Contents

1 Executive Summary

In this notebook, I am going to use Embeddings from Language Models (ELMO) for text classification. Unlike traditional word embeddings such as word2vec and GLoVe, the ELMo vector assigned to a token or word is actually a function of the entire sentence containing that word. Therefore, the same word can have different word vectors under different contexts.

2 Introduction

In the "Stack Overflow" database, we have two seperate tables which keep the track of submitted questions and answers. Each submitted question has a "tag" attached to it which shows the relevancy of the question with a certain technology or topic. In this notebook, I aim to build a logistic regression model for classifing the tags in our dataset.

3 Importing the Libraries

In [1]:

```
#importing the OS library↔
```

In [16]:

```
# Importing the libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
```

In [17]:

```
import tensorflow as tf

#TensorFlow Hub is a library for reusable machine learning modules. It consists of a large number
# of pre-trained models for use in TensorFlow. One of these models is ELMO.
import tensorflow_hub as hub
elmo = hub.Module("https://tfhub.dev/google/elmo/2", trainable=True)
```

4 Loading Data

In [18]:

```
dataset=pd.read_csv("tagdataset.csv")
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5518798 entries, 0 to 5518797
Data columns (total 3 columns):
 #   Column      Dtype 
---  ------      ----- 
 0   Unnamed: 0  int64 
 1   tags        object
 2   label       int64 
dtypes: int64(2), object(1)
memory usage: 126.3+ MB
```

In [19]:

```
dataset= dataset.rename(columns={'Unnamed: 0':'Id'})
```

In [20]:

```
dataset= dataset.set_index('Id')
```

In [21]:

```
# The dataset is big so let us focus on 1000000 radom samples
dataset2=dataset.sample(n = 10000, replace = False)
dataset2
```

Out[21]:

	tags	label
Id		
5472402	javascript jquery html	0
3364999	javascript json	0
5297551	javascript jquery	0
2331542	java hibernate many-to-many criteria	0
4069747	java arrays list spring-boot loops	0
...	...	...
24420	python pip jupyter-notebook	1
1920263	java maven spring-boot	0
5297137	javascript jquery	0
2621475	javascript reactjs material-ui	0
4185237	javascript jquery datatable	0

10000 rows × 2 columns

In [22]:

```
# We need to extract the ELMO features first. The following  function tells the model to run through the
#'sentences' list and return the default output (1024 dimension sentence vectors) for eac of them. Then
# we start a session and run ELMO to return the embeddings in variable x
def elmo_vectors(x):
    embeddings = elmo(
        x.tolist(),
        signature="default",
        as_dict=True)["elmo"]

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        sess.run(tf.tables_initializer())
        return sess.run(tf.reduce_mean(embeddings,1))
#embeddings.shape
```

In [23]:

```
# train test split 70-30%
from sklearn.model_selection import train_test_split
train, test = train_test_split(dataset2[['tags','label']], test_size=0.3, shuffle=True)
```

In [24]:

```
# Let us make buckets of size 100 records for our training and testing test
list_train = [train[i:i+100] for i in range(0,train.shape[0],100)]
list_test = [test[i:i+100] for i in range(0,test.shape[0],100)]
```

Let us import spaCy's language model.spaCy expects all model packages to follow the naming convention of [lang_[name]]. en_core_web_sm is a small (sm) English model trained on written web text (blogs, news, comments), that includes vocabulary, vectors, syntax and entities. I don't want some particular component of the pipeline (the parser and Named-entity recognition (NER)), I disable loading it. This can sometimes make a big difference and improve loading speed. Named-entity recognition (NER) is the process of automatically identifying the entities discussed in a text and classifying them into pre-defined categories such as 'person', 'organization', 'location' and so on.

Contents

5 Extract ELMO Embeddings

In [25]:
```
import en_core_web_sm
nlp = en_core_web_sm.load(disable=['parser', 'ner'])
```

In [ ]:
```
# Extract ELMo embeddings
elmo_train = [elmo_vectors(x['tags']) for x in list_train]
elmo_test = [elmo_vectors(x['tags']) for x in list_test]
```

In [27]:
```
# Concatenate the batches
elmo_train_new = np.concatenate(elmo_train, axis = 0)
elmo_test_new = np.concatenate(elmo_test, axis = 0)
```

The pickle module is imported. Then I use pickle.dump(object, filename) method to save the object into a file in byte format. I do it for both our training and our testing data.

In [28]:
```
try:
    import dill as pickle
except ImportError:
    import pickle
# a new file is opened in write-bytes “wb” mode.
pickle_out = open("elmo_train.pickle000","wb")
# the list will be saved to this file using pickle.dump() method.
pickle.dump(elmo_train_new, pickle_out)
# the file is closed
pickle_out.close()

pickle_out = open("elmo_test.pickle000","wb")
pickle.dump(elmo_test_new, pickle_out)
pickle_out.close()
```

Later I can use pickle.load(filename) to load back python object from the file where it was dumped before. I do it for both our training and our testing data. File in which the list was dumped is opened in read-bytes "RB" mode.

In [95]:
```
#pickle_in = open("elmo_train.pickle000", "rb")
#elmo_train_new = pickle.load(pickle_in)
#pickle_in = open("elmo_test.pickle000", "rb")
#elmo_test_new = pickle.load(pickle_in)
```

In [78]:
```
# Train test split 70/30
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(elmo_train_new,
                                                    train['label'],
                                                    random_state=101,
                                                    test_size=0.3)

```

In [30]:
```
x_train.shape
```

Out[30]:
```
(4900, 1024)
```

In [31]:
```
y_train.shape
```

Out[31]:
```
(4900,)
```

6 Building a Logistic Regression Model

In [79]:
```
# Build a logistic regression model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score

lreg = LogisticRegression()
lreg.fit(x_train, y_train)
```

Out[79]:
```
LogisticRegression()
```

In [81]:
```
preds = lreg.predict(x_test)
```

In [82]:
```
round(f1_score(y_test, preds),3)
```

Out[82]:
```
0.99
```

In [83]:
```
preds_test = lreg.predict(elmo_test_new)
```

In [84]:
```
pred_df = pd.DataFrame({'id':test.index, 'label':preds_test})

# write the prediction dataframe to a CSV file
pred_df.to_csv("results.csv", index=False)
```

7 Confusion Matrix

Contents

In [89]:

```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
cm = confusion_matrix(y_test, preds)

print("Accuracy: ",round(accuracy_score(y_test,preds),3))
print("F1: ",round(f1_score(y_test, preds),3))
print(classification_report(y_test,preds))
```

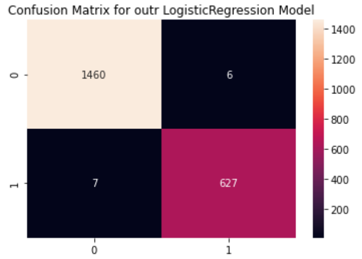
```
Accuracy:  0.994
F1:  0.99
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1466
           1       0.99      0.99      0.99       634

    accuracy                           0.99      2100
   macro avg       0.99      0.99      0.99      2100
weighted avg       0.99      0.99      0.99      2100
```

In [86]:

```python
import seaborn as sns
# plot the confusion matrix
ax = plt.axes()
sns.heatmap(cm, annot=True, fmt="d")
ax.set_title('Confusion Matrix for outr LogisticRegression Model')
```

Out[86]:

```
Text(0.5, 1.0, 'Confusion Matrix for outr LogisticRegression Model')
```


Confusion Matrix for outr LogisticRegression Model