BERT: Multilabel Text Classification

Zuzanna Deutschman Follow
Feb 9 · 3 min read ★

Introduction

In my previous article, I introduced various machine learning methods that enable assigning a set of relevant genres for a single movie description (please visit the article for dataset). The best F1 score = 0.43 was obtained for Classifier Chain model. My idea to be verified is to train neural network with BERT embeddings.

BE
re

It's main innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling. This results in a deeper sense of language context and flow than single-direction language models.

Code

Bert_serving enables using BERT model as a sentence encoding service for mapping a variable-length sentence to a fixed-length.

To find the best bunch of parameters I used sacred module. Sacred is a tool to help you configure, organize, log and reproduce experiments in order to:

- keep track of all the parameters of your experiment
- easily run your experiment for different settings
- save configurations for individual runs in a database

```
1  from bert_serving.server import BertServer
2  from bert_serving.server.helper import get_args_parser
3  from bert_serving.server.helper import get_shutdown_parser
```

```
from sklearn.preprocessing import MultiLabelBinarizer
import json
import os
import tensorflow as tf
from keras import backend as K
import collections
import numpy as np
from sacred import Experiment
from sacred.observers import MongoObserver
ex = Experiment()
ex.observers.append(MongoObserver(
    #url='mongodb://mongo_user:mongo_password@localhost:27017/?authMechanism=SCRAM-SHA-

base_path = ''


def prepare_data():
    trainfilename = 'movies_train.csv'
    evalfilename = 'movies_eval.csv'

    if os.path.exists(trainfilename) and os.path.exists(evalfilename):
        train_df = pd.read_pickle(trainfilename)
        eval_df = pd.read_pickle(evalfilename)

        return train_df, eval_df

    #data preprocessing
    meta = pd.read_csv(os.path.join(
        base_path, "movie.metadata.tsv"), sep='\t', header=None)
    meta.columns = ["movie_id", 1, "movie_name", 3, 4, 5, 6, 7, "genre"]
```

```
    plots = pd.read_csv(os.path.join(
        base_path, "plot_summaries.txt"), sep='\t', header=None)
    plots.columns = ["movie_id", "plot"]

    genres['movie_id'] = genres['movie_id'].astype(str)
    plots['movie_id'] = plots['movie_id'].astype(str)
    movies = pd.merge(plots, genres, on='movie_id')

    genres_lists = []
    for i in movies['genre']:
        genres_lists.append(list(json.loads(i).values()))
    movies['genre'] = genres_lists
    multilabel_binarizer = MultiLabelBinarizer()
    multilabel_binarizer.fit_transform(movies['genre'])
    # transform target variable
    y = multilabel_binarizer.transform(movies['genre'])
    for idx, genre in enumerate(multilabel_binarizer.classes_):
        movies[genre] = y[:, idx]
    movies.to_csv('movies.csv')
    movies_new = pd.read_csv('movies.csv')
    movies = movies_new
    movies_columns = movies.columns

    del movies['Unnamed: 0']
    del movies['movie_name']
    del movies['genre']

    df = pd.DataFrame()
    df['id'] = movies['movie_id']
    df['labels'] = list(map(list, zip(*[movies[col] for col in movies
```

```
    TRAIN_VAL_RATIO = 0.8
    LEN = df.shape[0]
    SIZE_TRAIN = int(TRAIN_VAL_RATIO*LEN)

    train_df = df[:SIZE_TRAIN].drop(labels='id', axis=1)
    eval_df = df[SIZE_TRAIN:]

    train_df.to_pickle(trainfilename)
    eval_df.to_pickle(evalfilename)

    return train_df, eval_df



def encode_with_bert(train_df, eval_df, max_seq_len = 50):
    #bert-serving-start -model_dir L-12_H-768_A-12/ -num_worker=2
    filename = f'max_seq_len_{max_seq_len}'
    trainfile = f'{filename}_train.npy'
    testfile = f'{filename}_test.npy'
    if not os.path.exists(trainfile) or not os.path.exists(testfile):
        args = get_args_parser().parse_args(['-model_dir', 'uncased_L-12_H-768_A-12/',
                                             '-max_seq_len', f'{max_seq_len}',
                                             '-port', '5555',
                                             '-fp16', '-xla',
                                             '-num_worker', '1'])
        server = BertServer(args)
        server.start()

        bc = BertClient()
        x_train = bc.encode(list(train_df.text))
```

```
            ['-ip', 'localhost', '-port', '5555', '-timeout', '5000'])
        BertServer.shutdown(shut_args)

        np.save(trainfile, x_train)
        np.save(testfile, x_test)
    else:
        x_train = np.load(trainfile)
        x_test = np.load(testfile)

    return x_train, x_test


def focal_loss(gamma=2., alpha=.25):
    def focal_loss_fixed(y_true, y_pred):
        pt_1 = tf.where(tf.equal(y_true, 1), y_pred, tf.ones_like(y_pred))
        pt_0 = tf.where(tf.equal(y_true, 0), y_pred, tf.zeros_like(y_pred))
        return -K.mean(alpha * K.pow(1. - pt_1, gamma) * K.log(pt_1)) - K.mean((1 - alp
    return focal_loss_fixed


class MultiLabelClassifier:
    def __init__(self):
        self.model = tf.keras.Sequential()
        self.model.add(tf.keras.Input(shape=(768,)))
        self.model.add(tf.keras.layers.Dense(512, activation='relu'))
        self.model.add(tf.keras.layers.Dropout(rate=0.8))
        self.model.add(tf.keras.layers.Dense(363, activation='sigmoid'))
        self.model.summary()

    def train(self, x_train, y_train, x_test, y_test, optimizer='adam', loss=[focal_los
```

```
            metrics = [tf.keras.metrics.AUC()]
        self.model.compile(optimizer, loss, metrics)

        history = self.model.fit(
            x_train, y_train, batch_size, epochs, validation_data=(x_test, y_test))
        results = self.model.evaluate(x_test, y_test, batch_size)
        for name, value in zip(self.model.metrics_names, results):
            print("%s: %.3f" % (name, value))
        return history, results

    def predict(self, x_test):
        return self.model.predict(x_test)

    def test_sample(self, test_data, actual):
        print("actual ground truth={}, predicted={}".format(
            actual, self.model.predict(test_data)))

@ex.config
def my_config():
  max_seq_len = 256
  batch_size = 128
  gamma = 2

@ex.automain
def train_and_evaluate(max_seq_len, batch_size, gamma):
  train_df, eval_df = prepare_data()
  y_train = np.array(train_df['labels'].tolist())
  y_test = np.array(eval_df['labels'].tolist())

  x_train, x_test = encode_with_bert(train_df, eval_df, max_seq_len=max_seq_len)
```

```
    history, results = classifier.train(x_train,
                                        y_train,
                                        x_test,
                                        y_test,
                                        optimizer='adam',
                                        loss=[focal_loss(alpha=.10, gamma=gamma)],
                                        metrics='auc',
                                        batch_size=batch_size,
                                        epochs=100
                                        )

    predictions = classifier.model.predict(x_test)

    t = 0.20
    predicted = []
    predicted = (predictions >= t).astype(int)
    #print(predicted)
    f1 = f1_score(y_test,predicted,average='micro')
    print("F1 of BERT model is:", f1)

    loss, auc = results

    ex.log_scalar("test_f1", f1)
    ex.log_scalar("train_loss", loss)
    ex.log_scalar("train_auc", auc)

```

bert_multilabel_extract.py hosted with ❤️ by GitHub view raw

1. **max_seq_len** — maximum length of sequence. The best found value is **256** and it required GPU to be used
2. **batch_size** — number of samples that will be propagated through the network. Chosen number is **128**
3. **gamma** — the focusing parameter in focal loss that smoothly adjusts the rate at which easy examples are down-weighted. The focal loss is designed to address class imbalance by down-weighting inliers (easy examples) such that their contribution to the total loss is small even if their number is large. It focuses on training a sparse set of hard examples. The most optimal value of gamma in our example is **2**

```
from bert_multiclass_torch_extract import ex
import itertools

max_seq_len_values = [256]
batch_size_values = [128]
gamma_values = [2]

for max_seq_len, batch_size, gamma in itertools.product(max_seq_len_values, batch_size_va
    ex.run(config_updates={'max_seq_len': max_seq_len, 'batch_size': batch_size, 'gamma':
```

run_sacred.py hosted with ❤️ by GitHub view raw

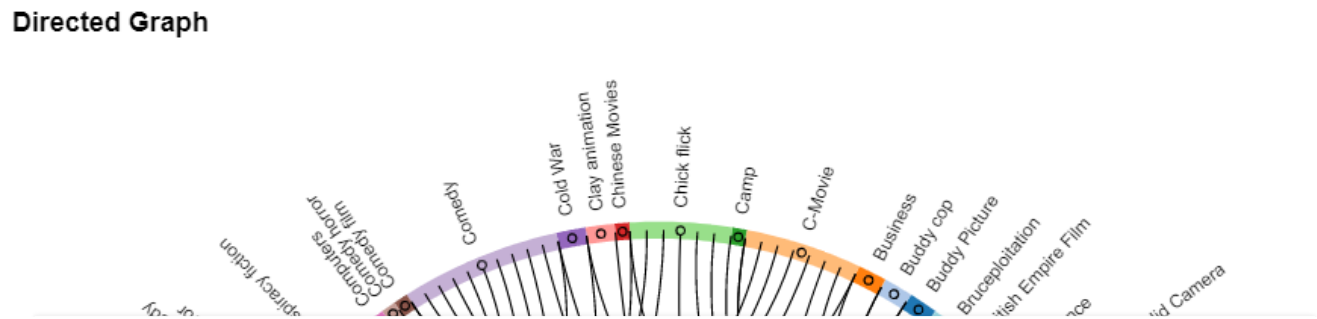
Ob

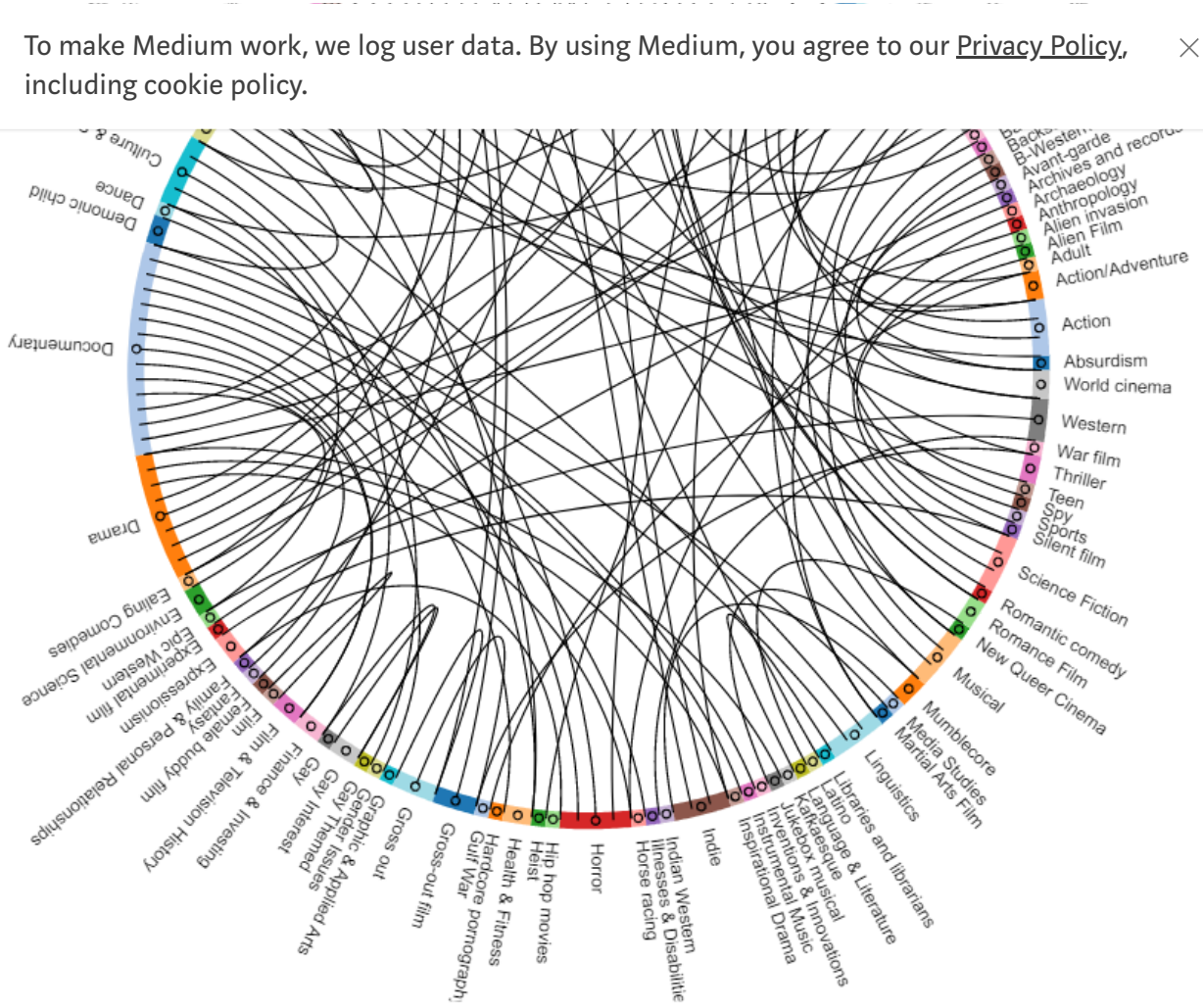
Labels co-occurrences

We need to remember that there are multiple labels in our dataset and sometimes one label indicate occurrence of the other one. In order to check this percentage dependency, I created matrix of co-occurrences. Now we know that, for example:

```
Action Comedy -> Action, 0.9722222222222222
Action Comedy -> Comedy, 0.9629629629629629
Action Thrillers -> Action, 0.9492537313432836
Action Thrillers -> Thriller, 0.9253731343283582
Adventure Comedy -> Comedy, 0.9101123595505618
```

The top 100 strongest dependencies represented on graph below.





Route chord of labels cooccurrences

With gained knowledge, we can try to modify our predictions now. After few trials I decided to change value of resulting label for 1, only if its co-occurrence with indicating label was greater than or equal 0.9.

```
    meta = pd.read_csv(os.path.join(
        base_path, "movie.metadata.tsv"), sep='\t', header=None)
    meta.columns = ["movie_id", 1, "movie_name", 3, 4, 5, 6, 7, "genre"]
    genres = meta[["movie_id", "movie_name", "genre"]]

    plots = pd.read_csv(os.path.join(
        base_path, "plot_summaries.txt"), sep='\t', header=None)
    plots.columns = ["movie_id", "plot"]

    genres['movie_id'] = genres['movie_id'].astype(str)
    plots['movie_id'] = plots['movie_id'].astype(str)
    movies = pd.merge(plots, genres, on='movie_id')

    genres_lists = []
    for i in movies['genre']:
        genres_lists.append(list(json.loads(i).values()))
    movies['genre'] = genres_lists
    multilabel_binarizer = MultiLabelBinarizer()
    multilabel_binarizer.fit_transform(movies['genre'])
    # transform target variable
    y = multilabel_binarizer.transform(movies['genre'])
    for idx, genre in enumerate(multilabel_binarizer.classes_):
        movies[genre] = y[:, idx]

    #del movies['Unnamed: 0']
    del movies['movie_name']
    del movies['movie_id']
    del movies['plot']
    del movies['genre']

```

```
    LEN = df.shape[0]
    SIZE_TRAIN = int(TRAIN_VAL_RATIO*LEN)

    train_df = df[:SIZE_TRAIN]

    return train_df


def cooccurrences(t):
    df = prepare_train_data_for_coocurrences()
    cooccurrence_matrix = np.dot(df.transpose(),df)
    cooccurrence_matrix_diagonal = np.diagonal(cooccurrence_matrix)
    with np.errstate(divide='ignore', invalid='ignore'):
        cooccurrence_matrix_percentage = np.nan_to_num(np.true_divide(cooccurrence_matri
    tuples = {}
    result = np.where(cooccurrence_matrix_percentage > t)
    listOfCoordinates= list(zip(result[0], result[1]))
    for cord in listOfCoordinates:
        if cord[0] != cord[1]:
            tuples[(df.columns.tolist()[cord[0]],
                        df.columns.tolist()[cord[1]])] = cooccurrence_matrix_percentag

movies_columns = get_columns_names()
pred_df = pd.DataFrame(predicted, columns = movies_columns)
coocs = coocurrences(t=0.9)
ind = [i[0] for i in coocs]
conc = [i[1] for i in coocs]

for i, label in enumerate(ind):
  for index, row in pred_df.iterrows():
    if row[label] == 1:
```

```
f1_updated = f1_score(y_test,pred_df.to_numpy(),average='micro')
print("Updated F1 of BERT model is:", f1_updated)
```

cooccurrences.py hosted with ❤ by GitHub view raw

Updated F1 score is **0.5**, which is small improvement.

Results

Sample results look as follows.

```
First Platoon centers around Rock Brannigan ([[Scott Gibson  and his
ragtag squad of ex-military zombie hunters trying to make a living
in the desert Southwest two years after the zombie apocalypse. Along
the way they encounter the grizzled Pa Jericho , and the eccentric
Rex Necro .
Action, Comedy, Horror, Parody, Science Fiction


The life of the S&M-theme artist and author Seiu Ito is depicted in
the film. His artistic life and Sadian philosophy, inspired by his
torturing of his two wives and Tae, his favorite prostitute, are
portrayed as shown in his journalistic writings. Tae is eventually
driven insane due to Ito's attentions.
Drama, Japanese Movies, World cinema
```

C

Usage of BERT embeddings enabled to gain 7% of F1 score improvement, (50% overall).

In future work, I think that good idea would be to reduce number of labels keep only the main ones. As 'Action Comedy' is in 100% of cases 'Action' and in 100% 'Comedy', maybe we don't really need this category.

Thank you for reading.

Deep Learning Data Science NLP Bert Machine Learning