Documentație

Proiect Inteligență Artificială – Machine Learning / Noiembrie 2022

Overview

Scopul acestui proiect este de a clasifica texte traduse în 5 limbi străine din engleza în funcție de dialectul lor nativ: dialect irlandez, englezesc sau scoțian. Această problemă de învățare supravegheată este una clasică de procesare a limbajului natural. Vom implementa diferite abordări pentru a observa care este mai bună pentru setul nostru de date.

Sorete Robert-Alexandru
Grupa 361

Pasul 1.1. Înțelegerea Setului de Date

Pentru modelul nostru avem următoarele seturi de date de intrare disponibile:

1. **train_data.csv** - *date de antrenare*, format dintr-un .csv ce conține 41570 de exemple de antrenare împărțite în 3 coloane: limba textului, textul tradus în limba respectiva, și dialectul textului nativ (**eticheta**)
2. **test_data.csv** - *date de testare* (conține textele ce trebuie clasificate)

Ca set de date de ieșire avem:

1. **submission.csv** - *predicția clasificatorului* nostru format din index și etichetă

Pasul 1.2. Citirea Setului de Date

În implementarea acestei probleme vom citi tabelele cu ajutorul librăriei **pandas** pentru a le stoca sub forma unui **dataframe** și ne vom extrage informațiile utile în variabilele X, y reprezentând textele de antrenare, respectiv etichetele de antrenare, iar în X_TEST textele de test.

```
import pandas as pd

# Citirea datelor
train_data_df= pd.read_csv('train_data.csv')
test_data_df = pd.read_csv('test_data.csv')

# Se elimina coloanele care nu sunt relevante
X = train_data_df['text']
X_TEST = test_data_df['text']
y = train_data_df['label']

# Atribuirea fiecarei clase un id
label2id = {'Ireland': 0, 'England': 1,  'Scotland': 2}
id2label = {0: 'Ireland', 1: 'England',  2: 'Scotland'}

y = [label2id[label] for label in y]
```

Pasul 1.3. Preprocesarea Datelor

Pentru a scăpa de caracterele inutile din text și a păstra cuvintele esențiale ce vor fi analizate de modelul nostru, vom folosi conceptul de preprocesare a textului. Putem efectua diferite metode de preprocesare a datelor de text astfel încât să păstrăm informațiile necesare pentru o estimare cât mai eficientă:

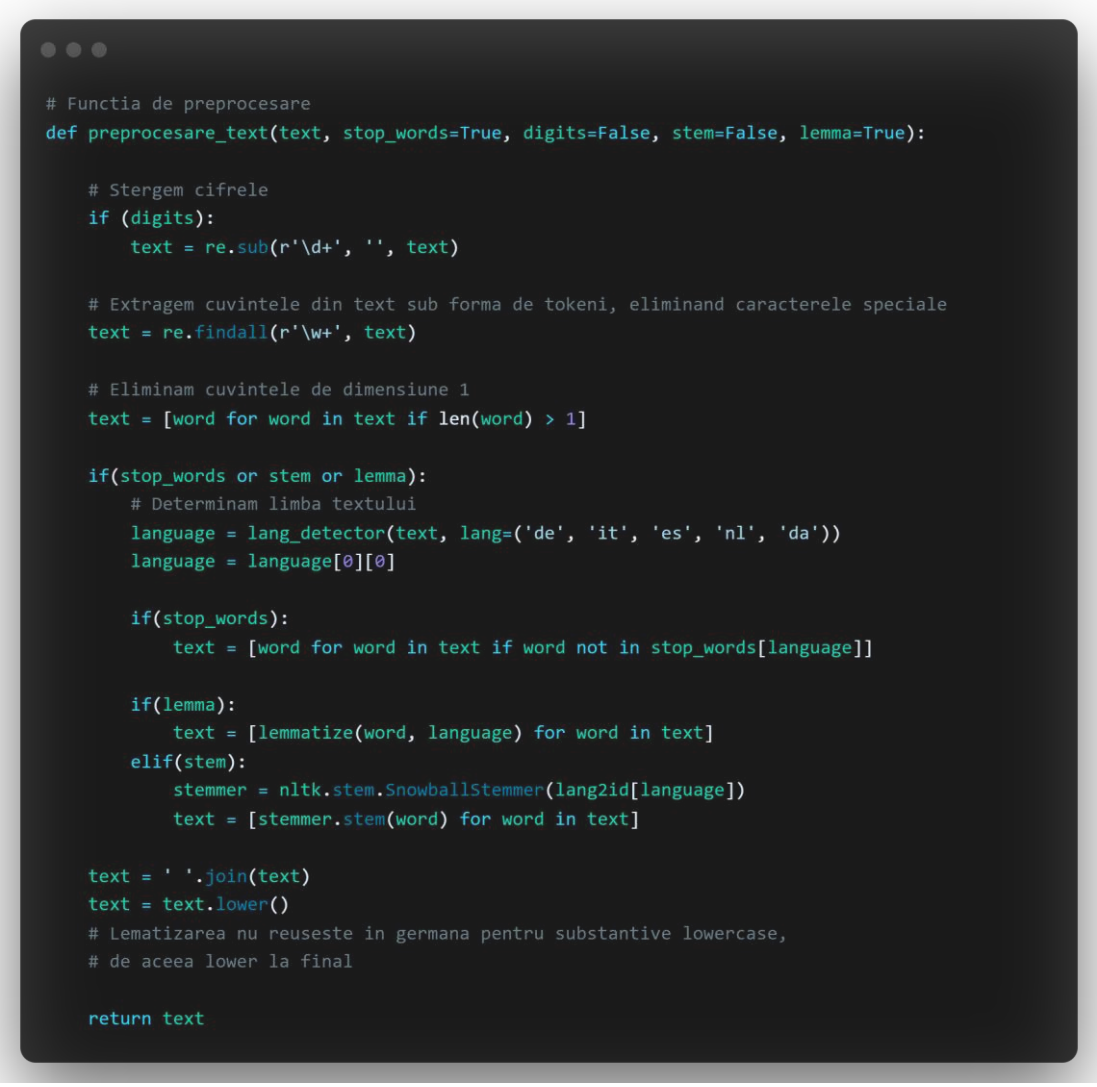
- ștergerea caracterelor speciale
- eliminarea **stopwords**
- **lematizarea/stematizarea** cuvintelor
- eliminarea cifrelor

Fiind un set de date destul de mare, nu ne putem da seama ce preprocesare este mai eficientă. Astfel, am ales să compun o funcție ce îmi va preprocesa datele în mai multe metode pentru a le putea compara și extrage pe cele mai optime.

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from simplemma import lang_detector, lemmatize  # pip install simplemma

nltk.download('stopwords')  # Descarcam stopwords

german_stopwords = stopwords.words('german')
italian_stopwords = stopwords.words('italian')
spanish_stopwords = stopwords.words('spanish')
dutch_stopwords = stopwords.words('dutch')
danish_stopwords = stopwords.words('danish')

# Constuirea dictionarului de stopwords
stop_words = {'de': german_stopwords, 'it': italian_stopwords, 'es': spanish_stopwords,
              'nl': dutch_stopwords, 'da': danish_stopwords}

# Acest dictionar este folosit pentru stemmatizarea textului,
# deoarece folosim o librarie diferita de lemmatizare
lang2id = { 'de': 'german', 'it': 'italian', 'es': 'spanish',
           'nl': 'dutch', 'da': 'danish', 'unk': 'german'}
```

Lematizare vs Stematizare – ambele au scopul de a reduce formele flexionare și derivate ale unui cuvânt la o forma de baza comuna

Stematizarea se referă, de obicei, la un proces euristic brut care taie capetele cuvintelor în speranța atingerii corecte acestui obiectiv. Lematizarea se referă la a face lucrurile în mod corespunzător cu utilizarea unui vocabular și a unei analize morfologice a cuvintelor, urmărind în mod normal să elimine numai terminațiile flexionare și să returneze forma de baza sau de dicționar a unui cuvânt, care este cunoscută sub numele de lema.



```python
# Functia de preprocesare
def preprocesare_text(text, stop_words=True, digits=False, stem=False, lemma=True):

    # Stergem cifrele
    if (digits):
        text = re.sub(r'\d+', '', text)

    # Extragem cuvintele din text sub forma de tokeni, eliminand caracterele speciale
    text = re.findall(r'\w+', text)

    # Eliminam cuvintele de dimensiune 1
    text = [word for word in text if len(word) > 1]

    if(stop_words or stem or lemma):
        # Determinam limba textului
        language = lang_detector(text, lang=('de', 'it', 'es', 'nl', 'da'))
        language = language[0][0]

        if(stop_words):
            text = [word for word in text if word not in stop_words[language]]

        if(lemma):
            text = [lemmatize(word, language) for word in text]
        elif(stem):
            stemmer = nltk.stem.SnowballStemmer(lang2id[language])
            text = [stemmer.stem(word) for word in text]

    text = ' '.join(text)
    text = text.lower()
    # Lematizarea nu reuseste in germana pentru substantive lowercase,
    # de aceea lower la final

    return text
```

Funcția este făcută astfel încât, să nu poți preprocesa utilizând lematizarea și stematizarea. Daca lemma și stem sunt ambele True, se va efectua doar lematizarea.

```
X_stop = [preprocesare_text(text, stop_words=True) for text in X]
X_TEST_stop = [preprocesare_text(text, stop_words=True) for text in X_TEST]


X_stop_stem = [preprocesare_text(text, stop_words=True, stem=True) for text in X]
X_TEST_stop_stem = [preprocesare_text(text, stop_words=True, stem=True) for text in X_TEST]


X_stop_lemma = [preprocesare_text(text, stop_words=True, lemma=True) for text in X]
X_TEST_stop_lemma = [preprocesare_text(text, stop_words=True, lemma=True) for text in X_TEST]

X_stop_lemma_digits = [preprocesare_text(text, stop_words=True, lemma=True, digits=True) for text in
X]
X_TEST_stop_lemma_digits = [preprocesare_text(text, stop_words=True, lemma=True, digits=True) for
text in X_TEST]
```

In continuarea acestui proiect, voi analiza setul de date ce folosește preprocesarea cu stopwords și lematizare (X_stop_lemma), deoarece s-a dovedit să fie cea mai performantă dintre preprocesări.

Timpii de preprocesare a datelor au fost de aprox: **20 min**. pentru datele de antrenare (41570 texte), **80 min**. pentru datele de test (13853 texte). Această discrepanță de timpi este redată de detectarea limbii la preprocesare. Datele de test, față de datele de antrenare, sunt amestecate în mod **aleator** în funcție de limbă (în train_data fiind ordonate în funcție de limba), fapt care îngreunează procesul de identificare a limbii, dar și a lematizării.

Pasul 1.4. Train – Test – Validation

Pentru a diviza setul de date, sklearn ne pune la dispoziție modulul *model_selection în* care avem funcția **train_test_split()**.

```
from sklearn.model_selection import train_test_split

X_train, X_valid, y_train, y_valid = train_test_split(X_stop_lemma, y, test_size=0.2,
random_state=42)
```

Parametri:

- X_stop_lemma → datele preprocesate
- y: → etichetele
- test_size → dimensiunea datelor de test, în cazul nostru 20%
- random_state → se comportă ca un seed, controlează modul de amestecare a datelor

Pasul 2.1. Bag Of Words

Conceptul „**Bag of Words**” este o reprezentare utilizată în procesarea limbajului natural. Procesul **BoW**, zis și „sac de cuvinte”, reprezintă strategia de a descrie documentele prin apariții de cuvinte, unde se ignoră gramatica și ordinea cuvintelor, păstrându-se doar frecvența acestora.

Principalele etape ale **BoW**:

- **tokenizarea** șirurilor și oferirea unui ID întreg pentru fiecare simbol posibil
- **numărarea** aparițiilor de cuvinte în fiecare document.
- **normalizarea** și ponderarea cuvintelor de importanță în scădere care apar în majoritatea documentelor.

Numim **vectorizare** procesul general de transformare a unei colecții de documente text în vectori de caracteristici numerice. (tokenizare, numărare, normalizare)

Folosind acest proces, putem converti o colecție de documente într-o matrice, fiecare document fiind un rând și fiecare cuvânt fiind coloana, iar valorile corespunzătoare (rând, coloană) fiind frecvența apariției fiecărui cuvânt în documentul respectiv.

În librăria **sklearn** exista 2 metode de implementare a unui BoW:

i. CountVectorizer()
ii. TfidfVectorizer()

Într-un corpus de text mare, unele cuvinte vor fi foarte prezente (în special **cuvintele de legătură /stopwords**), transportând astfel foarte puține informații semnificative despre conținutul real al documentului. Dacă ar fi să transmitem datele de numărare directă direct unui clasificator, acei termeni foarte frecvenți **ar umbri frecvențele** termenilor mai rari, dar mai interesanți. Aici apare conceptul de **Tf-idf** (Term frequency – inverse document frequency).

Pasul 2.2. Vectorizarea Datelor cu TfidfVectorizer

```
# Vectorizarea datelor
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(strip_accents='unicode', sublinear_tf=True,
                             min_df=3 ,max_features=104000)
X_train = vectorizer.fit_transform(X_train)
X_valid = vectorizer.transform(X_valid)
X_TEST = vectorizer.transform(X_TEST_stop_lemma)
```

Parametri:

- **strip_accents** → Normalizarea caracterelor {'ascii, 'unicode'}. Deoarece avem caractere din limbi străine (ä, ö, ü, ø, æ, å) ce în normalizarea 'ascii' ar dispărea, preferăm 'unicode' pentru a păstra toate cuvintele în forma exactă.
- **sublinear_tf** → Aplicarea scalării tf subliniară, adică înlocuirea tf cu 1 + log(tf).
- **min_df** → Numărul minim de documente în care trebuie să apară un caracter/șir de caractere pentru a fi luat în considerare.
- **max_features** → Păstrează primele max_features cele mai relevante caracteristici ordonate în funcție de frecvența termenilor din corpus.

Pasul 3.1. Logistic Regression

Regresia logistica este un model statistic care folosește la bază **funcția logistică** pentru estimarea parametrilor unui model. În mod normal, regresia logistică clasifică un model binar (două clase).

In cazul nostru, unde avem mai multe clase, algoritmul de antrenare utilizează schema **One-vs-Rest** (OvR), ce ar putea reprezenta un clasificator puternic în antrenarea modelului nostru. În acest proiect vom folosi din modulul *sklearn.linear_model* clasificatorul **LogisticRegression**.

```
# Antrenarea modelului folosind Logistic Regression
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=0, C=2).fit(X_train, y_train)
y_pred = clf.predict(X_valid)
```

Parametri:

- C → **hiperparametru** ce controlează trade-off-ul dintre margine și acuratețe
- random_state → se comportă ca un seed, controlează modul de amestecare a datelor

Antrenarea modelului durează în jur de 6.3 secunde.

Hiperparapemtrii acestui proiect au fost aleși prin numeroase grid search-uri folosite din modulul *sklearn.model_selection* cu funcția **GridSearchCV**.

Pasul 3.2. Testarea modelului Logistic Regression

```python
# Testarea modelului pe datele de validare
from sklearn.metrics import accuracy_score, f1_score

print(f'Accuracy of Logistic Regression: {accuracy_score(y_valid, y_pred)*100:.3f} %')
print(f'F1 Score of Logistic Regression: {f1_score(y_valid, y_pred, average="macro")*100:.3f} %')
```

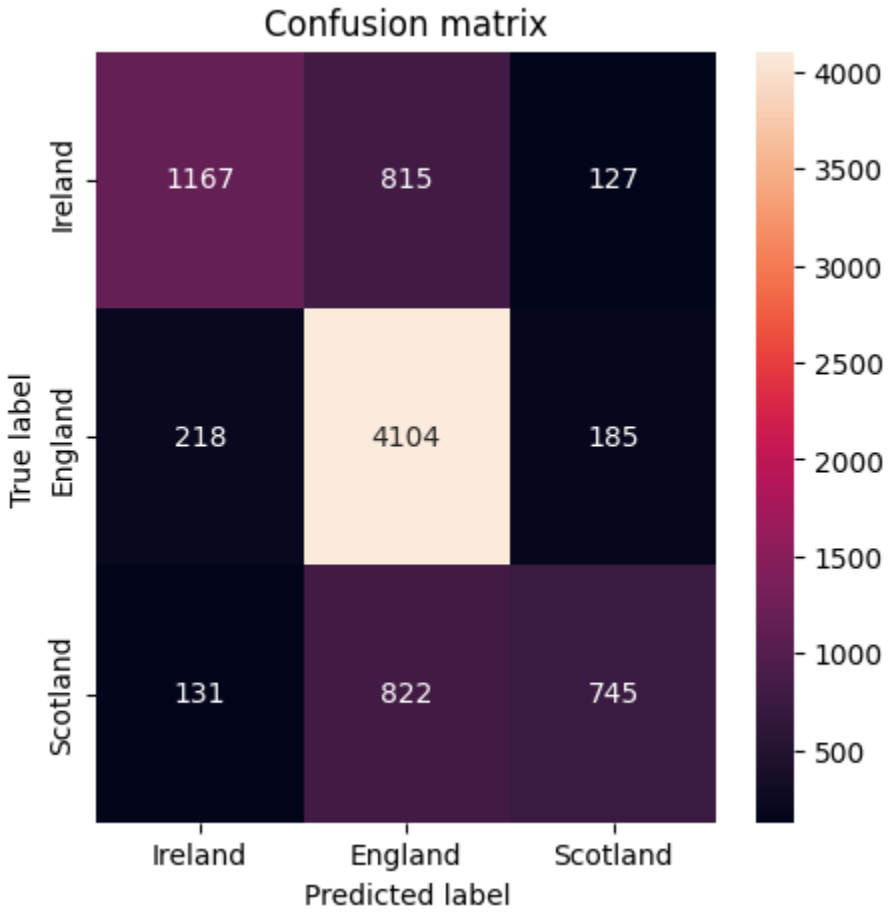
```
Accuracy of Logistic Regression: 72.360 %
F1 Score of Logistic Regression: 66.188 %
```

3.2.1. 5-Fold Cross-Validation

```python
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
accuracies = cross_val_score(estimator = clf, X = X_train, y = y_train, cv = cv)

print(f'Accuracy: {accuracies.mean()*100:.3f} %')
print(f'Standard Deviation: {accuracies .std()*100:.3f} %')
print(f'Accuracies: {accuracies}')
```

```
Accuracy: 72.826 %
Standard Deviation: 0.478 %
Accuracies: [0.71903187 0.72820204 0.73120866 0.73120866 0.73165965]
```

3.2.2. Matricea de confuzie



Putem observa cum majoritatea prezicerilor se îndreaptă spre Anglia. Acest fenomen apare deoarece majoritatea etichetelor din setul de date de antrenare aparțin Angliei, iar Logistic Regresiune, fiind un **algoritm probabilistic**, este de așteptat să influențeze predicția în funcție de probabilitatea de apariție a acestei clase.

Labels	Count
England	22700
Ireland	10535
Scotland	8335

3.2.3. Raportul de clasificare

```
# Raportul de clasificare
from sklearn.metrics import classification_report

print(classification_report(y_valid, y_pred, target_names=label2id.keys()))
```

	precision	recall	f1-score	support
Ireland	0.77	0.55	0.64	2109
England	0.71	0.91	0.80	4507
Scotland	0.70	0.44	0.54	1698
accuracy			0.72	8314
macro avg	0.73	0.63	0.66	8314
weighted avg	0.73	0.72	0.71	8314

Pasul 3.3. Predicția datelor de test pe Kaggle

```
# Predictia datelor de test
y_pred = clf.predict(X_TEST)

# Salvarea datelor in fisierul de output
prediction = [id2label[label] for label in y_pred]
submission = pd.DataFrame({'id':range(1, len(prediction) + 1),'label': prediction})
submission.to_csv('submission.csv', index=False)
```

Scorul Public (40% date test)	**Scorul Privat**
0.70580	0.69901

Pasul 4.1. SVM

SVM este al doilea clasificator pe care l-am ales pentru rezolvarea acestui exercițiu. Din librăria sklearn folosim **LinearSVC** – Linear Support Vector Classification. Similar **SVC**-ului, este implementat în termeni lineari, ce ii oferă mai multa flexibilitate în alegerea penalităților și a funcțiilor de pierdere, scalând mai bine la un număr mare de date.

```
# SVM
from sklearn.svm import LinearSVC

clf = LinearSVC(C=1.1, class_weight= {0: 0.63, 1: 0.33, 2: 0.63}).fit(X_TRAIN, y)

# predict
y_pred = clf.predict(X_TEST)
```

Parametri:

- C → **hiperparametru** ce controlează trade-off-ul dintre margine și acuratețe
- class_weight → greutatea claselor, $class\_weight[i] * C$, folosim în special din cauza discrepanței dintre etichete

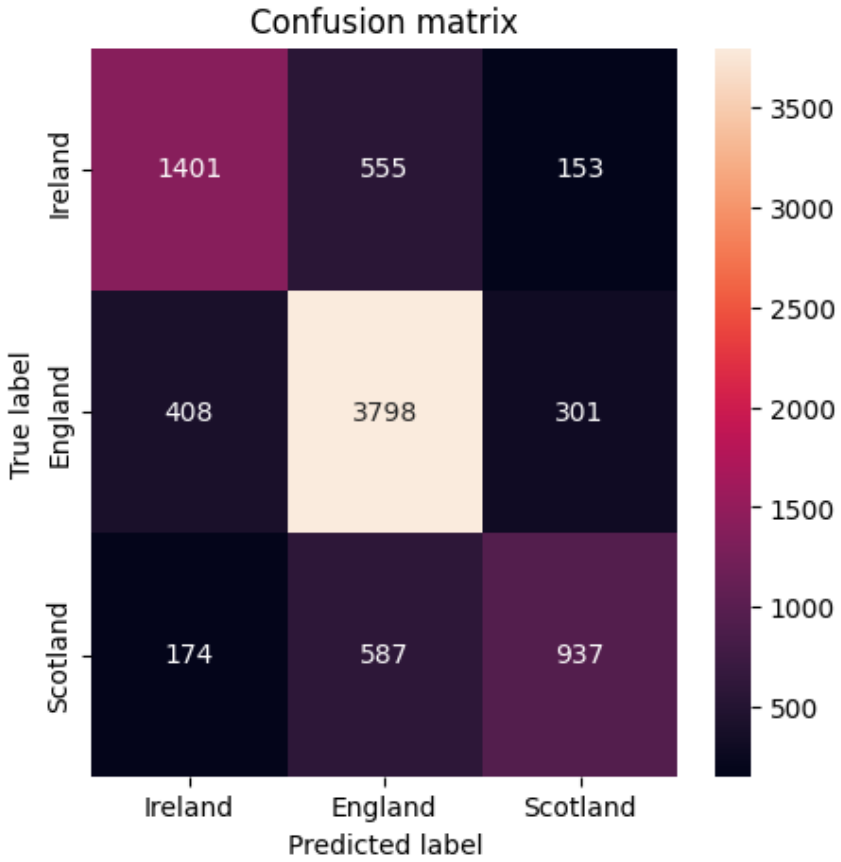
Timpul de antrenare al clasificatorului este de aprox. 1.9s

Pasul 4.2. Testarea modelului SVM

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
print(f'Accuracy of LinearSVC: {accuracy_score(y_valid, y_pred)*100:.3f} %')
print(f'F1 Score of LinearSVC: {f1_score(y_valid, y_pred, average="macro")*100:.3f} %')
```

```
Accuracy of LinearSVC: 73.803 %
F1 Score of LinearSVC: 69.849 %
```

4.2.1. Matricea de confuzie



4.2.2. Raportul de clasificare

	precision	recall	f1-score	support
Ireland	0.71	0.66	0.68	2109
England	0.77	0.84	0.80	4507
Scotland	0.67	0.55	0.61	1698
accuracy			0.74	8314
macro avg	0.72	0.69	0.70	8314
weighted avg	0.73	0.74	0.73	8314

4.3. Predicția datelor de test pe Kaggle

```
# Predictia datelor de test
y_pred = clf.predict(X_TEST)

# Salvarea datelor in fisierul de output
prediction = [id2label[label] for label in y_pred]
submission = pd.DataFrame({'id':range(1, len(prediction) + 1),'label': prediction})
submission.to_csv('submission.csv', index=False)
```

Scorul Public (40% date test)	**Scorul Privat**
0.69336	0.68939

După această întreagă analiză, observăm cum SVM-ul are un scor mai mare pe datele locale, iar pe Kaggle a scos un punctaj mai mic decât Logistic Regression, care la rândul lui a obținut un scor mai mic local decât SVM-ul.