Competition: [https://www.kaggle.com/c/nlp-getting-started/overview (https://www.kaggle.com/c/nlp-getting-started/overview)](https://www.kaggle.com/c/nlp-getting-started/overview)

Оглавление:

In [15]:

```
import pandas as pd
import numpy as np
import sklearn.metrics as metrics
import matplotlib.pyplot as plt
import re
import tensorflow as tf


from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow_hub as hub

#https://www.kaggle.com/product-feedback/91185
import bert_tokenization as tokenization
#The file "tokenization" is forked from:
#https://github.com/google-research/bert/blob/master/tokenization.py.
```

In [2]:

```
df_train = pd.read_csv("../input/nlp-getting-started/train.csv")
df_test = pd.read_csv("../input/nlp-getting-started/test.csv")
```

In [3]:

```
display(df_train.head())
print(df_train.shape, df_test.shape)
```

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

```
(7613, 5) (3263, 4)
```

In [4]:

```
df_train.drop(["keyword","location"], axis=1, inplace=True)
df_test.drop(["keyword","location"], axis=1, inplace=True)
```

Check for missing values

In [5]:

```
df_train.isnull().sum()
```

Out[5]:

```
id        0
text      0
target    0
dtype: int64
```

In [6]:

```
display(df_train.target.value_counts())
```

```
0    4342
1    3271
Name: target, dtype: int64
```

Detect & remove empty strings

In [7]:

```
blanks = []

for index, i, text, target in df_train.itertuples():  # iterate over the DataFrame
    if type(text)==str:            # avoid NaN values
        if text.isspace():         # test 'review' for whitespace
            blanks.append(i)

print(len(blanks), 'blanks: ', blanks)

df_train.drop(blanks, inplace=True)
```

```
0 blanks:  []
```

1. EDA

1. Избавимся от ссылок

In [8]:

```
line = df_train["text"].head(-5).values[-1]
print(line)

pattern = r'http://[/.\w]+'#cuz maybe https://... or http://  or just http
print(re.findall(pattern, line))

re.sub(pattern,'',line)
```

```
#stormchase Violent Record Breaking EF-5 El Reno Oklahoma Tornado Near
ly Runs Over ... - http://t.co/3SICroAaNz (http://t.co/3SICroAaNz) htt
p://t.co/I27Oa0HISp (http://t.co/I27Oa0HISp)
['http://t.co/3SICroAaNz', 'http://t.co/I27Oa0HISp']
```

Out[8]:

```
'#stormchase Violent Record Breaking EF-5 El Reno Oklahoma Tornado Nea
rly Runs Over ... -  '
```

In [9]:

```
def get_rid_of_link(text):
    raw_s = r'{}'.format(text)
    pattern = r'http[:/.\w]+'
    raw_s = re.sub(pattern,'',raw_s)
    return(raw_s)

df_train["text"] = df_train["text"].apply(get_rid_of_link)
df_test["text"] = df_test["text"].apply(get_rid_of_link)
```

2. Find time (am/pm/UTC/..)

In [10]:

```
'''
pattern = r"[\d]+:[\d]+:[\d]+"
pattern_2 = r"[\d]+:[\d]+"
pattern_3 = r"(am|pm|UTC)"

pattern = r"[\d]+:[\d]+:[\d]+|[\d]+:[\d]+|am|pm|UTC"

line = r"Earthquake : M 3.4 - 96km N of Brenas Puerto Rico: Time2015-08-05 10:34:24
print(re.findall(pattern, line))
print(re.sub(pattern, "",line))
print(line)

line = r"Meow, Sparta"
print(re.findall(pattern, line))
'''
```

Out[10]:

```
'\npattern = r"[\\d]+:[\\d]+:[\\d]+"\npattern_2 = r"[\\d]+:[\\d]+"\npa
ttern_3 = r"(am|pm|UTC)"\n\npattern = r"[\\d]+:[\\d]+:[\\d]+|[\\d]+:
[\\d]+|am|pm|UTC"\n\nline = r"Earthquake : M 3.4 - 96km N of Brenas Pu
erto Rico: Time2015-08-05 10:34:24 UTC2015-08-05 06:34:24 -4:00 at\x89
Û_"\nprint(re.findall(pattern, line))\nprint(re.sub(pattern, "",line))
\nprint(line)\n\nline = r"Meow, Sparta"\nprint(re.findall(pattern, lin
e))\n'
```

In [11]:

```
'''
def create_bool_time_feature(text):
    raw_s = r'{}'.format(text)
    pattern = r"[\d]+:[\d]+:[\d]+|[\d]+:[\d]+|am|pm|UTC"
    if(len(re.findall(pattern, raw_s))!=0):
        return(1)
    else:
        return(0)

def get_rid_of_time(text):
    raw_s = r'{}'.format(text)
    pattern = r"[\d]+:[\d]+:[\d]+|[\d]+:[\d]+|am|pm|UTC"
    raw_s = re.sub(pattern,'',raw_s)
    return(raw_s)

df_train["time"] = df_train["text"].apply(create_bool_time_feature)
df_test["time"] = df_test["text"].apply(create_bool_time_feature)

df_train["text"] = df_train["text"].apply(get_rid_of_time)
df_test["text"] = df_test["text"].apply(get_rid_of_time)

df_train["time"].value_counts()
'''
```

Out[11]:

```
'\ndef create_bool_time_feature(text):\n    raw_s = r\'{}\'.format(tex
t)\n    pattern = r"[\\d]+:[\\d]+:[\\d]+|[\\d]+:[\\d]+|am|pm|UTC"\n
if(len(re.findall(pattern, raw_s))!=0):\n        return(1)\n    els
e:\n        return(0)\n    \ndef get_rid_of_time(text):\n    raw_s = r
\'{}\'.format(text)\n    pattern = r"[\\d]+:[\\d]+:[\\d]+|[\\d]+:[\\d]
+|am|pm|UTC"\n    raw_s = re.sub(pattern,\'\',raw_s)\n    return(raw_
s)\n\ndf_train["time"] = df_train["text"].apply(create_bool_time_featu
re)\ndf_test["time"] = df_test["text"].apply(create_bool_time_feature)
\n\ndf_train["text"] = df_train["text"].apply(get_rid_of_time)\ndf_tes
t["text"] = df_test["text"].apply(get_rid_of_time)\n\ndf_train["tim
e"].value_counts()\n'
```

3. Удалим дубликаты

Есть дубликаты. Некоторые наблюдения полностью совпадают по "text", некоторые отличаются орфографической ошибкой в тексте.
Удалим те, что полностью идентичны по feature "text" (значения "taget" порой разные)

In [12]:

```
print(f"Amount of observations: {df_train.text.shape},\nNumber of unique observatio
df_train = df_train.drop_duplicates(subset=['text'])
```

```
Amount of observations: (7613,),
Number of unique observations: (6989,)
```

4. Удаление всех токенов вида цифры/цифры+слова

In [ ]:

```
def get_rid_of_digits(text):
    raw_s = r'{}'.format(text)
    pattern = r"\d+\w+|\w+\d+"
    raw_s = re.sub(pattern,'',raw_s)
    return(raw_s)

df_train["text"] = df_train["text"].apply(get_rid_of_digits)
```

5. Удаление Тэгов (# ... и @....) и слов с подчеркиванием (_ashj)

In [13]:

```
def get_rid_of_tags(text):
    raw_s = r'{}'.format(text)
    pattern = r"@\w+|#\w+|_+\w+|\w+_|"
    raw_s = re.sub(pattern,'',raw_s)
    return(raw_s)

df_train["text"] = df_train["text"].apply(get_rid_of_tags)
```

6. Отбор слов (создание списка stop_words)

In [ ]:

```
'''
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

vectorizer = TfidfVectorizer(min_df = 0, max_df = 5000, stop_words=ENGLISH_STOP_WOR

X_train_counts = vectorizer.fit(df_train["text"])
word_freq = X_train_counts.vocabulary_

word_freq = dict(sorted(word_freq.items(), key=lambda x: x[1], reverse=False))
word_freq
'''
```

---

2. BERT

In [14]:

```
def bert_encode(texts, tokenizer, max_len=512):
    all_tokens = []
    all_masks = []
    all_segments = []

    for text in texts:
        text = tokenizer.tokenize(text)

        text = text[:max_len-2]
        input_sequence = ["[CLS]"] + text + ["[SEP]"]
        pad_len = max_len - len(input_sequence)

        tokens = tokenizer.convert_tokens_to_ids(input_sequence)
        tokens += [0] * pad_len
        pad_masks = [1] * len(input_sequence) + [0] * pad_len
        segment_ids = [0] * max_len

        all_tokens.append(tokens)
        all_masks.append(pad_masks)
        all_segments.append(segment_ids)

    return np.array(all_tokens), np.array(all_masks), np.array(all_segments)
```

In [17]:

```
def build_model(bert_layer, max_len=512):
    input_word_ids = Input(shape=(max_len,), dtype=tf.int32, name="input_word_ids")
    input_mask = Input(shape=(max_len,), dtype=tf.int32, name="input_mask")
    segment_ids = Input(shape=(max_len,), dtype=tf.int32, name="segment_ids")

    _, sequence_output = bert_layer([input_word_ids, input_mask, segment_ids])
    print(sequence_output)
    clf_output = sequence_output[:, 0, :]
    print(clf_output.shape)
    out = Dense(1, activation='sigmoid')(clf_output)

    model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=out)
    model.compile(SGD(lr=0.0001, momentum=0.8), loss='binary_crossentropy', metrics

    return model
```

In [18]:

```
%%time
module_url = "https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1"
bert_layer = hub.KerasLayer(module_url, trainable=True)# Choice of the BERT model i
```

```
CPU times: user 12.2 s, sys: 3.09 s, total: 15.3 s
Wall time: 17.7 s
```

In [19]:

```
train = df_train
max_len = 128

vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()
tokenizer = tokenization.FullTokenizer(vocab_file, do_lower_case)

train_input = bert_encode(train.text.values, tokenizer, max_len=max_len)
train_labels = train.target.values

print(len(train_input), train_input[0].shape)
```

```
3 (6989, 128)
```

In [20]:

```python
model = build_model(bert_layer, max_len=max_len)
model.summary()
```

```
Tensor("keras_layer/cond/Identity_1:0", shape=(None, None, 768), dtype
=float32)
(None, 768)
Model: "functional_1"
______________________________________________________________________
____________________________
Layer (type)                    Output Shape         Param #     Conne
cted to
======================================================================
============================
input_word_ids (InputLayer)     [(None, 128)]        0
______________________________________________________________________
____________________________
input_mask (InputLayer)         [(None, 128)]        0
______________________________________________________________________
____________________________
segment_ids (InputLayer)        [(None, 128)]        0
______________________________________________________________________
____________________________
keras_layer (KerasLayer)        [(None, 768), (None, 109482241   input
_word_ids[0][0]
                                                                 input
_mask[0][0]
                                                                 segme
nt_ids[0][0]
______________________________________________________________________
____________________________
tf_op_layer_strided_slice (Tens [(None, 768)]        0           keras
_layer[0][1]
______________________________________________________________________
____________________________
dense (Dense)                   (None, 1)            769         tf_op
_layer_strided_slice[0][0]
======================================================================
============================
Total params: 109,483,010
Trainable params: 109,483,009
Non-trainable params: 1
______________________________________________________________________
____________________________
```

In [21]:

```
checkpoint = ModelCheckpoint('model.h5', monitor='val_accuracy', save_best_only=Tru

train_history = model.fit(
    train_input, train_labels,
    validation_split=0.2,
    epochs=10,
    callbacks=[checkpoint],
    batch_size=32,
    verbose=1
)
```

```
Epoch 1/10
175/175 [==============================] - 86s 494ms/step - loss: 0.63
07 - accuracy: 0.6493 - val_loss: 0.5681 - val_accuracy: 0.7210
Epoch 2/10
175/175 [==============================] - 86s 490ms/step - loss: 0.52
14 - accuracy: 0.7501 - val_loss: 0.4890 - val_accuracy: 0.7747
Epoch 3/10
175/175 [==============================] - 86s 490ms/step - loss: 0.47
35 - accuracy: 0.7850 - val_loss: 0.4619 - val_accuracy: 0.7868
Epoch 4/10
175/175 [==============================] - 86s 490ms/step - loss: 0.44
66 - accuracy: 0.8036 - val_loss: 0.4433 - val_accuracy: 0.8047
Epoch 5/10
175/175 [==============================] - 85s 488ms/step - loss: 0.42
87 - accuracy: 0.8135 - val_loss: 0.4309 - val_accuracy: 0.8062
Epoch 6/10
175/175 [==============================] - 85s 489ms/step - loss: 0.41
49 - accuracy: 0.8226 - val_loss: 0.4241 - val_accuracy: 0.8097
Epoch 7/10
175/175 [==============================] - 86s 490ms/step - loss: 0.40
28 - accuracy: 0.8287 - val_loss: 0.4200 - val_accuracy: 0.8162
Epoch 8/10
175/175 [==============================] - 85s 489ms/step - loss: 0.39
14 - accuracy: 0.8342 - val_loss: 0.4227 - val_accuracy: 0.8219
Epoch 9/10
175/175 [==============================] - 83s 475ms/step - loss: 0.38
35 - accuracy: 0.8378 - val_loss: 0.4118 - val_accuracy: 0.8205
Epoch 10/10
175/175 [==============================] - 83s 475ms/step - loss: 0.37
39 - accuracy: 0.8426 - val_loss: 0.4100 - val_accuracy: 0.8212
```

Predict test dataset to submit

In [33]:

```
test_input = bert_encode(df_test.text.values, tokenizer, max_len=max_len)
test_pred = model.predict(test_input)
submission = train.truncate(after = -1)
submission['id'] = df_test['id']
submission['text'] = df_test['text']
submission['target'] = test_pred.round().astype(int)
```

In [34]:

```
submission = submission[['id','target']]
```

In [35]:

```
submission.to_csv("./answer.csv", index=False)
```

Continue training

In [28]:

```
#load_model = tf.keras.models.load_model('my_model.h5')

# retraining the model
model.fit(train_input, train_labels,
    validation_split=0.2,
    epochs=10,
    callbacks=[checkpoint],
    batch_size=32,
    verbose=1)
```

```
Epoch 1/10
175/175 [==============================] - 85s 487ms/step - loss: 0.36
47 - accuracy: 0.8467 - val_loss: 0.4092 - val_accuracy: 0.8233
Epoch 2/10
175/175 [==============================] - 83s 474ms/step - loss: 0.35
60 - accuracy: 0.8517 - val_loss: 0.4076 - val_accuracy: 0.8205
Epoch 3/10
175/175 [==============================] - 83s 474ms/step - loss: 0.34
79 - accuracy: 0.8567 - val_loss: 0.4080 - val_accuracy: 0.8212
Epoch 4/10
175/175 [==============================] - 83s 474ms/step - loss: 0.33
88 - accuracy: 0.8634 - val_loss: 0.4092 - val_accuracy: 0.8197
Epoch 5/10
175/175 [==============================] - 83s 474ms/step - loss: 0.33
08 - accuracy: 0.8655 - val_loss: 0.4109 - val_accuracy: 0.8190
Epoch 6/10
175/175 [==============================] - 83s 474ms/step - loss: 0.32
14 - accuracy: 0.8741 - val_loss: 0.4102 - val_accuracy: 0.8233
Epoch 7/10
175/175 [==============================] - 83s 475ms/step - loss: 0.31
20 - accuracy: 0.8771 - val_loss: 0.4148 - val_accuracy: 0.8226
Epoch 8/10
175/175 [==============================] - 85s 488ms/step - loss: 0.30
34 - accuracy: 0.8814 - val_loss: 0.4154 - val_accuracy: 0.8240
Epoch 9/10
175/175 [==============================] - 83s 474ms/step - loss: 0.29
44 - accuracy: 0.8864 - val_loss: 0.4150 - val_accuracy: 0.8219
Epoch 10/10
175/175 [==============================] - 83s 474ms/step - loss: 0.28
43 - accuracy: 0.8914 - val_loss: 0.4232 - val_accuracy: 0.8226
```

Out[28]:

```
<tensorflow.python.keras.callbacks.History at 0x7ff378fd5c50>
```

Выводы и score в kaggle:

- Метрику f1-score и confusion matrix не использовал, поскольку имеется лишь небольшой дисбаланс между классами у целевой переменной.

- При тестировании классических и бустинг алгоритмов ML использовалась дополнительная предобработка. Однако это позволило лишь дойти до топ-45% leaderboard.
- Использовались вычислительные мощности kaggle'a - NVidia K80 GPUs.
- С таким notebook попал в топ-17% (200/1245) leaderboard.
  [https://www.kaggle.com/c/nlp-getting-started/leaderboard (https://www.kaggle.com/c/nlp-getting-started/leaderboard)](https://www.kaggle.com/c/nlp-getting-started/leaderboard)
  [https://www.kaggle.com/konstantinlp (https://www.kaggle.com/konstantinlp)](https://www.kaggle.com/konstantinlp)