Nitro Language Processing - 3rd Edition - Satire

Clasificarea binară a articolelor de știri românești în categoriile satiră / non-satiră

George Florea 342, Dragos Teleaga 342, Sergiu Stanciu 342, Aldea Alexia 344

Care a fost scopul proiectului?

Tema a constat in implementarea unui model eficient de clasificare a stirilor romanesti intre cele satirice si non-satirice. Avand la indemana o gama variata de date, care cuprind atat stiri reale cat si compuse, a trebuit sa venim cu un algoritm capabil de a face distinctia intre cele doua categorii de texte prezentate.

Inainte de a incepe orice tentativa de a scrie modelul de clasificare binara propriu zis, a trebuit sa luam in considerare mai multe aspecte importante precum:

- Preprocesarea textului luand in calcul aparitia diacriticelor in datele de intrare
- Selectia caracteristicilor relevante
- Reprezentarea textului in formatul numeric adecvat, Selectia caracteristicilor relevante
- Alegerea unui model de clasificare binar care sa functioneze bine pe un set de date bazat pe texte
- O posibila optimizare a hiperparametrilor si Evaluarea performantei conform cerintei specificate de competitie.

Dupa ce ne-am intemeiat un plan care incepea sa prinda din ce in ce mai mult contur, am inceput sa ne implementam modelul folosind tehnicile enumerate mai sus in ordinea:

1. Preprocesarea textelor

```python
import spacy
from spacy.lang.ro.stop_words import STOP_WORDS

nlp = spacy.load("ro_core_news_sm", disable=["parser", "ner"])

def preprocess_texts(texts):
    texts = [str(text) if pd.notnull(text) else '' for text in texts]

    docs = nlp.pipe(texts, batch_size=50)
    preprocessed_texts = []
    for doc in docs:
        preprocessed_text = ' '.join(token.lemma_ for token in doc if not token.is_stop and not token.is_punct and token.lemma_ != "-PRON-")
        preprocessed_texts.append(preprocessed_text)
    return preprocessed_texts


train['title_processed'] = preprocess_texts(train['title'].tolist())
train['content_processed'] = preprocess_texts(train['content'].tolist())
```

Codul acesta primește o listă de texte ca intrare și aplică preprocesarea textului folosind biblioteca SpaCy pentru procesarea limbajului natural (NLP). Preprocesarea include eliminarea cuvintelor de oprire, a semnelor de punctuație și reducerea cuvintelor la forma lor de bază. Aceste operații sunt esențiale pentru pregătirea datelor textuale înainte de a le folosi pentru antrenarea sau evaluarea unui model NLP, ajutând la reducerea zgomotului și concentrându-se pe informațiile semnificative din text.

2. Reprezentarea textului in format numeric adecvat

Această bucată de cod antrenează un model Word2Vec pe baza datelor preprocesate și tokenizate, pentru a învăța reprezentări vectoriale ale cuvintelor. Apoi, folosind acest model, generează vectori de documente pentru seturile de date de antrenare și de testare. Acești vectori de documente reprezintă textele sub formă numerică și sunt esențiali pentru antrenarea și evaluarea modelelor de clasificare NLP, deoarece capturează semantica textului.

```
train['content_processed']=train['content_processed']+train['title_processed']
test['content_processed']=test['content_processed']+test['title_processed']
```

```
from gensim.models import Word2Vec
from sklearn.model_selection import train_test_split
# Prepare Documents for Word2Vec
documents = [text.split() for text in pd.concat([train['content_processed'], test['content_processed']])]

# Train Word2Vec on the entire dataset
word2vec_model = Word2Vec(sentences=documents, vector_size=100, window=5, min_count=1, workers=4)
```

```
# Document Vector Function
def document_vector(word2vec_model, doc):
    doc = [word for word in doc if word in word2vec_model.wv.key_to_index]
    if not doc:
        return np.zeros(word2vec_model.vector_size)
    return np.mean(word2vec_model.wv[doc], axis=0)

# Apply the Function to Create Document Vectors
train_vectors = np.array([document_vector(word2vec_model, doc.split()) for doc in train['content_processed']])
test_vectors = np.array([document_vector(word2vec_model, doc.split()) for doc in test['content_processed']])
```

3. Alegerea unui model adecvat pentru setul de date de intrare

3.1. Clasificatorul Naive Bayes Gaussian

Pentru a ne asigura ca ceea ce am facut pana in acest punct isi atinge scopul dorit, am preferat sa avem Naive Bayes drept primul nostru model intrucat este un model simplu, eficient din punct de vedere computational, are performante acceptabile, oferindu-ne un balanced accuracy de aproximativ 0.78.

```python
# Train-Test Split (For Evaluating Model Performance)
from sklearn.metrics import accuracy_score

labels = train['class'].astype(int).values
X_train, X_val, y_train, y_val = train_test_split(train_vectors, labels, test_size=0.2, random_state=42)

# Initialize and Train the Classifier
clf = GaussianNB()
clf.fit(X_train, y_train)

# Predict on the Validation Set and Evaluate
y_val_pred = clf.predict(X_val)
print(f"Validation Accuracy: {accuracy_score(y_val, y_val_pred)}")
```

3.2. Clasificatorul SVM

Facand primii pasi si vazand ca incetul cu incetul ne atingem obiectivele, am inceput sa schimbam putin clasificatorul folosit si sa alegem unul de tip SVM in schimb. Dupa putin research, am ajuns la concluzia ca un astfel de clasificator are o capacitate mult mai mare de a gestiona date complexe dovedind performante superioare pentru astfel de seturi comparativ cu un clasificator NaiveBayes.

```python
from sklearn.svm import SVC
import pandas as pd

# Initialize the SVC classifier
clf = SVC(kernel='linear', random_state=42)

# Train the classifier
clf.fit(train_vectors, labels)  # Assuming train_vectors and labels are defined

# Predict on the Test Set
test_predictions = clf.predict(test_vectors)

# Ensure test_predictions are in the format of integers 0 and 1
test_predictions_int = [int(pred) for pred in test_predictions]

# Create a DataFrame with IDs and predictions
submission_df = pd.DataFrame({
    'id': test['id'],  # Replace 'id' with the actual name of your ID column if it's different
    'class': test_predictions_int
})

# Save the DataFrame to a CSV file
submission_df.to_csv('submission_svm.csv', index=False)

print("Submission CSV file for SVM predictions has been created.")
```

3.3 Clasificatorul CNN

Acest cod folosește o rețea neuronală convoluțională (CNN) pentru clasificarea binară a datelor. Se pregătesc datele, se definește și se compilează modelul CNN în Keras, apoi se antrenează și se evaluează. Performanța modelului este evaluată folosind scorul de acuratețe echilibrat.

```
X_train = np.expand_dims(train_vectors, axis=2)
X_test = np.expand_dims(test_vectors, axis=2)
from keras.models import Sequential
from keras.layers import Conv1D, GlobalMaxPooling1D, Dense, Dropout

model = Sequential()
model.add(Conv1D(filters=256, kernel_size=7, activation='relu', input_shape=(100, 1)))
model.add(GlobalMaxPooling1D())
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
X = train_vectors  # Feature vectors
y = train['labels'].values
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
history = model.fit(X_train, y_train, epochs=5, validation_split=0.1, batch_size=32)

from sklearn.metrics import balanced_accuracy_score

# Assuming you have predictions for your validation set
y_pred = model.predict(X_val)
y_pred_binary = (y_pred > 0.5).astype(int)  # Convert probabilities to binary predictions

balanced_acc = balanced_accuracy_score(y_val, y_pred_binary)
print("Balanced Accuracy:", balanced_acc)
```

Disclaimer: CNN-urile nu merg bine pe un set de date de tip text => un astfel de model nu era potrivit pentru problema noastra

```
history = model.fit(X,y, epochs=5, validation_split=0.1, batch_size=32)
```

```
y_pred = model.predict(test_vectors)
y_pred_binary = (y_pred > 0.5).astype(int)  # Convert probabilities to binary predictions
```

3.4. Clasificatorul Random Forest

Dupa o incercare esuata de a forta marirea indicelui "balanced accuracy" folosind un CNN (a dat un indice chiar mai mic fata de cat ne-am asteptat), am decis sa ne orientat spre clasificatori care sunt cunoscuti pentru aplicatiile lor pe seturi de date de tip text. Astfel am dat de clasificatorul Random Forest care ne-a dat cel mai mare scor si anume: 0.87516

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_vectors, train['labels'], test_size=0.2, random_state=42)
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
predictions = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)

# Additional evaluation metrics
print(classification_report(y_test, predictions))
```

```
rf_classifier.fit(X,y)
predictions = rf_classifier.predict(test_vectors)
```

```
y_pred_binary = (y_pred > 0.5).astype(int)  # Convert probabilities to binary predictions
import csv
submission_df = pd.DataFrame({'class': predictions.flatten()})
print(submission_df)
submission_df.to_csv('predictionFolly.csv', index=True)
print("Submission CSV file for SVM predictions has been created.")
```