Università degli Studi di Padova

Progetto per il corso
Apprendimento Automatico

Anno Accademico: 2017/2018

Studente:
Luca Dal Medico - 1099176

9 giugno 2018

Indice

1 Introduzione

1.1 Scopo del progetto

Il progetto vuole analizzare la precisione di alcuni algoritmi di classificazione per l'analisi di testo.
Il dataset utilizzato rappresenta i dati della serie TV "I Simpson", in particolare si andrà ad analizzare le battute di copione delle puntate di questa serie e l'algoritmo, data una frase, dovrà individuare quale personaggio della serie TV possa averla detta.

1.2 Sorgente dei dati

I dati utilizzati sono stati scaricati da Kaggle all'indirizzo https://bit.ly/2M50wt0.
I file utilizzato è *simpsons_script_lines.csv*.

N.B.
L'importazione dei dati presenti nel file *simpsons_script_lines.csv* presentava diversi warning dovuti ad errori nel file come ad esempio la mancanza di alcune virgolette. Il programma funziona ugualmente, stampando solo errori, ma per evitare queste segnalazioni è consigliato utilizzare il file *Data/simpsons_script_lines.csv* presente nella cartella del progetto che è stato corretto manualmente.

1.3 Librerie necessarie

- Sklearn: http://scikit-learn.org/stable/index.html;
- Scipy: https://www.scipy.org/;
- Pandas: https://pandas.pydata.org/;
- Pandas_ml: https://pypi.org/project/pandas_ml/;
- Numpy: http://www.numpy.org/;
- NLTK: https://www.nltk.org/.

1.4 File presenti

All'interno della cartella del progetto sono contenuti:

- le seguenti cartelle:
  - **Data:** che contiene i dati da importare;
  - **Saved_model:** contiene i classificatori salvati, sono necessari per fare i test;
  - **Utils:** file di utilità, cioè: *Classifier.txt* che contiene gli identificatori dei classificatori usati e *Saved_Vectorizer.plk* che permette di ricaricare il dizionario utilizzato durante il training, senza doverlo ricalcolare;
- i seguenti file:
  - **functions.py:** contiene le funzioni utilizzate all'interno del progetto;
  - **simpson.py:** va semplicemente a testare gli algoritmi con delle frasi date in input;
  - **training.py:** effettua il training degli algoritmi.

2 Analisi dei dati

Il dataset presenta le seguenti caratteristiche:

```
Dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158315 entries, 0 to 158314
Data columns (total 13 columns):
id                      158315 non-null int64
episode_id              158315 non-null int64
number                  158315 non-null int64
raw_text                158315 non-null object
timestamp_in_ms         158315 non-null int64
speaking_line           158315 non-null object
character_id            140789 non-null float64
location_id             157908 non-null float64
raw_character_text      140789 non-null object
raw_location_text       157908 non-null object
spoken_words            132151 non-null object
normalized_text         132125 non-null object
word_count              132148 non-null float64
dtypes: float64(3), int64(4), object(6)
memory usage: 15.7+ MB
None
```

I campi che ci interessano sono quelli contenenti la battuta di copione e quelli contenenti l' "id" del personaggio che l'ha detta (*character_id*), non utilizziamo la colonna *raw_character_text* perché per la classificazione ci servono valori numerici e la trasformazione di questo campo in un valore reale non avrebbe fatto altro che assegnarli un nuovo identificatore.
Per le battute di copione prendiamo il campo *normalized_text*, evitando così l'operazione di normalizzazione della stringa.

Le righe con dati mancanti sono state eliminate, vista la grandezza del dataset questo non comporta un eccessiva perdita d'informazione. Inoltre, per ridurre il rumore dovuto al gran numero di diversi personaggi presenti nella serie si è deciso di considerare solo i quattro personaggi con più battute e di utilizzare solo un sotto-gruppo del dataset iniziale perché il training di così tanti dati richiedeva prestazioni e tempi elevati.
Infine dividiamo i dati in training set e test set.

```
# We keep only the script lines where someone is talking
dataset_script_lines = dataset_script_lines[dataset_script_lines['speaking_line'] == 'true'].dropna
    ()

# Selecting the 4 characters who have the more lines and delete all other entries

selected_characters = dataset_script_lines['raw_character_text'].value_counts()[:4]

to_delete_from_script = []
for index, row in dataset_script_lines.iterrows():
    if row['raw_character_text'] not in selected_characters:
        to_delete_from_script.append(index)

dataset_script_lines = dataset_script_lines.drop(to_delete_from_script)


# Keep only a portion of the original dataset
dataset_script_lines = dataset_script_lines[:10000]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
Selected characters:
Homer Simpson     27851
Marge Simpson     13174
Bart Simpson      13000
Lisa Simpson      10754
Name: raw_character_text, dtype: int64
```

```
Train size: (6700, 5088)
Test size: (3300, 5088)
```

Si può notare che i dati sono sbilanciati, infatti il numero di battute per il personaggio “Homer Simpson” è molto maggiore rispetto agli altri personaggi.
Su alcuni classificatori è stato possibile impostare il campo “class_weight” in modo che questo non influenzasse la loro classificazione.

3 Preparazione dei dati

Per lavorare con valori testuali è stato necessario trasformare il testo delle battute in valori reali, per far ciò è stato utilizzato un oggetto di tipo `TfidfVectorizer` (documentazione alla pagina https://bit.ly/2sJCoVN).
L’oggetto è stato così dichiarato:

```
def Vectorizer(data):
  filename = 'Utils/Saved_Vectorizer.plk'

  if os.path.isfile(filename):
    vectorizer = joblib.load(filename)
  else:
    vectorizer = TfidfVectorizer(
        stop_words='english',
        ngram_range=(1, 2),
        min_df=1,
        sublinear_tf=True,
        tokenizer=LemmaTokenizer(),
        )


    vectorizer.fit(data)
    joblib.dump(vectorizer, filename)

  return vectorizer
```

La classe `LemmaTokenizer()` è dichiarata nello stesso file e sfrutta l’implementazione di un lemmmatizer presente nella libreria NTLK. Se questa libreria non fosse presente è possibile commentare le righe di codice inerenti all’importazione della libreria, la dichiarazione della classe e la riga che definisce il *tokenizer*. Questo però potrebbe portare a peggioramenti delle prestazioni.

I dati sono stati trasformati nel seguente modo:

```
vectorizer = Vectorizer(dataset_script_lines['normalized_text'])

X = vectorizer.transform(dataset_script_lines['normalized_text'])

y = np.asarray(dataset_script_lines.loc[:,'character_id'], dtype=np.float64)
```

4 Classificatori utilizzati

Per il compito di classificazione sono stati utilizzati i seguenti algoritmi:

4.1 Linear classifiers with SGD training.

Per questo classificatore è stata usata la classe `SGDClassifier`, la sua documentazione è presente all’indirizzo: https://bit.ly/22dnqoa.

Parametri

Per effettuare la Grind Search Cross Validation sono stati usati i seguenti parametri:

```
'loss': ['log','hinge'],
'alpha': [10 ** x for x in range(-6, 1)],
'n_jobs' : [-1],
```

Risultati

Migliori parametri	*alpha:* 0.0001, *n_jobs:* -1, *loss:* log
Validation score	0.530895522388
Accuracy	0.523939393939

Confusion matrix

Predicted / Actual	Marge Simpson	Homer Simpson	Bart Simpson	Lisa Simpson	**Total**
Marge Simpson	220	307	39	26	**592**
Homer Simpson	96	1154	106	46	**1402**
Bart Simpson	46	476	242	45	**809**
Lisa Simpson	52	260	72	113	**497**
Total	**414**	**2197**	**414**	**230**	**3300**

4.2 Naive Bayes classifier

Per questo classificatore è stata usata la classe `MultinomialNB`, la sua documentazione è presente all'indirizzo: https://bit.ly/22rC9bD.

Parametri

Per effettuare la Grind Search Cross Validation sono stati usati i seguenti parametri:

```
'alpha' : [0, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5],
'fit_prior' : [True, False],
```

Risultati

Migliori parametri	*alpha*: 1.75, *fit_prior*: False
Validation score	0.514776119403
Accuracy	0.49696969697

Confusion matrix

Predicted / Actual	Marge Simpson	Homer Simpson	Bart Simpson	Lisa Simpson	**Total**
Marge Simpson	220	321	30	21	**592**
Homer Simpson	148	1093	87	74	**1402**
Bart Simpson	84	441	222	62	**809**
Lisa Simpson	63	252	77	105	**497**
Total	**515**	**2107**	**416**	**262**	**3300**

4.3 Random forest classifier

Per questo classificatore è stata usata la classe `RandomForestClassifier`, la sua documentazione è presente all'indirizzo: https://bit.ly/2fDrBrA.

Parametri

Per effettuare la Grind Search Cross Validation sono stati usati i seguenti parametri:

```
'n_estimators' : [3, 10, 50],
'class_weight' : ['balanced', None],
'n_jobs' : [-1],
```

Risultati

Migliori parametri	*n_estimators:* 50, *n_jobs:* −1, *class_weight:* balanced
Validation score	0.528059701493
Accuracy	0.513333333333

Confusion matrix

Predicted / Actual	Marge Simpson	Homer Simpson	Bart Simpson	Lisa Simpson	**Total**
Marge Simpson	171	365	37	19	**592**
Homer Simpson	75	1204	76	47	**1402**
Bart Simpson	46	505	205	53	**809**
Lisa Simpson	32	289	62	114	**497**
Total	**324**	**2363**	**380**	**233**	**3300**

5 Test

Alcuni test dei classificatori utilizzando il file *simpson.py*:

```
python simpsons.py "d'oh!"

SGDClassifier
-> Homer Simpson

MultinomialNB
-> Homer Simpson

RandomForestClassifier
-> Homer Simpson
```

```
python simpsons.py "if anyone wants me, I'll be in my room"

SGDClassifier
-> Lisa Simpson

MultinomialNB
-> Homer Simpson

RandomForestClassifier
-> Lisa Simpson
```

```
python simpsons.py "I don't think that's a good idea Homer"

SGDClassifier
-> Marge Simpson

MultinomialNB
-> Marge Simpson
```

```
RandomForestClassifier
-> Marge Simpson
```

```
python simpsons.py "eat my shorts!"
SGDClassifier
-> Bart Simpson

MultinomialNB
-> Homer Simpson

RandomForestClassifier
-> Bart Simpson
```

Proviamo introducendo anche errori di battitura:

```
python simpsons.py "eat ymm Shorst!"
SGDClassifier
-> Bart Simpson

MultinomialNB
-> Homer Simpson

RandomForestClassifier
-> Bart Simpson
```

6 Conclusioni

I classificatori hanno una discreta precisione, il valore riportato per per la precisione è infatti intorno al 50% circa per ognuno ma riescono comunque a riconoscere frasi tipiche di certi personaggi. Il più impreciso è il *Naive Bayes classifier* con un risultato leggermente inferiore agli altri. Dai test si nota che, quando sbaglia, tende ad assegnare la frase a "Homer Simpson" che è la classe più presente nel dataset. Questi errori potrebbero essere colpa dello sbilanciamento del dataset, che porta spesso a classificare la maggior parte dei casi con l'etichetta più presente.

L'utilizzo della tecnica di *Lemmatisation* aumenta molto la precisione ed inoltre permette di riconoscere frasi anche con errori di battitura, cosa non possibile se avessi semplicemente trasformato le parole da stringhe a valori reali.

Per migliorare ulteriormente la precisione degli algoritmi è possibile utilizzare tutto il dataset, aumentare i parametri utilizzati nella Cross Validation oppure utilizzare tecniche più complesse come una rete neurale multistrato, sia per la classificazione che per la conversione delle parole nelle battute in valori reali (utilizzando ad esempio *Word2vec*). Per far ciò sono però necessarie risorse computazionali maggiori.