000`: păstrează cei mai informativi 5000 de termeni, selectați în funcție de scorul TF-IDF.
- `max_df=0.9`: ignoră termenii care apar în peste 90% din texte (considerați irelevanți sau non-discriminativi).
- `ngram_range=(1, 2)`: se folosesc unigrame (cuvinte individuale) și bigrame (perechi consecutive de cuvinte).
- `sublinear_tf=True`: aplică o scalare logaritmică frecvențelor.
- `norm='l2'`: normalizează fiecare vector de caracteristici la norma L2.

Se antrenează un model **SVM (Support Vector Machine)** cu:

- `kernel="rbf"` – funcția de kernel radială, potrivită pentru separări nelineare.
- `C=1.0` – penalizarea erorilor (regularizare).
- `gamma='scale'` – coeficientul pentru kernel (calculat automat).
- `probability=True` – permite estimarea probabilităților.

```
[8]: train_sub1 = train_df.copy()
test_sub1 = test_df[test_df["subtaskID"] == 1].copy()

# TF-IDF
tfidf1 = TfidfVectorizer( max_features=5000,
                           max_df=0.9,
```

```

        ngram_range=(1, 2),
        sublinear_tf=True,
        norm='l2')
X_train_1 = tfidf1.fit_transform(train_sub1["text"])
y_train_1 = train_sub1["label"]

X_test_1 = tfidf1.transform(test_sub1["text"])

# SVM
svm = SVC(kernel="rbf", C=1.0, gamma='scale', probability=True)
svm.fit(X_train_1, y_train_1)

# Predictie
test_sub1["answer"] = svm.predict(X_test_1)

submission.append(test_sub1[["subtaskID", "ID", "answer"]].rename(columns={"ID": "datapointID"}))
submission_df = pd.concat(submission, ignore_index=True)
submission_df.to_csv("submission_task1.csv", index=False)

```

```

[9]: from sklearn.decomposition import PCA

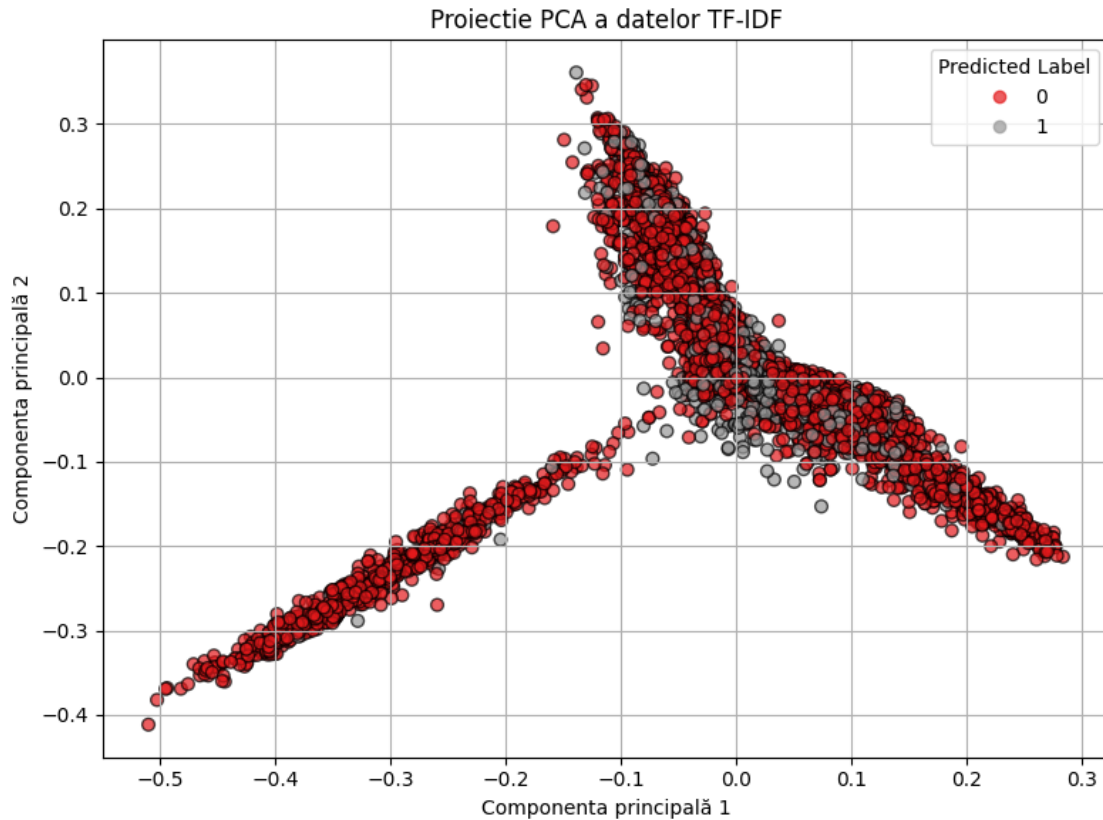
# === Reducem datele din antrenare la 2D cu PCA
pca = PCA(n_components=2, random_state=42)
X_train_2D = pca.fit_transform(X_train_1.toarray())

# === Obtinem predictiile modelului deja antrenat
y_pred_train = svm.predict(X_train_1)

# === Afisam un grafic
plt.figure(figsize=(8, 6))
scatter = plt.scatter(
    X_train_2D[:, 0], X_train_2D[:, 1],
    c=y_pred_train,
    cmap=plt.cm.Set1,
    alpha=0.7,
    edgecolors='k',
    s=40
)

plt.legend(*scatter.legend_elements(), title="Predicted Label")
plt.xlabel("Componenta principală 1")
plt.ylabel("Componenta principală 2")
plt.title("Proiectie PCA a datelor TF-IDF")
plt.grid(True)
plt.tight_layout()
plt.show()

```



## 1.2 Rezolvarea subtask-ului 2

Pentru subtask-ul 2, pare că avem de rezolvat o problemă de clasificare. Tradițional, problemele de clasificare sunt rezolvate folosind algoritmi de învățare supervizată. Însă acest task este unul atipic, deoarece nu avem la dispoziție date etichetate din care să învățăm cum să realizăm clasificarea.

Cu alte cuvinte, este necesar să abordăm problema din perspectiva **învățării nesupervizate**, încercând să descoperim automat tipare și teme recurente în textele furnizate.

```
[10]: import pandas as pd
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import random
import numpy as np
import re
import string

SEED = 42
random.seed(SEED)
np.random.seed(SEED)
```

### 1.2.1 Curățarea datelor

Primul pas constă în curățarea textelor:

- Convertim toate caracterele la litere mici;
- Eliminăm cifrele și semnele de punctuație;
- Standardizăm spațiile albe.

```
[11]: def clean_text(text):
    text = text.lower()
    text = re.sub(r"\d+", "", text)
    text = text.translate(str.maketrans("", "", string.punctuation))
    text = re.sub(r"\s+", " ", text).strip()
    return text

df_test = pd.read_csv("test_data.csv")
df_test = df_test[df_test['subtaskID'] == 2]
df_test["clean_text"] = df_test["text"].astype(str).apply(clean_text)
```

### 1.2.2 Vectorizarea textelor

Pentru a putea aplica modele statistice pe texte, acestea trebuie transformate într-o formă numerică. În acest scop, am folosit **TfidfVectorizer**. Acesta construiește o reprezentare **TF-IDF** (Term Frequency-Inverse Document Frequency), care nu ține cont doar de frecvența cuvintelor, ci și de distribuția acestora în întregul corpus. Astfel, cuvintele comune (precum „said” sau „the”) sunt penalizate, iar cuvintele mai informative (specifice anumitor teme) sunt scoase în evidență.

Am eliminat stop words din limba engleză, iar argumentele `max_df=0.8` și `min_df=5` controlează următoarele: - `max_df=0.8`: ignorăm cuvintele care apar în mai mult de 80% din documente (prea frecvente); - `min_df=5`: ignorăm cuvintele care apar în mai puțin de 5 documente (prea rare).

```
[12]: vectorizer = TfidfVectorizer(max_df=0.8, min_df=5, stop_words='english')
X = vectorizer.fit_transform(df_test['clean_text'])
```

### 1.2.3 Gruparea textelor cu KMeans

Aplicăm algoritmul **KMeans**. Acesta este un algoritm clasic de clustering care împarte datele în K grupuri în funcție de distanțele față de centroizii fiecărui cluster. În cazul nostru, am setat `n_clusters=4` pentru a reflecta cele patru teme cerute în task: **SCIENCE**, **BUSINESS**, **CRIME**, **RELIGION**.

Parametrul `n_init` controlează **de câte ori se va rula algoritmul KMeans** cu setări diferite de inițializare ale centroizilor, păstrând în final soluția care are **cea mai mică valoare a funcției de cost (inertia)**.

**De ce este necesar?** Algoritmul KMeans este sensibil la poziționarea inițială a centroizilor. Dacă aceștia sunt aleși prost, algoritmul poate converge către un minim local nefavorabil (rezultând într-o grupare slabă).



Prin rularea mai multor inițializări ( $n\_init > 1$ ), se pot obține rezultate:

- mai stabile
- mai precise
- mai puțin dependente de norocul alegerilor inițiale

```
[13]: kmeans = KMeans(n_clusters=4, random_state=SEED, n_init=100)
      kmeans.fit(X)
      predicted_labels = kmeans.labels_
```

În urma aplicării algoritmului de clustering, fiecărui text  $i$  se asociază eticheta clusterului în care a fost încadrat. Clusterele nu au un înțeles semantic prestabilit, așa că trebuie interpretate manual.

```
[14]: print(predicted_labels)
```

```
[2 0 2 0 1 1 3 3 3 2 0 2 1 3 3 2 0 0 3 0 2 2 3 3 2 1 2 2 1 0 3 3 0 1 0 3 2
 2 0 3 1 3 0 0 0 2 2 2 2 0 1 0 1 0 3 0 3 1 3 0 3 2 0 3 2 2 3 1 0 0 1 2 3 3
 1 3 0 2 1 3 1 0 1 0 0 3 3 2 0 0 0 2 3 0 1 3 1 1 3 0 0 1 0 1 0 2 2 2 2 3 0
 3 3 2 3 1 2 2 0 2 2 1 0 0 1 3 1 0 1 1 3 3 0 3 0 2 3 1 2 2 3 3 2 1 1 0 3 1
 1 0 0 1 1 2 2 1 2 1 0 2 3 2 3 2 3 2 0 0 0 3 0 2 2 0 0 3 2 1 2 1 1 3 1 0 1
 1 0 3 0 3 2 1 0 1 2 2 0 2 1 3 2 0 3 2 1 3 0 3 3 1 3 0 2 0 3 3 3 3 0 2 3 3
 1 1 0 0 1 3 2 2 2 1 0 3 3 0 3 0 2 3 1 2 3 2 0 2 2 0 3 3 1 2 1 3 3 3 0 2 0
 3 1 1 1 2 2 1 1 2 0 3 1 2 1 0 0 1 1 0 2 0 3 3 1 1 3 3 0 2 1 3 2 1 0 0 3 3
 1 2 0 2 1 1 0 2 3 1 2 1 0 1 3 1 0 0 3 3 0 2 2 2 0 3 3 3 0 1 3 3 3 1 0 2 2
 2 3 2 0 0 2 2 3 1 2 2 3 0 0 3 1 0 3 0 2 0 3 2 3 1 2 3 3 1 2 0 3 3 3 1 3 0
 3 1 1 0 3 0 0 3 3 3 1 3 3 3 2 3 1 1 0 2 1 2 0 1 2 1 3 1 0 1 1 0 2 2 2 2 0
 2 0 2 0 3 3 0 1 1 1 1 2 2 0 2 3 3 3 3 1 1 2 2 2 2 0 2 3 0 0 1 0 0 1 1 1 0
 0 2 2 2 1 0 2 0 1 0 3 0 0 3 2 1 0 2 1 0 0 2 0 3 3 2 0 2 2 2 0 1 2 1 1 1 3
 0 1 2 1 3 2 3 3 2 0 1 3 1 0 0 1 2 3 3 2 0 0 1 2 2 1 2 1 3 0 2 2 0 1 3 1 3
 2 0 2 2 3 3 1 0 1 2 3 1 2 2 0 0 2 0 1 2 2 0 3 2 3 0 0 2 1 0 3 1 1 2 3 1 0
 0 1 3 0 3 3 2 3 1 1 0 2 1 1 0 1 3 0 3 1 2 1 0 3 2 3 3 3 2 2 3 3 2 1 3 2 3
 2 1 0 0 1 2 2 2 1 3 2 3 2 0 2 0 1 2 2 2 2 0 3 3 3 2 2 2 3 3 3 2 2 0 2 1 1
 3 3 3 2 2 1 1 1 2 0 1 0 2 0 1 2 1 1 0 0 0 1 0 3 1 3 1 0 0 1 0 1 0 0 2 1 3
 3 3 1 3 3 2 3 1 1 0 1 1 3 3 0 0 1 2 1 3 0 0 1 1 0 1 3 0 2 3 1 0 3 0 0 2 2
 2 3 2 0 3 0 3 1 2 3 2 2 2 2 0 3 3 1 2 3 2 2 3 1 2 1 3 1 1 3 0 1 1 1 2 1
 1 2 2 1 1 0 0 2 1 1 3 2 1 2 3 1 1 2 1 3 2 0 2 1 0 0 0 3 0 0 1 1 2 3 0 0 3
 1 3 2 0 0 2 0 3 1 1 2 2 0 3 3 1 0 1 1 3 0 2 2]
```

#### 1.2.4 Interpretarea clusterelor

Pentru a înțelege ce semnifică fiecare cluster, analizăm termenii cei mai frecvenți din fiecare grupare. Acest lucru ne ajută să atribuim fiecărui cluster o temă semantică relevantă.

```
[15]: terms = vectorizer.get_feature_names_out()
      centroids = kmeans.cluster_centers_

      for cluster_idx in range(4):
          top_terms_idx = centroids[cluster_idx].argsort()[::-1][:10]
```

```
top_terms = [terms[i] for i in top_terms_idx]
print(f"Cluster {cluster_idx}: {' '.join(top_terms)}")
```

Cluster 0: scientists, science, space, nasa, earth, climate, said, mars, planet, research

Cluster 1: police, said, school, shooting, told, according, county, yearold, cosby, suspect

Cluster 2: company, uber, said, percent, business, women, amazon, new, companies, employees

Cluster 3: god, church, religious, people, jesus, white, christian, faith, trump, said

După inspecția manuală, asociem fiecare cluster unei categorii tematice după cum urmează:

```
[16]: label_map = {
      0: "SCIENCE",
      2: "BUSINESS",
      3: "RELIGION",
      1: "CRIME"
    }
```

### 1.2.5 Crearea fișierului de predicții

Pe baza etichetelor de cluster și a mapării noastre semantice, am generat coloana answer cu predicțiile finale. Rezultatul a fost salvat într-un fișier .csv, conform formatului cerut în enunț.

```
[17]: df_test['cluster'] = predicted_labels
      df_test['answer'] = df_test['cluster'].map(label_map)
      submission = df_test[["answer"]].copy()
      submission["subtaskID"] = 2
      submission["datapointID"] = df_test["ID"]
      submission.to_csv("submission_task2.csv", index=False)
```

### 1.2.6 Interpretarea vizuală a clusterelor folosind WordCloud

Pentru a înțelege semnificația fiecărui cluster generat de algoritmul KMeans, am reprezentat vizual cuvintele predominante din centroizii fiecărui cluster folosind tehnica **word cloud**. Aceasta oferă o reprezentare intuitivă a temelor centrale, pe baza următoarelor idei:

- **Dimensiunea cuvântului** reflectă importanța lui în cadrul acelui cluster (scorul TF-IDF din centroid).
- Prin inspectarea vizuală, putem deduce tema asociată fiecărui cluster chiar și în lipsa unor etichete explicite în setul de antrenare.
- Astfel, acest pas servește ca o **metodă de etichetare semi-supervizată**, bazată pe intuiția umană.

De exemplu:

- Clusterul în care apar frecvent termeni precum *scientists*, *research*, *science* a fost etichetat ca SCIENCE.
- Clusterul dominat de termeni precum *religious*, *church*, *god* a fost etichetat RELIGION.
- Termeni precum *uber*, *company*, *business* sugerează tema BUSINESS.
- Termeni ca *police*, *suspect*, *shooting* indică tema CRIME.

Această abordare oferă un mod eficient de **clasificare tematică** a textelor în absența unui set de date etichetat.

```
[18]: import matplotlib.pyplot as plt
      from wordcloud import WordCloud

      terms = vectorizer.get_feature_names_out()
      centroids = kmeans.cluster_centers_

      fig, axes = plt.subplots(4, 1, figsize=(10, 18))

      for cluster_idx, ax in enumerate(axes):
          top_terms_idx = centroids[cluster_idx].argsort()[::-1][:30]
          top_terms = {terms[i]: centroids[cluster_idx][i] for i in top_terms_idx}

          wordcloud = WordCloud(width=800, height=400, background_color='white')
          wordcloud.generate_from_frequencies(top_terms)

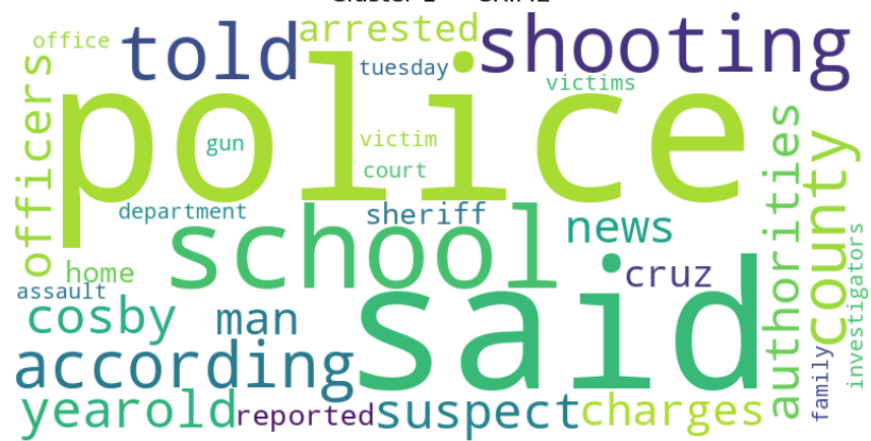
          ax.imshow(wordcloud, interpolation='bilinear')
          ax.axis('off')
          ax.set_title(f"Cluster {cluster_idx} - {label_map[cluster_idx]}",
                      ↪fontsize=16)

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

Cluster 0 — SCIENCE



Cluster 1 — CRIME



Cluster 2 — BUSINESS



Cluster 3 — RELIGION



### 1.2.7 Vizualizarea clusterelor KMeans cu PCA

Pentru a înțelege mai bine cum sunt grupate textele în cluster, putem proiecta vectorii TF-IDF în două dimensiuni folosind **PCA** (Principal Component Analysis). Această reducere de dimensionalitate este esențială pentru vizualizare, deoarece datele TF-IDF originale se află într-un spațiu de dimensiune mare.

Vizualizarea ne poate oferi:

- **Intuiție** despre cât de bine s-au separat tematicile.
- **Posibilitatea de a identifica suprapuneri** sau zone în care clusterelor nu sunt clare.
- **Confirmări vizuale** pentru alegerea numărului de cluster.

```
[19]: from sklearn.decomposition import PCA

pca = PCA(n_components=2, random_state=SEED)
X_reduced = pca.fit_transform(X.toarray())
centroids_2d = pca.transform(kmeans.cluster_centers_)
for i in range(4):
    plt.scatter(X_reduced[predicted_labels == i, 0], X_reduced[predicted_labels_
    ↪== i, 1], label=f"Cluster {i}")
plt.scatter(centroids_2d[:, 0], centroids_2d[:, 1], c='black', s=150, alpha=0.6,
    ↪marker='X', label='Centroids')
plt.title(f"KMeans Clustering")
plt.legend()
plt.tight_layout()
plt.show()
```

