Detectarea știrilor false folosind inteligența artificială – continuare

Etapa aceasta a proiectului implică utilizarea a doi algoritmi diferiți (Naïve Bayes și K-Nearest Neighbors pentru a rezolva problema de clasificare a știrilor false abordată în tema anterioară. Ne dorim să evaluăm performanțele fiecărui algoritm și să le comparăm cu rezultatele obținute prin antreare folosind regresia logistică. În cele din urmă, voi încerca să maximizez acuratețea predicțiilor prin utilizarea ambilor algoritmi.

1. Definirea cerințelor și preprocesarea datelor

Setul de date folosit a fost același ca în tema 1. Ca atare, etapa de preprocesare a datelor rămâne neschimbată. Vectorizarea TF-IDF și concatenarea câmpurilor cu text din fișierele .csv rămân și ele, astfel încât comparația să fie cât mai corectă:

```
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")

# Concatenate title and text
train_df['combined_text'] = train_df['title'] + ' ' + train_df['text']
test_df['combined_text'] = test_df['title'] + ' ' + test_df['text']

# Handle NaN values
train_df['combined_text'].fillna('', inplace=True)
test_df['combined_text'].fillna('', inplace=True)

# TF-IDF Vectorization
vectorizer = TfidfVectorizer(max_features=5000, stop_words=["english",
"french", "spanish", "german", "italian"])
X_train = vectorizer.fit_transform(train_df['combined_text'])
X_test = vectorizer.transform(test_df['combined_text'])
y_train = train_df['label']
```

2. Algoritmul Naïve Bayes

2.1. Aplicabilitatea:

Algoritmul Naïve Bayes este folosit cu preponderență în procesarea textului, deci va fi un caz interesant de studiat. Numele algorimtului face referire la presupunerea (de obicei naivă) că toate trăsăturile analizate sunt complet independente. Trăsăturile textului sunt cuvinte individuale, fiecare cu o probabilitate diferită de a apărea într-un articol fals față de unul veridic. Cât timp corelația dintre frecvența de apariție a unor cuvinte anume și tipul articolului este ridicată, algoritmul va genera, cel puțin la nivel teoretic, predicții corecte.

2.2. Procesul de antrenare:

Pentru început, acestea sunt bibliotecile Python care vor fi utilizate:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix
```

Instanțiera și antrenarea modelului:

```
model = MultinomialNB()
model.fit(X_train, y_train)
```

2.3. Verificarea rezultatelor:

```
accuracy = accuracy_score(submit_df['label'], predictions_df['label'])
print(f"Test Accuracy: {accuracy*100:.4f}%")
```

```
Test Accuracy: 60.6923%
```

Pentru a determina motivul pentru rezultatul destul de slab, am împărțit setul de antrenare și am folosit cross-validation.

```
# Split the training data
train_data, valid_data = train_test_split(train_df, test_size=0.2,
random_state=42)
y_train = train_data['label']
y_valid = valid_data['label']

# 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'])

# Instantiate the Naive Bayes model
model = MultinomialNB()

# Cross-validation
cross_val_scores = cross_val_score(model, X_train, y_train, cv=5)
print(f"Cross-validation scores: {cross_val_scores}")
print(f"Average score: {cross_val_scores.mean():.4f}")

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

# Evaluate 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}%")

# Generate predictions on the test set
test_preds = model.predict(X_test)
predictions_df['label'] = test_preds
```

```
accuracy = accuracy_score(submit_df['label'], predictions_df['label'])
print(f"Test Accuracy: {accuracy*100:.4f}%")

conf_matrix = confusion_matrix(submit_df['label'], predictions_df['label'])
conf_matrix_df = pd.DataFrame(conf_matrix, index=['True Negative', 'True
Positive'], columns=['Predicted Negative', 'Predicted Positive'])
print(conf_matrix_df)
print("\nConfusion Matrix:")
```

```
Cross-validation scores: [0.89513221 0.91646635 0.90955529 0.90594952 0.9140625 ]
Average score: 0.9082
Validation Accuracy: 91.1058%
Test Accuracy: 61.2115%

Confusion Matrix:
              Predicted Negative  Predicted Positive
True Negative                1508                 831
True Positive                1186                1675

Process finished with exit code 0
```

Observăm că avem de a face cu un caz de overfitting pe setul original de date. Modificarea numărului maxim de trăsături gruparea cuvintelor nu au vreo influență semnificativă asupra rezultatelor. Parametrul „alpha” al funcției MultinomialNB() nu afectează nici el rezultatul, întrucât setul de date este mare și nu are nevoie să fie netezit.

Concluzia pare să fie că algorimtul Naïve Bayes (deși este potrivit pentru filtrarea spam-ului, de exemplu) nu este optim în această situație.

3. Algorimtul K-Nearest Neighbors

3.1. Aplicabilitatea:

K Nearest Neighbors este un algoritm de clasificare a datelor în funcție de clasa cea mai comuă printre vecinii săi cei mai apropiați. Așadar, gradul de similitudine între două articole îl reprezintă distanța (euclidiană, Manhattan etc. – după caz) dintre ele, în spațiul vectorial în care fiecare dimensiune corespunde unei trăsături (cuvânt sau grup de cuvinte). Ne putem aștepta, din acest motiv, să fie nevoie să reducem numărul maxim de trăsături.

3.2. Procesul de antrenare:

Va fi nevoie de clasa KNeighborsClassifier din modulul sklearn.neighbors:

```
from sklearn.neighbors import KNeighborsClassifier
```

Singura modificare care va fi adusă inițial, față de script-ul anterior, va fi:

```
model = KNeighborsClassifier(n_neighbors=5)
```

Deoarece antrenarea modelului dura prea mult timp, am păstrat doar titlurile articolelor.

3.3. Verificarea rezultatelor:

Folosind parametrii originali, algoritmul decide că aproape fiecare articol este fals:

```
Validation Accuracy: 51.2260%
Test Accuracy: 55.9423%

Confusion Matrix:
               Predicted Negative  Predicted Positive
True Negative                  86                2253
True Positive                  38                2823
```

Acest lucru se întâmplă pentru orice valoare a lui K, așa că soluția se află în altă parte. Următorul parametru ca importanță îl reprezintă numărul maxim de trăsături, pe care îl vom reduce substanțial. Motivul are de a face cu cantitatea de date de antrenare de care am avea nevoie pentru a ne asigura că există suficiente exemple pentru cât mai multe combinații de trăsături. Aceasta crește exponențial cu numărul de trăsături folosite (https://en.wikipedia.org/wiki/Curse_of_dimensionality#Machine_learning).

3.4. Modificarea parametrilor și rezultatele finale:

Am redus numărul maxim de trăsături la 30 și crescut K la 3000 (valori determinate prin încercări repetate – o regulă comună este folosirea rădăcinii pătrate a numărului de intrări din setul de antrenare, în cazul acesta 141). Rezultatele obținute sunt mult mai favorabile, deși încă au un *bias* către știri false:

```
Cross-validation scores: [0.81219952 0.80949519 0.8031851  0.80198317 0.80979567]
Average score: 0.8073
Validation Accuracy: 77.2115%
Test Accuracy: 74.8462%

Confusion Matrix:
              Predicted Negative  Predicted Positive
True Negative                1255                1084
True Positive                 224                2637
```

4. Crearea unui model care îmbină cei doi algoritmi

4.1. Ideea de bază

Principiul pe care se bazează implementarea finală este următorul: sunt create ambele modele, apoi un algoritm este folosit pentru a valida rezultatele pentru care celălalt nu are un grad de siguranță suficient de ridicat. Predicțiile de bază vor fi cele KNN, deoarece a avut rezultate mai bune.

4.2. Preprocesarea datelor:

Întrucât Naïve Bayes a fost antrenat pe întreg textul articolului, iar KNN doar pe titlu, vom folosi două seturi de trăsături diferite:

```
# for Naive Bayes
train_df['combined_text_nb'] = train_df['title'].str.lower() + ' ' +
train_df['text'].str.lower()
test_df['combined_text_nb'] = test_df['title'].str.lower() + ' ' +
test_df['text'].str.lower()

# for KNN
train_df['title_knn'] = train_df['title'].str.lower()
test_df['title_knn'] = test_df['title'].str.lower()
```

4.3. Combinarea rezultatelor:

Algoritmul de combinare a predicțiilor celor două modele constă în parcurgerea fiecărui rezultat în parte, pentru ca cele unde KNN a fost nesigur să fie înlocuite cu predicțiile Naïve Bayes.

```
knn_preds = model_knn.predict(X_test_knn)
nb_preds = model_nb.predict(X_test_nb)

knn_probs = model_knn.predict_proba(X_test_knn)  # KNN with probability
knn_prob_threshold = 0.8

# Getting probabilities and neighbors for KNN
final_predictions = []
for i in range(len(knn_preds)):
    max_prob = np.max(knn_probs[i])  # The proportion of neighbors that
agree with the prediction
    if max_prob < knn_prob_threshold:
        final_predictions.append(nb_preds[i])  # Naive Bayes prediction
    else:
        final_predictions.append(knn_preds[i])  # KNN prediction

predictions_df['label'] = final_predictions
```

4.4. Rezultatele finale:

Din păcate, diferența de performanță dintre cele două modele este atât de mare încât până și predicțiile nesigure KNN sunt mai precise decât predicțiile Naïve Bayes. Ca urmare, rezultatele vor fi cu atât mai slabe cu cât Naïve Bayes trebuie să intervină mai des:

```
knn_prob_threshold = 0.8
```

```
Naive Bayes Validation Accuracy: 89.6635%
KNN Validation Accuracy: 77.2115%
Test Accuracy: 61.1731%

Confusion Matrix:
              Predicted Negative  Predicted Positive
True Negative                1553                 786
True Positive                1233                1628
```

```
knn_prob_threshold = 0.6
```

```
Naive Bayes Validation Accuracy: 89.6635%
KNN Validation Accuracy: 77.2115%
Test Accuracy: 74.0577%

Confusion Matrix:
              Predicted Negative  Predicted Positive
True Negative                1344                 995
True Positive                 354                2507
```

5. Concluzii:

În mod neașteptat, modelul K Nearest Neighbors, antrenat doar pe titlurile articolelor, a dus la rezultate mult mai bune. Provocarea în antrenarea sa pe întreg textul articolelor constă în faptul că parametrii K și max_features trebuie ajustați, lucru dificil când fiecare proces de clasificare durează aproximativ 5-10 minute (pentru varianta în care am folosit doar titlul, a fost nevoie de aproximativ 20 de modificări pentru a ajunge la valorile prezentate). Un alt motiv pentru care am putut obține rezultate bune a fost faptul că algoritmul KNN are mai mulți parametri personalizabili care aduc schimări reale. Acesta poate fi îmbunătățit și mai departe, prin eliminarea *bias*-ului.