13-jc-data-cleaning-gridsearch

January 16, 2018

Using a LSTM single model to text various cleaning steps and impact on score.
Controls: - CNN single model - maxlen: 65 - min occurance vocab: 5 - glove.6B.100D - epochs: 2 - cv: 3 - max features 20000

In [1]:
```
model_name = 'grid_benchmark'
```

In [2]:
```
import os
```

In [3]:
```
dir_path = os.path.realpath('..')
```

In [4]:
```
# Import custom transformers

        path = 'src/features'
        full_path = os.path.join(dir_path, path)
        import sys
        sys.path.append(full_path)
        from transformers import TextCleaner, KerasProcesser
```

```
Using TensorFlow backend.
```

0.1 Import data

In [5]:
```
import numpy as np
        import pandas as pd
```

In [6]:
```
path = 'data/raw/train.csv'

        full_path = os.path.join(dir_path, path)
        df_train = pd.read_csv(full_path, header=0, index_col=0)
        print("Dataset has {} rows, {} columns.".format(*df_train.shape))
```

```
Dataset has 95851 rows, 7 columns.
```

In [7]:
```
# fill NaN with string "unknown"
        df_train.fillna('unknown',inplace=True)
```

0.2 Pre-processing

In [8]:
```
from sklearn.model_selection import train_test_split
```

In [9]:
```
seed = 42
np.random.seed(seed)
test_size = 0.2
target = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
corpus = 'comment_text'

X = df_train[corpus]
y = df_train[target]


Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=test_size, random_state
```

In [10]:
```
max_features=20000
max_length=65
```

0.3 Model fit

In [11]:
```
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.pipeline import Pipeline
```

In [12]:
```
from keras.models import Sequential
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation

# Function to create model, required for KerasClassifier
def create_model(optimizer='adam', max_features=max_features, max_length=max_length):
    model = Sequential()
    model.add(Embedding(max_features, 100, input_length=max_length))
    model.add(Bidirectional(LSTM(50, return_sequences=True, dropout=0.1, recurrent_dr
    model.add(GlobalMaxPool1D())
    model.add(Dense(50, activation="relu"))
    model.add(Dropout(0.1))
    model.add(Dense(6, activation='sigmoid'))  #multi-label (k-hot encoding)
    # compile network
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

In [13]:
```
def save_model(model, model_path):
    # serialize model to JSON
    model_json = model.to_json()
    with open(model_path + ".json", "w") as json_file:
        json_file.write(model_json)
    # serialize weights to HDF5
    model.save_weights(model_path + ".h5")
    print("Saved model to disk")
```

```
In [14]: model = KerasClassifier(build_fn=create_model, epochs=3, verbose=2)

In [15]: p = Pipeline([
             ('cleaner', TextCleaner()),
             ('keraser', KerasProcesser(num_words=max_features, maxlen=max_length)),
             ('clf', model)
         ])

         param_grid = {"cleaner__regex": ['\S+'],
                       "cleaner__remove_digits": [False],
                       "cleaner__english_only": [False],
                       "cleaner__stop_words": [None],
                       "cleaner__filters": [r'[!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n]'],
                       "cleaner__lower": [True],
                       "keraser__num_words": [max_features],
                       "keraser__maxlen": [max_length]
                      }

In [16]: %%time
         grid = GridSearchCV(p, param_grid=param_grid, cv=3)
         grid_result = grid.fit(Xtrain, ytrain)

Epoch 1/3
 - 332s - loss: 0.0930 - acc: 0.9726
Epoch 2/3
 - 332s - loss: 0.0556 - acc: 0.9808
Epoch 3/3
 - 331s - loss: 0.0486 - acc: 0.9825
Epoch 1/3
 - 331s - loss: 0.0926 - acc: 0.9733
Epoch 2/3
 - 330s - loss: 0.0558 - acc: 0.9807
Epoch 3/3
 - 331s - loss: 0.0469 - acc: 0.9826
Epoch 1/3
 - 338s - loss: 0.0924 - acc: 0.9729
Epoch 2/3
 - 337s - loss: 0.0541 - acc: 0.9811
Epoch 3/3
 - 337s - loss: 0.0453 - acc: 0.9832
Epoch 1/3
 - 498s - loss: 0.0837 - acc: 0.9748
Epoch 2/3
 - 498s - loss: 0.0549 - acc: 0.9807
Epoch 3/3
 - 495s - loss: 0.0479 - acc: 0.9824
CPU times: user 3h 7min 12s, sys: 49min 19s, total: 3h 56min 32s
Wall time: 1h 20min 54s
```

```
In [17]: trained_model = grid_result.best_estimator_.named_steps['clf'].model

In [18]: # save the model
         model_path = os.path.join(dir_path, 'models', model_name)
         save_model(trained_model, model_path)
```

```
Saved model to disk
```

0.4 Evaluation

```
In [19]: # summarize results
         means = grid_result.cv_results_['mean_test_score']
         stds = grid_result.cv_results_['std_test_score']
         params = grid_result.cv_results_['params']
         for mean, stdev, param in zip(means, stds, params):
             print("%f (%f) with: %r" % (mean, stdev, param))

         print("Best score {} with params {}".format(grid_result.best_score_, grid_result.best_
```

```
0.947359 (0.003732) with: {'cleaner__lower': True, 'keraser__num_words': 20000, 'cleaner__filte
Best score 0.9473591538786453 with params {'cleaner__lower': True, 'keraser__num_words': 20000
```