NLP TP3 REPORT

Model Improvement

SEHILI CHAIMA

GROUP 01

2024

The previous models that achieved approximately **66%** accuracy on the test set employed **TF-IDF** and **one-hot encoding** vectorization techniques, so I chose to use the TF-IDF vectorization technique and do the study on multiple machine learning algorithms and deep neural network architectures.

PREVIOUSLY USED ARCHITECTURE:

A simple **MLP** consists of a **single hidden layer** with **50 neurons**. It utilizes the **ReLU** activation function, employs the **Adam** solver for optimization,

```
tfidf_classifier = MLPClassifier(hidden_layer_sizes=(50),
                                 max_iter=300,activation = 'relu',
                                 solver='adam',
                                 random_state=0)
```

```
MLPClassifier
MLPClassifier(hidden_layer_sizes=50, max_iter=300, random_state=0)
```

PREVIOUS RESULTS :

```
Classification Report for TF-IDF Representation:
              precision    recall  f1-score   support

           0       0.73      0.71      0.72      2370
           1       0.73      0.70      0.71      1690
           2       0.69      0.70      0.69      1814

   micro avg       0.71      0.70      0.71      5874
   macro avg       0.71      0.70      0.71      5874
weighted avg       0.71      0.70      0.71      5874
 samples avg       0.74      0.70      0.69      5874

Accuracy for TF-IDF Representation: 0.665645216207014
```

TOKENIZATION TECHNIQUE UPDATE:

RULE BASED TOKENIZATION (TREEBANK TOKENIZER)

```python
from nltk.tokenize import TreebankWordTokenizer
def rule_based_tokenization(text):
    tokenizer=TreebankWordTokenizer()
    return tokenizer.tokenize(text)
```

```
process, howev, afford, mean, ascertain, di
    [never, occur, fumbl, might, mere, mi
  [left, hand, gold, snuff, box, caper, hill,
```

After the change of the tokenization method from subword sokenization using sentence piece to rule based tokenization using treebank and holding the same MLP architecture, i noticed an improvement in the model's accuray to 75%.

Results after changing the tokenization technique and keeping the same MLP architecture :

```
Classification Report for TF-IDF Representation:
              precision    recall  f1-score   support

           0       0.76      0.78      0.77      1580
           1       0.78      0.72      0.75      1127
           2       0.72      0.75      0.73      1209

    accuracy                           0.75      3916
   macro avg       0.75      0.75      0.75      3916
weighted avg       0.75      0.75      0.75      3916

Accuracy for TF-IDF Representation: 0.7502553626149132
```

Keeping the rule based tokenization technique and move forward to building our models ...

Splitting the data to x_train, x_test, y_train, y_test and showing the corresponding shape of each:

```
X shape: (19579, 1)
X train shape: (15663, 1)
X test shape: (3916, 1)
Y shape: (19579,)
Y train shape: (15663,)
Y test shape: (3916,)
```

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=0,
                                                    stratify=y)
```

MACHINE LEARNING ALGORITHMS:

1- LOGISTIC REGRESSION:

```
from sklearn import linear_model
classes=np.unique(y_train)
lr = linear_model.LogisticRegression(multi_class='multinomial',
                                     solver='sag',
                                     class_weight={c: 1 for c in classes})
lr.fit(x_train_tfidf, y_train)
```

✓ 4m 3.5s

```
LogisticRegression
LogisticRegression(class_weight={0: 1, 1: 1, 2: 1}, multi_class='multinomial',
                   solver='sag')
```

	precision	recall	f1-score	support
0	0.78	0.87	0.82	1580
1	0.83	0.78	0.80	1127
2	0.84	0.77	0.80	1209
accuracy			0.81	3916
macro avg	0.82	0.80	0.81	3916
weighted avg	0.81	0.81	0.81	3916

The classification report, accuracy and log loss:

ACCURACY:
0.8097548518896833

LOG LOSS LR:
0.5421346903824495

2- SIMPLE NAIVE BAYES:

```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(x_train_tfidf, y_train)
```

✓ 0.8s

```
MultinomialNB
MultinomialNB()
```

	precision	recall	f1-score	support
0	0.76	0.88	0.82	1580
1	0.90	0.73	0.81	1127
2	0.83	0.81	0.82	1209
accuracy			0.82	3916
macro avg	0.83	0.81	0.81	3916
weighted avg	0.82	0.82	0.82	3916

The classification report, accuracy and log loss:

ACCURACY:
0.8153728294177732

LOG LOSS NB:
0.584931054230106

- VERY FAST TRAINING AND PREDICTION TIME!

3- SVM (SUPPORT VECTOR MACHINES):

```
from sklearn.svm import SVC
svm= SVC(kernel='linear',probability=True)
svm.fit(x_train_tfidf,y_train)
```

```
▾ SVC
SVC(kernel='linear', probability=True)
```

ACCURACY:
0.8010725229826353

LOG LOSS SVM:
0.47319472036205884

- VERY LONG TRAINING AND PREDICTION TIME.

4- XGBOOST:

```
import xgboost as XGB
params = {
            'objective':'binary:logistic',
            'max_depth': 4,
            'alpha': 10,
            'learning_rate': 0.1,
            'n_estimators':100
        }

xgb_clf = XGB.XGBClassifier(**params)
xgb_clf.fit(x_train_tfidf, y_train)
```

```
▾ XGBClassifier
XGBClassifier(alpha=10, base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None,
              num_parallel_tree=None, ...)
```

ACCURACY:
0.5967824310520939

LOG LOSS XGB:
0.9029549654440971

The classification report, accuracy and log loss:

	precision	recall	f1-score	support
0	0.53	0.90	0.66	1580
1	0.74	0.38	0.50	1127
2	0.77	0.40	0.53	1209
accuracy			0.60	3916
macro avg	0.68	0.56	0.56	3916
weighted avg	0.66	0.60	0.58	3916

IMPROVING THE DEEP NEURAL NETWORK ARCHITECTURE:

I Improved the previoualy used architecture, to a **2 hidden layers** and added a **dropout layer** with **dropout rate 0.3**, below the model architecture and the results obtained after **10 epochs**:

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 128)               1797760

 dropout_4 (Dropout)         (None, 128)               0

 dense_13 (Dense)            (None, 64)                8256

 dense_14 (Dense)            (None, 3)                 195

=================================================================
Total params: 1806211 (6.89 MB)
Trainable params: 1806211 (6.89 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
Deep Neural Network - Train accuracy: 1.0
Deep Neural Network - Test accuracy: 0.81
```

	precision	recall	f1-score	support
0	0.80	0.84	0.82	1580
1	0.83	0.80	0.82	1127
2	0.81	0.78	0.80	1209
accuracy			0.81	3916
macro avg	0.81	0.81	0.81	3916
weighted avg	0.81	0.81	0.81	3916

Adding a **dropout layer** and a **batch normalization** layer:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               1797760

 dropout (Dropout)           (None, 128)               0

 batch_normalization (Batch  (None, 128)               512
 Normalization)

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 3)                 195

=================================================================
Total params: 1806723 (6.89 MB)
Trainable params: 1806467 (6.89 MB)
Non-trainable params: 256 (1.00 KB)
```

```
Deep Neural Network - Train accuracy: 1.0
Deep Neural Network - Test accuracy: 0.8
```

	precision	recall	f1-score	support
0	0.79	0.82	0.80	1580
1	0.80	0.81	0.81	1127
2	0.81	0.76	0.78	1209
accuracy			0.80	3916
macro avg	0.80	0.80	0.80	3916
weighted avg	0.80	0.80	0.80	3916

CONCLUSION:

All the depp neural network models and machine learning algorithms gave approxiamte results for this task.