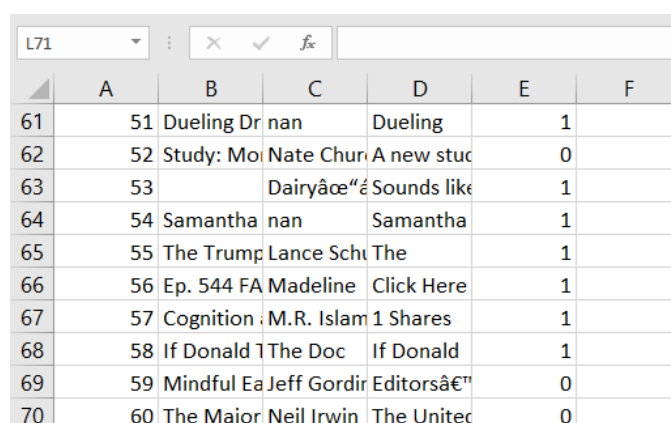


Detectarea știrilor false folosind inteligența artificială

Acest proiect constă în dezvoltarea unei soluții eficiente, bazate pe învățare automată, pentru detectarea știrilor false, folosind un set de date populat cu articole în mai multe limbi.

1. Alegerea și completarea setului de date

Am plecat de la setul de date preexistent compilat de William Lifferth pentru site-ul Kaggle (<https://www.kaggle.com/c/fake-news>), deoarece include aproximativ 25000 de intrări individuale (20000 pentru antrenare și 5000 pentru testare), structurate conform formatului .csv. Pentru adăugarea de articole noi, se poate folosi un editor de text (precum notepad++), un site precum convertcsv.com sau orice aplicație similară cu Microsoft Excel. De asemenea, se pot folosi API-uri (acolo unde există) sau se poate face web scraping pentru a obține rapid cantități mari de date, inclusiv folosind extensii pentru Google Chrome.



	A	B	C	D	E	F
61	51	Dueling Dr	nan	Dueling	1	
62	52	Study: Mo	Nate Chur	A new stud	0	
63	53		Dairyâce	" Sounds like	1	
64	54	Samantha	nan	Samantha	1	
65	55	The Trump	Lance Schi	The	1	
66	56	Ep. 544 FA	Madeline	Click Here	1	
67	57	Cognition	M.R. Islam	1 Shares	1	
68	58	If Donald T	The Doc	If Donald	1	
69	59	Mindful Ea	Jeff Gordir	Editorsâ€™	0	
70	60	The Major	Neil Irwin	The Unitec	0	

Fișierul „train.csv” deschis în Excel

Fiecare intrare conține următoarele atribute:

- ID
- Titlu
- Autor
- Text
- Etichetă (1 pentru știrile false și 0 pentru cele în care putem avea încredere).

Setul de date „train.csv” nu conține eticheta, aceasta putând fi găsită asociind ID-ul știrii cu ID-ul corespondent din fișierul „submit.csv”.

2. Alegerea modulelor și abordarea problemei

Preluarea și prelucrarea datelor se va face cu ajutorul modulului **Panda**, în timp ce **sklearn** ajută la antrenare și testare.

Metoda aleasă este regresia logistică, implementarea acesteia constând într-o funcție sklearn. Mai importantă este procesarea textului, pentru care am ales vectorizarea TF-IDF (Term Frequency - Inverse Document Frequency). Prin vectorizare TF-IDF, fiecărui termen din șir îi este asociată o valoare întreagă, în funcție de frecvența aparițiilor sale. Partea de „IDF” din numele metodei se referă la faptul că un cuvânt este considerat mai puțin important dacă apare în multe articole diferite.

3. Preprocesarea datelor

Etapa de preprocesare a datelor este destul de simplă. Singura modificare făcută în această etapă este concatenarea titlului articolului cu textul și cu numele autorului, pentru simplificarea problemei. Pentru optimizarea ulterioară a modelului am putea separa categoriile (în special autorul) și oferi fiecăreia o pondere în procesul de clasificare.

```
train_df['combined_text'] = train_df['title'] + ' ' + train_df['author'] + ' ' + train_df['text']
test_df['combined_text'] = test_df['title'] + ' ' + test_df['author'] + ' ' + test_df['text']
```

Concatenarea categoriilor

2. Antrenarea modelului / Afișarea și interpretarea rezultatului antrenării

Primul pas a fost împărțirea setului de date de antrenament în date de antrenament și date de test. Am făcut acest lucru pentru a putea valida performanța modelului pe setul de date inițial, lucru care va fi util mai târziu. Ulterior, modelul este reantrenat folosind întreg setul de date de antrenament și pot fi folosite datele de testare finale. Așadar, în această etapă, codul este următorul:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
submit_df = pd.read_csv("submit.csv")
predictions_df = pd.read_csv("predictions.csv")

train_df['combined_text'] = train_df['title'] + ' ' +
train_df['text']
test_df['combined_text'] = test_df['title'] + ' ' +
test_df['text']

# Handle NaN
```

```
train_df['combined_text'].fillna('', inplace=True)
test_df['combined_text'].fillna('', inplace=True)

# Splitting data for validation
train_data, valid_data = train_test_split(train_df,
test_size=0.2, stratify=train_df['label'], random_state=42)

# TF-IDF Vectorization
vectorizer = TfidfVectorizer(max_features=5000,
stop_words=["english", "french", "spanish", "german", "italian"])

X_train = vectorizer.fit_transform(train_data['combined_text'])
X_valid = vectorizer.transform(valid_data['combined_text'])
X_test = vectorizer.transform(test_df['combined_text'])

y_train = train_data['label']
y_valid = valid_data['label']

model = LogisticRegression(random_state=42)

# Training the model
model.fit(X_train, y_train)

# Predictions on the validation set
valid_preds = model.predict(X_valid)

valid_accuracy = accuracy_score(y_valid, valid_preds)
print(f"Validation Accuracy: {valid_accuracy*100:.4f}%")

# Retrain on the entire train.csv
X_train_full = vectorizer.transform(train_df['combined_text'])
y_train_full = train_df['label']

model.fit(X_train_full, y_train_full)

# Generate predictions on the test set
test_preds = model.predict(X_test)

# Assign labels to the predictions.csv file
predictions_df['label'] = test_preds

# Check percentage of correct results
correct_results_percentage = accuracy_score(submit_df['label'],
predictions_df['label']) * 100
print(f"Percentage of Correct Results on Submit.csv:
{correct_results_percentage:.2f}%")
```

Validation Accuracy: 95.6010%

Un rezultat aparent bun, procentul ridicat al acurateții testării pe setul de antrenare poate arăta spre fenomenul de overfitting, în cazul în care procentul de rezultate corecte pentru setul de testare este scăzut.

4. Testarea modelului / Afișarea și interpretarea rezultatului testării

Percentage of Correct Results on Submit.csv: 63.71%

Se pare că, într-adevăr, există o disparitate mare între rezultatele obținute seturile de testare diferite. Putem fi aproape siguri că este vorba despre overfitting, întrucât setul de date este unul foarte mare, iar alte modele de regresie folosite pe același set (vezi <https://www.kaggle.com/c/fake-news>) au rezultate mult mai bune.

Din fericire, clasa `LogisticRegression` are un parametru de regularizare C , care poate fi setat la orice valoare. În cazurile în care avem de a face cu overfitting, C ar trebui să scadă. Setând $C = 0.1$ obținem rezultate similare, dar, mergând la $C = 0.01$:

```
model = LogisticRegression(random_state=27, C=0.01)
```

obținem următorul rezultat:

```
main x
↑ Validation Accuracy: 81.0337%
↓ Percentage of Correct Results on Submit.csv: 73.81%
|:|
|:| Process finished with exit code 0
```

Cele două rezultate converg în jurul valorii de 77%, pentru $C = 0.005$:

```
main x
↑ Validation Accuracy: 77.8365%
↓ Percentage of Correct Results on Submit.csv: 76.87%
|:|
|:| Process finished with exit code 0
```

Pentru mai multe informații utile, putem apela la funcția `scikit classification_rep`:

```
classification_rep = classification_report(submit_df['label'],
predictions_df['label'])
print("\nClassification Report:")
print(classification_rep)
```

```
Classification Report:
      precision    recall  f1-score   support

     0       0.72      0.80      0.76      2339
     1       0.82      0.74      0.78      2861

 accuracy          0.77      5200
 macro avg       0.77      0.77      0.77      5200
weighted avg       0.77      0.77      0.77      5200
```

Observăm că modelul are tendința să clasifice articole adevărate ca fiind false, dar diferența de acuratețe nu este una foarte mare.