TaskB_LSTM

November 27, 2024

[1]:
```
import nltk
import keras
import pandas as pd
import numpy as np
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from keras.layers import TextVectorization
from keras.models import Sequential
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
```

[2]:
```
"""
Download dataset SubtaskB.jsonl from
https://github.com/mbzuai-nlp/M4GT-Bench.
"""
DATA_PATH = "C:/Users/Admin/Desktop/cse847_proj/SubtaskB.jsonl"

# initialize dataframe
df = pd.read_json(DATA_PATH, lines=True)
```

[3]:
```
print(df.source.value_counts())
print()
print(df.model.value_counts())
```

```
source
wikihow      23556
reddit       20999
outfox       20999
arxiv        20998
wikipedia    19368
peerread     16891
Name: count, dtype: int64

model
bloomz           17332
human            17179
```

```
chatGPT          16892
cohere           16678
gpt4             14344
davinci          14340
dolly            14046
gpt-3.5-turbo     6000
davinci-003       3000
dolly-v2-12b      3000
Name: count, dtype: int64
```

[7]:
```
print(df.label.unique())
```

```
[2 1 0 3 5 4 6]
```

[4]:
```
df[['text', 'label']]
```

[4]:

	text	label
0	We consider a system of many polymers in solut…	2
1	We present a catalog of 66 YSOs in the Serpens…	2
2	Spectroscopic Observations of the Intermediate…	2
3	We present a new class of stochastic Lie group…	2
4	ALMA as the ideal probe of the solar chromosph…	2
…	…	…
122806	Title: The Unsung Heroes: Seagoing Cowboys and…	0
122807	Title: The Benefits of Autonomy: Student-led P…	0
122808	The Electoral College system, established by t…	0
122809	In the ever-evolving landscape of education, c…	0
122810	When faced with critical decisions, the wise o…	0

[122811 rows x 2 columns]

[8]:
```
"""
Pre-process dataframe.
"""
MAX_VOCAB = 10_000
MAX_LENGTH = 200

# init text vectorizer
vectorize_layer = TextVectorization(
    max_tokens=MAX_VOCAB,
    standardize='lower_and_strip_punctuation',
    split='whitespace',
    ngrams=None,
    output_mode='int',
    output_sequence_length=MAX_LENGTH,
    pad_to_max_tokens=False,
    vocabulary=None,
    idf_weights=None,
```

```
    sparse=False,
    ragged=False,
    encoding='utf-8',
    name=None,
)

# create vocabulary
vectorize_layer.adapt(df['text'])
vocab = vectorize_layer.get_vocabulary()
```

[9]:
```
# vectorize text data (in subsets for memory constraints)
X = []
y = df['label']

subset_size = df.shape[0] // 100
for i in range(0, df.shape[0], subset_size):
    subset = df['text'][i : i + subset_size]
    X.append(vectorize_layer(subset))

X = np.vstack(X)
print(X.shape, y.shape)
```

```
(122811, 200) (122811,)
```

[10]:
```
"""
LSTM model generator.
"""
EMBEDDING_DIM = 128
N_HIDDEN = 100
OPTIMIZER = 'adam'
N_CLASSES = 7

def get_model(model_path=None):
    if model_path:
        # load existing model
        model = keras.models.load_model(model_path)
    else:
        # create new model
        model = Sequential()
        embeddings = Embedding(
            input_dim=MAX_VOCAB,
            output_dim=EMBEDDING_DIM,
        )
        model.add(embeddings)
        model.add(LSTM(N_HIDDEN, return_sequences=False))
        model.add(Dense(N_CLASSES)) # , activation='softmax'
        model.compile(
            loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
```

```
            optimizer=OPTIMIZER,
            metrics=['accuracy']
        )
    return model
```

[12]:
```
"""
Train and evaluate model.
"""
# create new model
model = get_model()

# create data splits
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.15,
    random_state=777,
)

# train the model
model.fit(
    X_train,
    y_train,
    epochs=10,
    batch_size=64
)

# final evaluation of the model
scores = model.evaluate(
    X_test,
    y_test,
    verbose=0
)
accuracy = scores[1]

# report results
print("Accuracy: %.2f%%" % (accuracy * 100))

# save model
model.save("models/taskB_lstm.keras")
```

```
Epoch 1/10
1632/1632              256s 155ms/step
- accuracy: 0.3499 - loss: 1.6626
Epoch 2/10
1632/1632              251s 154ms/step
- accuracy: 0.6444 - loss: 0.9270
Epoch 3/10
```

```
1632/1632             250s 153ms/step
- accuracy: 0.8015 - loss: 0.5502
Epoch 4/10
1632/1632             253s 155ms/step
- accuracy: 0.8765 - loss: 0.3587
Epoch 5/10
1632/1632             259s 158ms/step
- accuracy: 0.9159 - loss: 0.2512
Epoch 6/10
1632/1632             261s 160ms/step
- accuracy: 0.9452 - loss: 0.1685
Epoch 7/10
1632/1632             264s 162ms/step
- accuracy: 0.9630 - loss: 0.1162
Epoch 8/10
1632/1632             266s 163ms/step
- accuracy: 0.9761 - loss: 0.0760
Epoch 9/10
1632/1632             267s 163ms/step
- accuracy: 0.9838 - loss: 0.0519
Epoch 10/10
1632/1632             269s 165ms/step
- accuracy: 0.9870 - loss: 0.0410
Accuracy: 80.39%
```

[10]: