Disaster Tweet Prediction

Ввод [ ]:

```
# Predict whether a given tweet is about a real disaster or not. If so, predict a 1. If not
```

Ввод [ ]:

```
# The structure of this project is following.
#
# 1. Primary processing of the text data (text cleaning from stop-words and punctuation, le
# 2. Text transformation by three ways
#   : Bag-of-Words (BOW)
#   : TF-IDF
#   : Word Embedding
# 3. Building of classification models
# 3.1 Definition of functions for classification (hyperparameters selection by grid, visual
# 3.2 Logistic Regression model realized for all three text representations
#     3.2.1 BOW
#     3.2.2 TF-IDF
#     3.2.3 Word Embedding
# 3.3 SVC model for word embedding text representation
# 3.4 DecisionTreeClassifier model for word embedding text representation
# 3.5 SGDClassifier model for word embedding text representation
# 3.6 RandomForestClassifier model for word embedding text representation
# 3.7 XGBClassifier model for word embedding text representation
#
# Word Embedding representation works the best way. The following two models show the best
# Logistic Regression and gradient boosting XGB Classifier.
#
# So, the prediction of the KAGGLE data is presented in 3.2.3 with a resulting accuracy ~79
# This value is not a big but the result looks adequate considering heterogenious and often
# This prediction was also uploaded on KAGGLE, which is shown on the snapshot in 3.2.3.
```

1. Primary processing of the text data

Ввод [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns

import time
import warnings

from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
import string
from nltk.corpus import wordnet
#nltk.download('wordnet')
from nltk import pos_tag
#nltk.download('averaged_perceptron_tagger')
import spacy
#!python -m spacy download en_core_web_lg

from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,f1_score
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import ElasticNet
from sklearn.ensemble import RandomForestClassifier

import xgboost as xgb

from IPython.display import Image
```

Ввод [2]:

```
data = pd.read_csv('Data/DisasterTweets/train.csv', sep=',', skipinitialspace=True)
```

Ввод [247]:

```
data.head()
```

Out[247]:

	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

Ввод [21]:

```
data.shape
```

Out[21]:

```
(7613, 5)
```

Ввод [18]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        7613 non-null   int64
 1   keyword   7552 non-null   object
 2   location  5079 non-null   object
 3   text      7613 non-null   object
 4   target    7613 non-null   int64
dtypes: int64(2), object(3)
memory usage: 297.5+ KB
```

Ввод [20]:

```
# just look on some data raw
list(data.iloc[567])
```

Out[20]:

```
[819,
 'battle',
 'West Richland, WA',
 '@DetroitPls interested to see who will win this battle',
 0]
```

Ввод [3]:

```
# It is reasonable to keep the fields 'text' and 'target' for further analysis
X = data.text
y = data.target
```

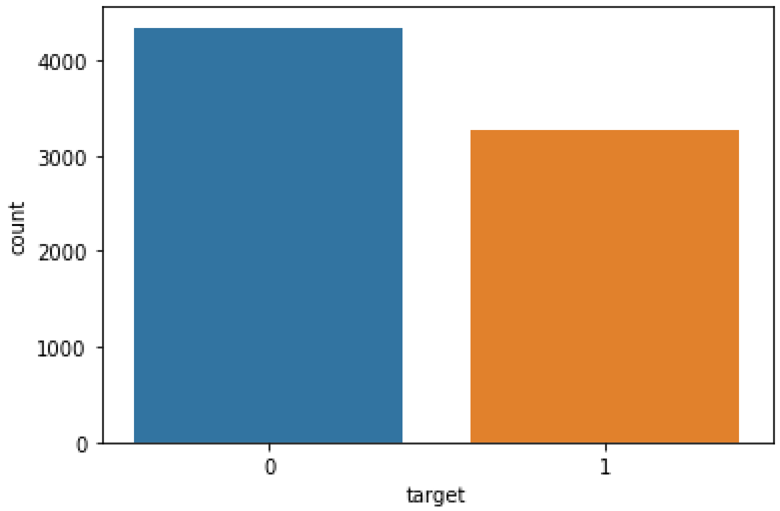
Ввод [24]:

```
# Look at distribution of the target feature 'target'
warnings.filterwarnings("ignore")
sns.countplot(y)
```

```
<IPython.core.display.Javascript object>
```

Out[24]:

```
<AxesSubplot:xlabel='target', ylabel='count'>
```



Ввод [ ]:

```
# The data is balanced, no need to apply balancing techniques
```

Ввод [4]:

```
# create the list of stop-words and punctuation
stop = set(STOPWORDS)
punctuation = list(string.punctuation)
stop.update(punctuation)
```

Ввод [5]:

```
def get_simple_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN
```

Ввод [6]:

```
# function of smart lemmatizing of the text
lemmatizer = WordNetLemmatizer()
def lemmatize_words(text):
    final_text = []
    for i in text.split():
        if i.strip().lower() not in stop:
            # define syntactic and semantic information, bank of linguistic trees
            # decryption of all possible values here: www.ling.upenn.edu/courses/Fall_2003/
            pos = pos_tag([i.strip()])
            # the 2-nd parameter of lemmatizer - POS tag, it helps to improve lemmatizing q
            word = lemmatizer.lemmatize(i.strip(),get_simple_pos(pos[0][1]))
            final_text.append(word.lower())
    return " ".join(final_text)
```

Ввод [50]:

```
# example how 'pos_tag()' function works
pos_tag([X.iloc[0].split()[1].strip()])
```

Out[50]:

```
[('Deeds', 'NNS')]
```

Ввод [7]:

```
# apply lemmatizing to the whole data set
X = X.apply(lemmatize_words)
```

2. Text transformation

2.1 BOW and TF-IDF transformation

Ввод [8]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2 , random_state =
```

Ввод [9]:

```
# Case №1. Bag-of-words

# Initialize the module CountVectorizer
CV = CountVectorizer()
# Create a dictionary from X_train
CV.fit(X_train)
# The size of the created dictionary
print("The dictionary has",len(CV.vocabulary_),"words")
# переводим текстовые данные в числовые
X_train_cv = CV.transform(X_train)
X_test_cv = CV.transform(X_test)

print('The size of the training dataset:',X_train_cv.shape)
print('The size of the testing dataset:',X_test_cv.shape)
```

```
The dictionary has 17747 words
The size of the training dataset: (6090, 17747)
The size of the testing dataset: (1523, 17747)
```

Ввод [74]:

```
np.unique(X_train_cv.toarray())
```

Out[74]:

```
array([ 0,  1,  2,  3,  4,  6,  7,  9, 13], dtype=int64)
```

Ввод [10]:

```
# Case №2. TF-IDF

# Initialize the module TfidfVectorizer
TF = TfidfVectorizer()
# Create the dictionary from X_train
TF.fit(X_train)
# The size of the created dictionary
print("The dictionary has",len(TF.vocabulary_),"words")
# переводим текстовые данные в числовые
X_train_tf = TF.transform(X_train)
X_test_tf = TF.transform(X_test)

print('The size of the training dataset:',X_train_tf.shape)
print('The size of the testing dataset:',X_test_tf.shape)
```

```
The dictionary has 17747 words
The size of the training dataset: (6090, 17747)
The size of the testing dataset: (1523, 17747)
```

2.2 Word Embedding

Ввод [185]:

```
# Load the large model (lg) to get the vectors
nlp = spacy.load('en_core_web_lg')
```

Ввод [186]:

```
%%time
# combine all the word vectors into a single document vector
# by AVERAGING the vectors for each word in the document.
# So, the average document vector:
with nlp.disable_pipes():
    vectors = np.array([nlp(text).vector for text in X])
```

Ввод [189]:

```
# embedding size (here 300) - this is the weights in hidden layer of neural network
# which are created in calculation of probabilities the word belongs to dictionary words
# if there is more than one word in the documents - the average weight values are taken
vectors.shape
```

Out[189]:

```
(7613, 300)
```

Ввод [201]:

```
X_train_emb, X_test_emb, y_train_emb, y_test_emb = train_test_split(vectors, y, test_size =
```

3. Models building

3.1 Functions definition

Ввод [205]:

```
# FUNCTION OF HYPERPARAMETERS SELECTION

# model - the model which will be tuned
# tuned_parameters - grid of hyperparameters to select
def best_model(model,tuned_parameters,X_train,y_train,X_test,y_test,score = "f1",cv = 0):
    print("# Tuning hyper-parameters for %s" % score)
    print()
    if cv > 0:
        clf = GridSearchCV(model, tuned_parameters,n_jobs=-1, scoring='%s' % score, cv=cv)
    else:
        clf = GridSearchCV(model, tuned_parameters,n_jobs=-1, scoring='%s' % score)
    clf.fit(X_train, y_train)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
                  % (mean, std * 2, params))
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, clf.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()
    return clf.best_params_
```

Ввод [174]:

```
# FUNC OF CLASSIFICATOR VISUALIZATION

# Visualize the result of classification - the words inducing the maximal contribution
def visualize_coefficients(classifier, feature_names,coef = None, n_top_features=30):
    # get coefficients with large absolute values
    # here the trained classificator is the input to the function
    if coef is None:
        # ravel() removes excessive dimensions of np-array
        coef = classifier.coef_.ravel()
    # list of indexes of the sorted coefficients
    positive_coefficients = np.argsort(coef)[-n_top_features:]
    negative_coefficients = np.argsort(coef)[:n_top_features]
    # Combine 2 indexes lists above in one
    interesting_coefficients = np.hstack([negative_coefficients, positive_coefficients])
    # plot them
    plt.figure(figsize=(15, 5))
    colors = ["red" if c < 0 else "blue" for c in coef[interesting_coefficients]]
    plt.bar(np.arange(2 * n_top_features), coef[interesting_coefficients], color=colors)
    feature_names = np.array(feature_names)
    plt.xticks(np.arange(1, 1 + 2 * n_top_features), feature_names[interesting_coefficients
```

Ввод [181]:

```
# FUNCTION of PREDICTION

def fitModel(model,x_train=X_train,y_train=y_train,x_test=X_test,y_test=y_test):
    start_time = time.time()
    lr = model
    lr.fit(x_train, y_train)
    fit_time = round(time.time() - start_time,0)
    print("---time_fit model %s seconds ---" % (fit_time))
    preds = lr.predict(x_test)
    accuracy = accuracy_score(y_test, preds)
    f1 = f1_score(y_test, preds)
    report = classification_report(y_test,preds,target_names = ['0','1'])

    cm_1 = confusion_matrix(y_test,preds)
    cm_1 = pd.DataFrame(cm_1, index=[0,1], columns=[0,1])
    cm_1.index.name = 'Actual'
    cm_1.columns.name = 'Predicted'
    plt.figure(figsize = (10,10))
    sns.heatmap(cm_1,cmap= "Blues",annot = True, fmt='')
    return {'model':lr,'f1_score':f1,'accuracy_score':accuracy,'report':report,'time_fit':f
```

3.2 LogisticRegression model

3.2.1 The results for Bag-of-words transformation

Ввод [176]:

```
%%time
parameters = [{'penalty':['l2'], "solver":['newton-cg', 'lbfgs', 'liblinear'],
                'C': [0.1, 1, 10, 100, 1000]},
              {'penalty':['none'], "solver":['newton-cg', 'lbfgs', 'liblinear'],
                'C': [0.1, 1, 10, 100, 1000]}]
best_params_LR = best_model(LogisticRegression(),parameters,X_train=X_train_cv,
                            y_train=y_train,y_test=y_test,X_test=X_test_cv,score = "accurac
```

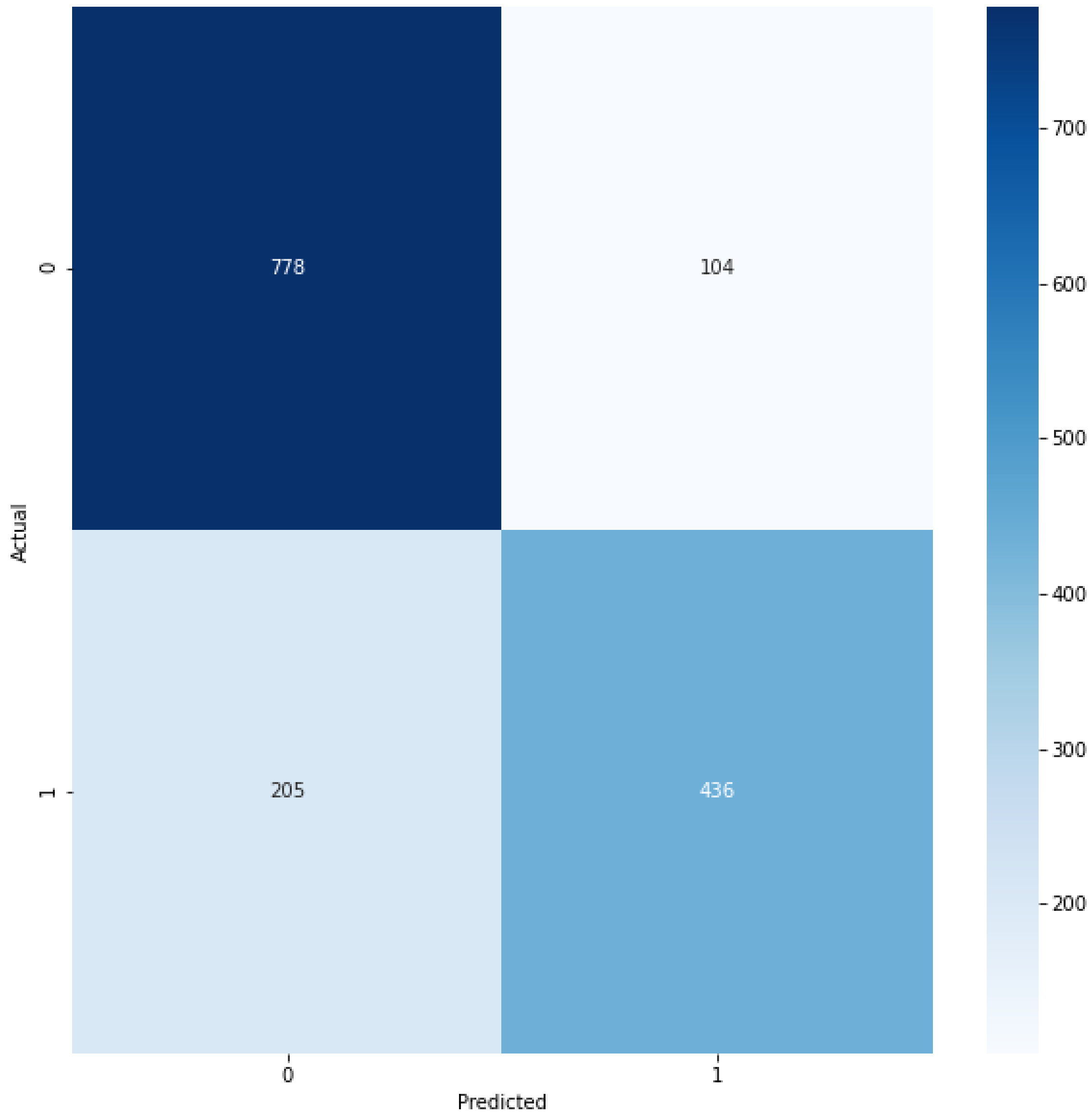
```
# Tuning hyper-parameters for accuracy

Best parameters set found on development set:

{'C': 1, 'penalty': 'l2', 'solver': 'newton-cg'}

Grid scores on development set:

0.791 (+/-0.013) for {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.791 (+/-0.013) for {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.791 (+/-0.013) for {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.798 (+/-0.005) for {'C': 1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.798 (+/-0.005) for {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.797 (+/-0.007) for {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
0.789 (+/-0.016) for {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.789 (+/-0.017) for {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
0.788 (+/-0.016) for {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.782 (+/-0.016) for {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.781 (+/-0.017) for {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.782 (+/-0.016) for {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.779 (+/-0.016) for {'C': 1000, 'penalty': 'l2', 'solver': 'newton-cg'}
0.776 (+/-0.007) for {'C': 1000, 'penalty': 'l2', 'solver': 'lbfgs'}
0.779 (+/-0.015) for {'C': 1000, 'penalty': 'l2', 'solver': 'liblinear'}
0.772 (+/-0.010) for {'C': 0.1, 'penalty': 'none', 'solver': 'newton-cg'}
0.768 (+/-0.026) for {'C': 0.1, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 0.1, 'penalty': 'none', 'solver': 'liblinear'}
0.772 (+/-0.010) for {'C': 1, 'penalty': 'none', 'solver': 'newton-cg'}
0.768 (+/-0.026) for {'C': 1, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 1, 'penalty': 'none', 'solver': 'liblinear'}
0.772 (+/-0.010) for {'C': 10, 'penalty': 'none', 'solver': 'newton-cg'}
0.768 (+/-0.026) for {'C': 10, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 10, 'penalty': 'none', 'solver': 'liblinear'}
0.772 (+/-0.010) for {'C': 100, 'penalty': 'none', 'solver': 'newton-cg'}
0.768 (+/-0.026) for {'C': 100, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 100, 'penalty': 'none', 'solver': 'liblinear'}
0.772 (+/-0.010) for {'C': 1000, 'penalty': 'none', 'solver': 'newton-cg'}
0.768 (+/-0.026) for {'C': 1000, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 1000, 'penalty': 'none', 'solver': 'liblinear'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       0.79      0.88      0.83       882
           1       0.81      0.68      0.74       641

    accuracy                           0.80      1523
   macro avg       0.80      0.78      0.79      1523
```

```
weighted avg       0.80      0.80      0.79      1523


Wall time: 53.9 s
```

Ввод [182]:

```
# Take the optimal selected hyperparameters above and run the function of prediction
bestmod = fitModel(LogisticRegression(C=best_params_LR['C'],solver = best_params_LR['solver
                   x_train=X_train_cv,y_train=y_train,x_test=X_test_cv,y_test=y_test)
# bestmod - trained model
# visualize_coefficients(bestmod['model'], CV.get_feature_names())
best_params_of_all_models = []
best_params_of_all_models.append(bestmod)
```

```
---time_fit model 0.0 seconds ---

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```



Ввод [183]:

```
best_params_of_all_models[0]['accuracy_score']
```

Out[183]:

```
0.7971109652002626
```

3.2.2 The results for TF-IDF transformation

Ввод [116]:

```
%%time
parameters = [{'penalty':['l2'], "solver":['newton-cg', 'lbfgs', 'liblinear'],
                'C': [0.1, 1, 10, 100, 1000]},
              {'penalty':['none'], "solver":['newton-cg', 'lbfgs', 'liblinear'],
                'C': [0.1, 1, 10, 100, 1000]}]
best_params_LR = best_model(LogisticRegression(),parameters,X_train=X_train_tf,
                            y_train=y_train,y_test=y_test,X_test=X_test_tf,score = "accurac
```

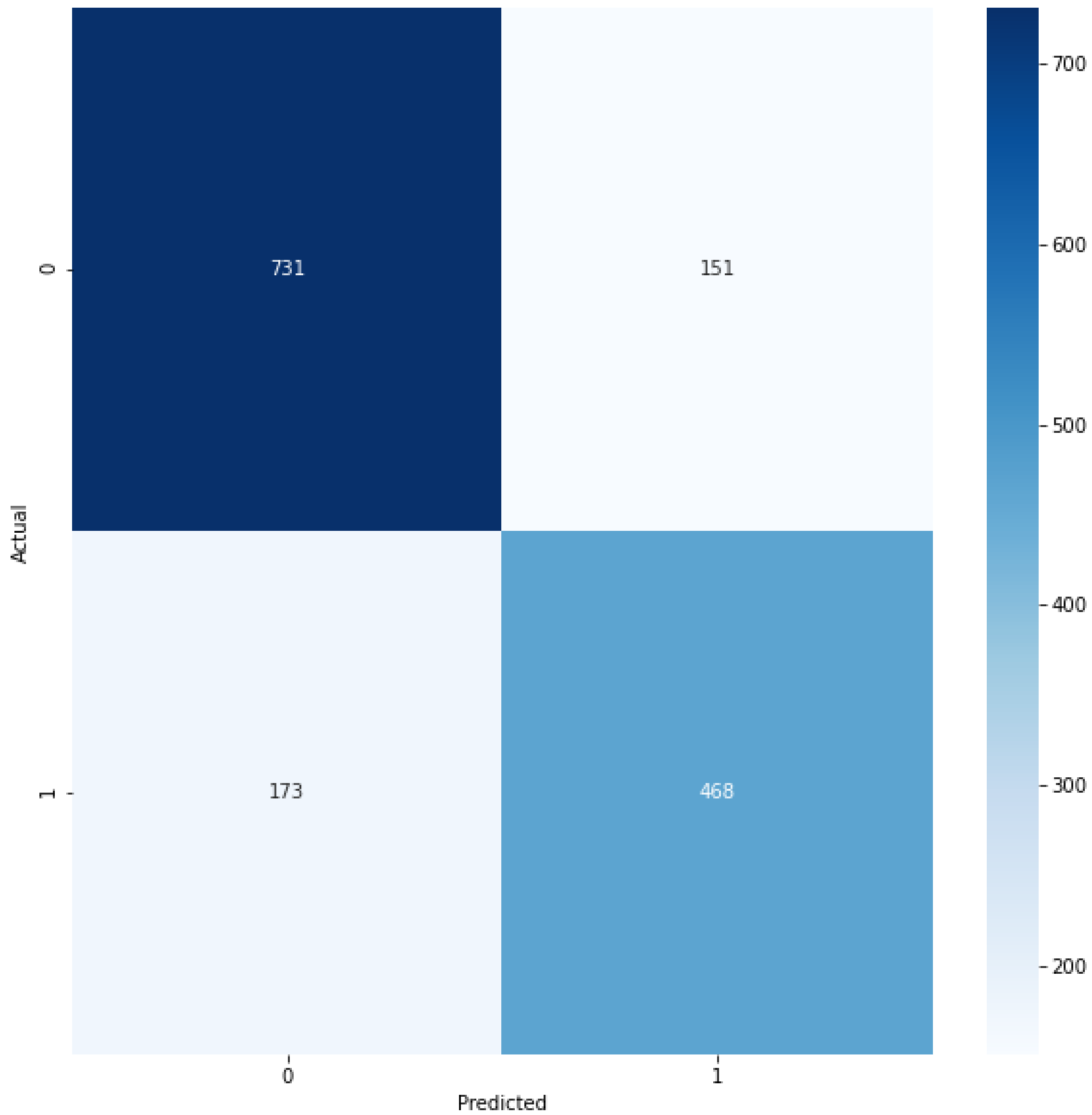
```
# Tuning hyper-parameters for accuracy

Best parameters set found on development set:

{'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}

Grid scores on development set:

0.700 (+/-0.009) for {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.700 (+/-0.009) for {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.701 (+/-0.010) for {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.790 (+/-0.010) for {'C': 1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.790 (+/-0.010) for {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.790 (+/-0.010) for {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
0.795 (+/-0.016) for {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.795 (+/-0.015) for {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
0.795 (+/-0.015) for {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.784 (+/-0.013) for {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.784 (+/-0.012) for {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.784 (+/-0.013) for {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.780 (+/-0.013) for {'C': 1000, 'penalty': 'l2', 'solver': 'newton-cg'}
0.779 (+/-0.017) for {'C': 1000, 'penalty': 'l2', 'solver': 'lbfgs'}
0.780 (+/-0.013) for {'C': 1000, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.009) for {'C': 0.1, 'penalty': 'none', 'solver': 'newton-cg'}
0.774 (+/-0.022) for {'C': 0.1, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 0.1, 'penalty': 'none', 'solver': 'liblinear'}
0.773 (+/-0.009) for {'C': 1, 'penalty': 'none', 'solver': 'newton-cg'}
0.774 (+/-0.022) for {'C': 1, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 1, 'penalty': 'none', 'solver': 'liblinear'}
0.773 (+/-0.009) for {'C': 10, 'penalty': 'none', 'solver': 'newton-cg'}
0.774 (+/-0.022) for {'C': 10, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 10, 'penalty': 'none', 'solver': 'liblinear'}
0.773 (+/-0.009) for {'C': 100, 'penalty': 'none', 'solver': 'newton-cg'}
0.774 (+/-0.022) for {'C': 100, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 100, 'penalty': 'none', 'solver': 'liblinear'}
0.773 (+/-0.009) for {'C': 1000, 'penalty': 'none', 'solver': 'newton-cg'}
0.774 (+/-0.022) for {'C': 1000, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 1000, 'penalty': 'none', 'solver': 'liblinear'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       0.81      0.83      0.82       882
           1       0.76      0.73      0.74       641

    accuracy                           0.79      1523
   macro avg       0.78      0.78      0.78      1523
```

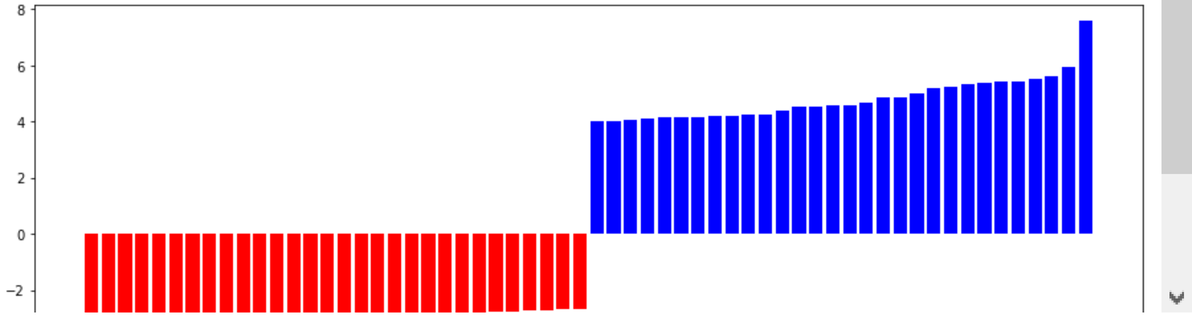
```
weighted avg       0.79      0.79      0.79      1523

Wall time: 23.3 s
```

Ввод [117]:

```
bestmod = fitModel(LogisticRegression(C=best_params_LR['C'],solver = best_params_LR['solver
                   x_train=X_train_tf,y_train=y_train,x_test=X_test_tf,y_test=y_test)

visualize_coefficients(bestmod['model'], CV.get_feature_names())
best_params_of_all_models_TF = []
best_params_of_all_models_TF.append(bestmod)
```

```
---time_fit model 0.23715591430664062 seconds ---

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```





Ввод [118]:

```
best_params_of_all_models_TF[0]['accuracy_score']
```

Out[118]:

```
0.7872619829284307
```

3.2.3 The results of Word Embedding transformation

Ввод [254]:

```
%%time
parameters = [{'penalty':['l2'], "solver":['newton-cg', 'lbfgs', 'liblinear'],
                'C': [0.1, 1, 10, 100, 1000]},
              {'penalty':['none'], "solver":['newton-cg', 'lbfgs', 'liblinear'],
                'C': [0.1, 1, 10, 100, 1000]}]
best_params_LR = best_model(LogisticRegression(),parameters,X_train=X_train_emb,
                            y_train=y_train_emb,y_test=y_test_emb,X_test=X_test_emb,score =
```

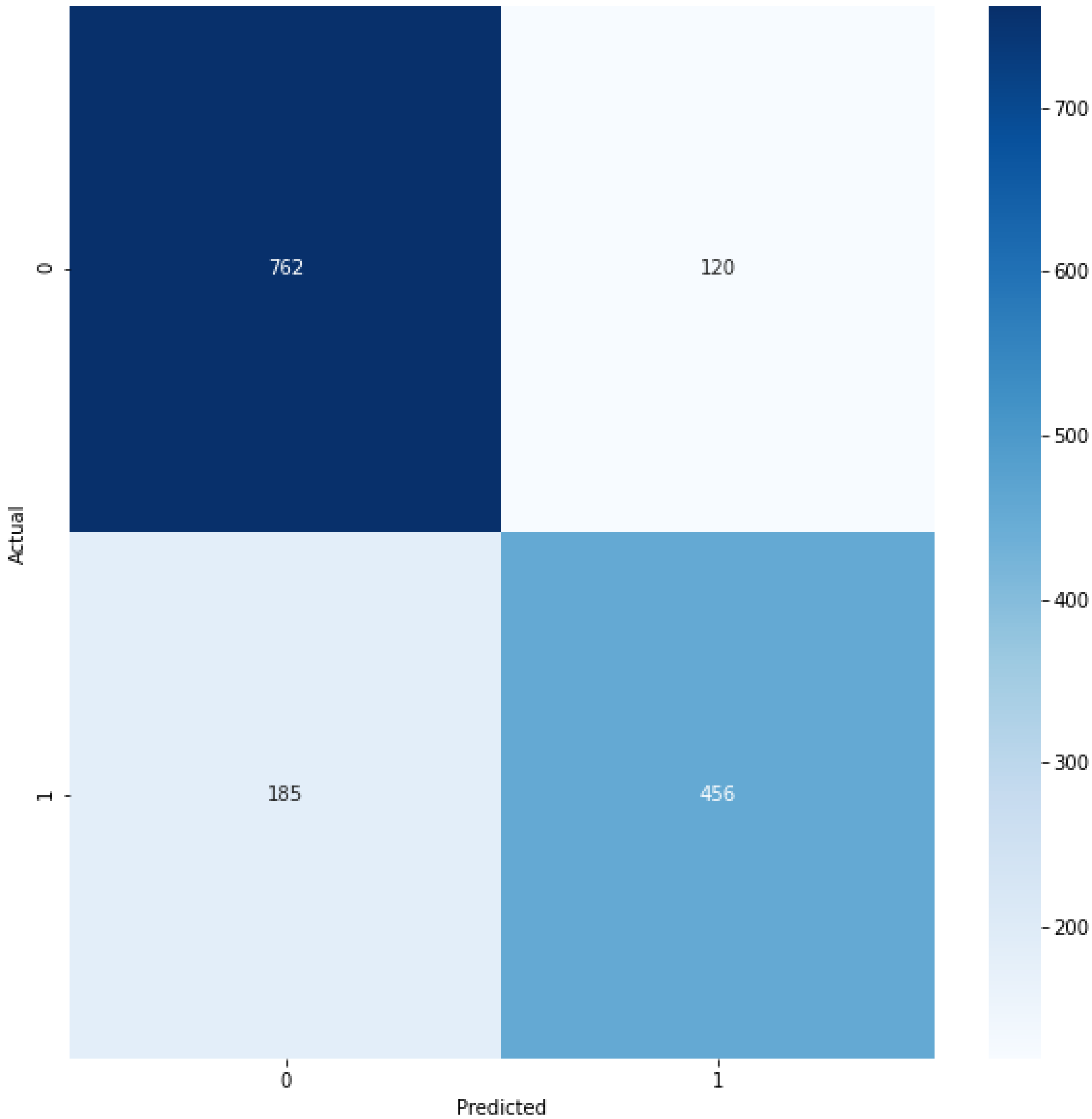
```
# Tuning hyper-parameters for accuracy

Best parameters set found on development set:

{'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}

Grid scores on development set:

0.788 (+/-0.010) for {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.788 (+/-0.010) for {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.787 (+/-0.011) for {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.788 (+/-0.011) for {'C': 1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.788 (+/-0.011) for {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.788 (+/-0.011) for {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
0.783 (+/-0.014) for {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.783 (+/-0.012) for {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
0.783 (+/-0.013) for {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.781 (+/-0.011) for {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.782 (+/-0.012) for {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.781 (+/-0.011) for {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.781 (+/-0.012) for {'C': 1000, 'penalty': 'l2', 'solver': 'newton-cg'}
0.781 (+/-0.013) for {'C': 1000, 'penalty': 'l2', 'solver': 'lbfgs'}
0.781 (+/-0.012) for {'C': 1000, 'penalty': 'l2', 'solver': 'liblinear'}
0.781 (+/-0.012) for {'C': 0.1, 'penalty': 'none', 'solver': 'newton-cg'}
0.781 (+/-0.012) for {'C': 0.1, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 0.1, 'penalty': 'none', 'solver': 'liblinear'}
0.781 (+/-0.012) for {'C': 1, 'penalty': 'none', 'solver': 'newton-cg'}
0.781 (+/-0.012) for {'C': 1, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 1, 'penalty': 'none', 'solver': 'liblinear'}
0.781 (+/-0.012) for {'C': 10, 'penalty': 'none', 'solver': 'newton-cg'}
0.781 (+/-0.012) for {'C': 10, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 10, 'penalty': 'none', 'solver': 'liblinear'}
0.781 (+/-0.012) for {'C': 100, 'penalty': 'none', 'solver': 'newton-cg'}
0.781 (+/-0.012) for {'C': 100, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 100, 'penalty': 'none', 'solver': 'liblinear'}
0.781 (+/-0.012) for {'C': 1000, 'penalty': 'none', 'solver': 'newton-cg'}
0.781 (+/-0.012) for {'C': 1000, 'penalty': 'none', 'solver': 'lbfgs'}
nan (+/-nan) for {'C': 1000, 'penalty': 'none', 'solver': 'liblinear'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       0.80      0.86      0.83       882
           1       0.79      0.71      0.75       641

    accuracy                           0.80      1523
   macro avg       0.80      0.79      0.79      1523
```

```
weighted avg       0.80      0.80      0.80      1523


Wall time: 1min 13s
```

Ввод [255]:

```
bestmod = fitModel(LogisticRegression(C=best_params_LR['C'],solver = best_params_LR['solver
                   x_train=X_train_emb,y_train=y_train_emb,x_test=X_test_emb,y_test=y_test_
```

```
---time_fit model 1.0 seconds ---

<IPython.core.display.Javascript object>
```



Ввод [211]:

```
best_params_of_all_models = []
best_params_of_all_models.append(bestmod)
```

Make a prediction on KAGGLE dataset

Ввод [257]:

```
df = pd.read_csv('Data/DisasterTweets/test.csv', sep=',', skipinitialspace=True)
```

Ввод [259]:

```
df.head(2)
```

Out[259]:

	id	keyword	location	text
0	0	NaN	NaN	Just happened a terrible car crash
1	2	NaN	NaN	Heard about #earthquake is different cities, s...

Ввод [258]:

```
df_sample = pd.read_csv('Data/DisasterTweets/sample_submission.csv', sep=',', skipinitialsp
df_sample.head(2)
```

Out[258]:

	id	target
0	0	0
1	2	0

Ввод [261]:

```
%%time
X_for_pred = df.text
# apply lematizing
X_for_pred = X_for_pred.apply(lemmatize_words)
```

```
Wall time: 37.2 s
```

Ввод [262]:

```
X_for_pred.head()
```

Out[262]:

```
0                            happen terrible car crash
1     heard #earthquake different cities, stay safe ...
2     forest fire spot pond, geese flee across stree...
3                 apocalypse lighting. #spokane #wildfires
4                    typhoon soudelor kill 28 china taiwan
Name: text, dtype: object
```

Ввод [271]:

```
%%time
# combine all the word vectors into a single document vector
# by AVERAGING the vectors for each word in the document.
# So, the average document vector:
with nlp.disable_pipes():
    X_vec = np.array([nlp(text).vector for text in X_for_pred])
```

```
Wall time: 59.1 s
```

Ввод [272]:

```
preds = bestmod['model'].predict(X_vec)
```

Ввод [287]:

```
pred_final = pd.DataFrame(np.array([df['id'].values, preds]).transpose(),columns=[['id','ta
```

Ввод [288]:

```
pred_final.head()
```

Out[288]:

	id	target
0	0	1
1	2	1
2	3	1
3	9	1
4	11	1

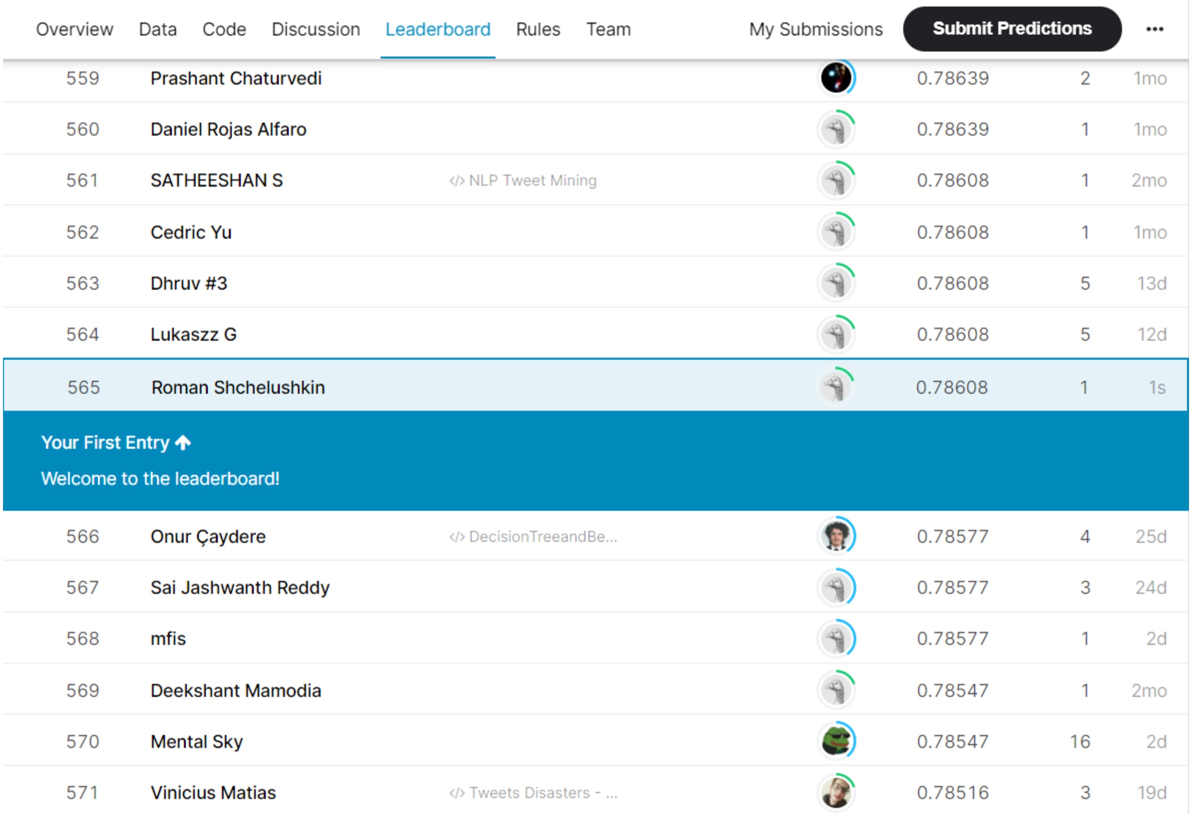
Ввод [289]:

```
pred_final.to_csv('Data/DisasterTweets/Pred1.csv',sep=',',header=True,index=False)
```

Ввод [293]:

```
Image("Data/DisasterTweets/Kaggle_compet.PNG")
```

Out[293]:



559	Prashant Chaturvedi		0.78639	2	1mo
560	Daniel Rojas Alfaro		0.78639	1	1mo
561	SATHEESHAN S	</> NLP Tweet Mining	0.78608	1	2mo
562	Cedric Yu		0.78608	1	1mo
563	Dhruv #3		0.78608	5	13d
564	Lukaszz G		0.78608	5	12d
565	Roman Shchelushkin		0.78608	1	1s

Your First Entry ↑
Welcome to the leaderboard!

566	Onur Çaydere	</> DecisionTreeandBe...	0.78577	4	25d
567	Sai Jashwanth Reddy		0.78577	3	24d
568	mfis		0.78577	1	2d
569	Deekshant Mamodia		0.78547	1	2mo
570	Mental Sky		0.78547	16	2d
571	Vinicius Matias	</> Tweets Disasters - ...	0.78516	3	19d

3.3 SVC model

Ввод [ ]:

```
%%time
parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],'C': [1, 10, 100, 1000]},
              {'kernel': ['linear'], 'C': [1, 10, 100, 1000]},
              {'kernel': ['poly'], 'gamma': [1e-3, 1e-4],'C': [1, 10, 100, 1000]}]
best_params_SVC = best_model(SVC(),parameters,X_train=X_train_emb,
                            y_train=y_train_emb,y_test=y_test_emb,X_test=X_test_emb,score =
```

Ввод [ ]:

```
bestmod = fitModel(SVC(C=best_params_SVC['C'],gamma = best_params_SVC['gamma'],kernel=best_
                   x_train=X_train_emb,y_train=y_train_emb,x_test=X_test_emb,y_test=y_test_

best_params_of_all_models.append(bestmod)
```

Ввод [ ]:

```
bestmod['accuracy_score']
```

3.4 DecisionTreeClassifier model

Ввод [220]:

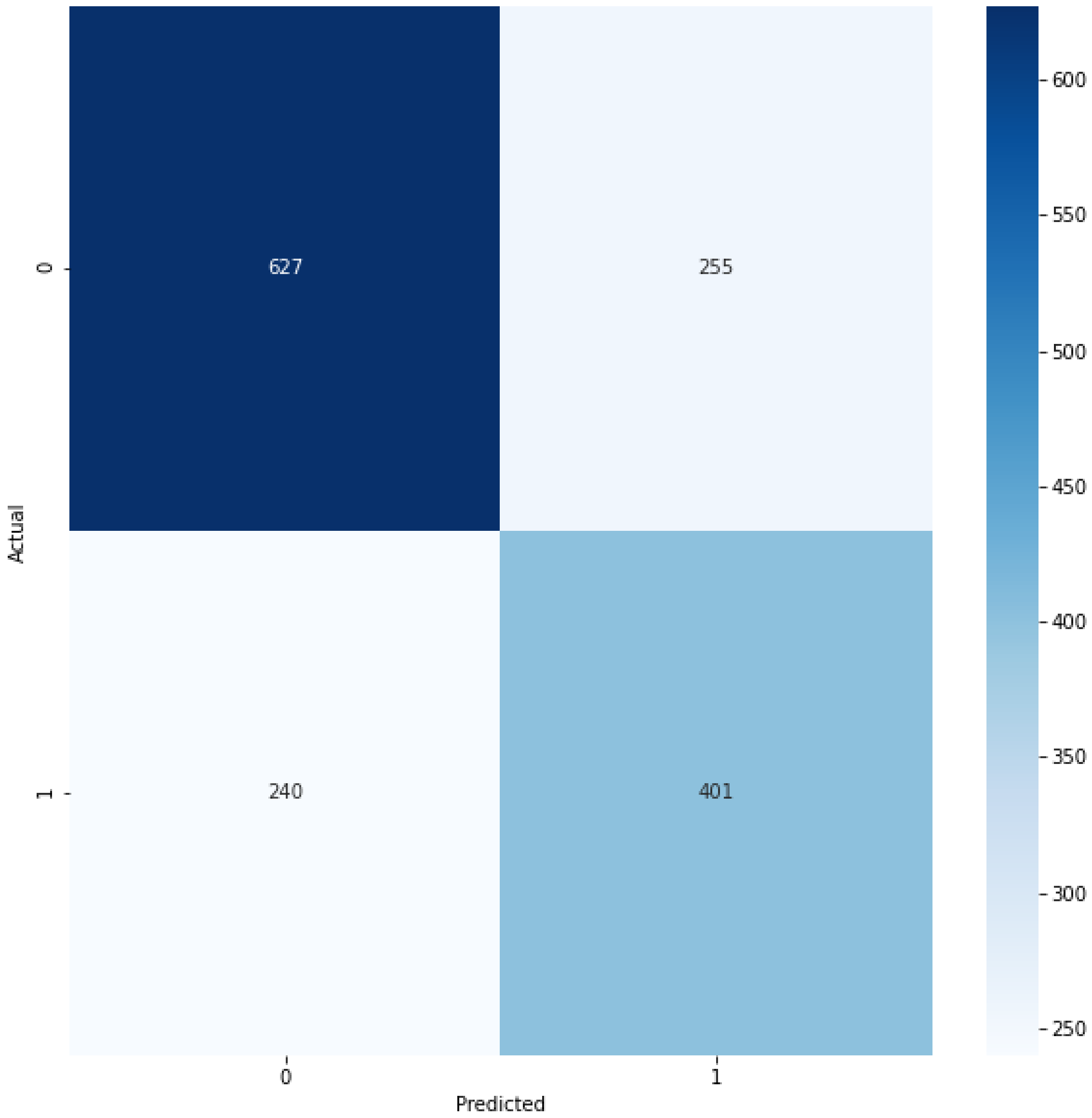
```
%%time
parameters = [{'criterion':['gini', 'entropy'],'splitter':['best', 'random'],'max_depth':[2
               'max_features':[None,'auto'],'class_weight':['balanced',None]}]
best_params_DTC = best_model(DecisionTreeClassifier(),parameters,X_train=X_train_emb,
                            y_train=y_train_emb,y_test=y_test_emb,X_test=X_test_emb,score =
```

```
# Tuning hyper-parameters for accuracy

Best parameters set found on development set:

{'class_weight': None, 'criterion': 'entropy', 'max_depth': 50, 'max_featu
res': None, 'splitter': 'random'}

Grid scores on development set:

0.685 (+/-0.017) for {'class_weight': 'balanced', 'criterion': 'gini', 'ma
x_depth': 20, 'max_features': None, 'splitter': 'best'}
0.676 (+/-0.020) for {'class_weight': 'balanced', 'criterion': 'gini', 'ma
x_depth': 20, 'max_features': None, 'splitter': 'random'}
0.670 (+/-0.011) for {'class_weight': 'balanced', 'criterion': 'gini', 'ma
x_depth': 20, 'max_features': 'auto', 'splitter': 'best'}
0.650 (+/-0.038) for {'class_weight': 'balanced', 'criterion': 'gini', 'ma
x_depth': 20, 'max_features': 'auto', 'splitter': 'random'}
0.685 (+/-0.017) for {'class_weight': 'balanced', 'criterion': 'gini', 'ma
x_depth': 30, 'max_features': None, 'splitter': 'best'}
```

Ввод [222]:

```
bestmod = fitModel(DecisionTreeClassifier(criterion=best_params_DTC['criterion'],splitter=b
                                        max_depth=best_params_DTC['max_depth'],max_features
                                        class_weight=best_params_DTC['class_weight']),
                   x_train=X_train_emb,y_train=y_train_emb,x_test=X_test_emb,y_test=y_test_

best_params_of_all_models.append(bestmod)
```

```
---time_fit model 1.0 seconds ---

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```



Ввод [223]:

```
bestmod['accuracy_score']
```

Out[223]:

```
0.6749835850295469
```

3.5 SGDClassifier model

Ввод [224]:

```
%%time
parameters = [{'loss' : ['hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron'],
               'max_iter' : [100,1000,2000],'tol':[10**(-3),10**(-2),10**(-4)]}]
best_params_SGD = best_model(SGDClassifier(),parameters,X_train=X_train_emb,
                            y_train=y_train_emb,y_test=y_test_emb,X_test=X_test_emb,score =
```

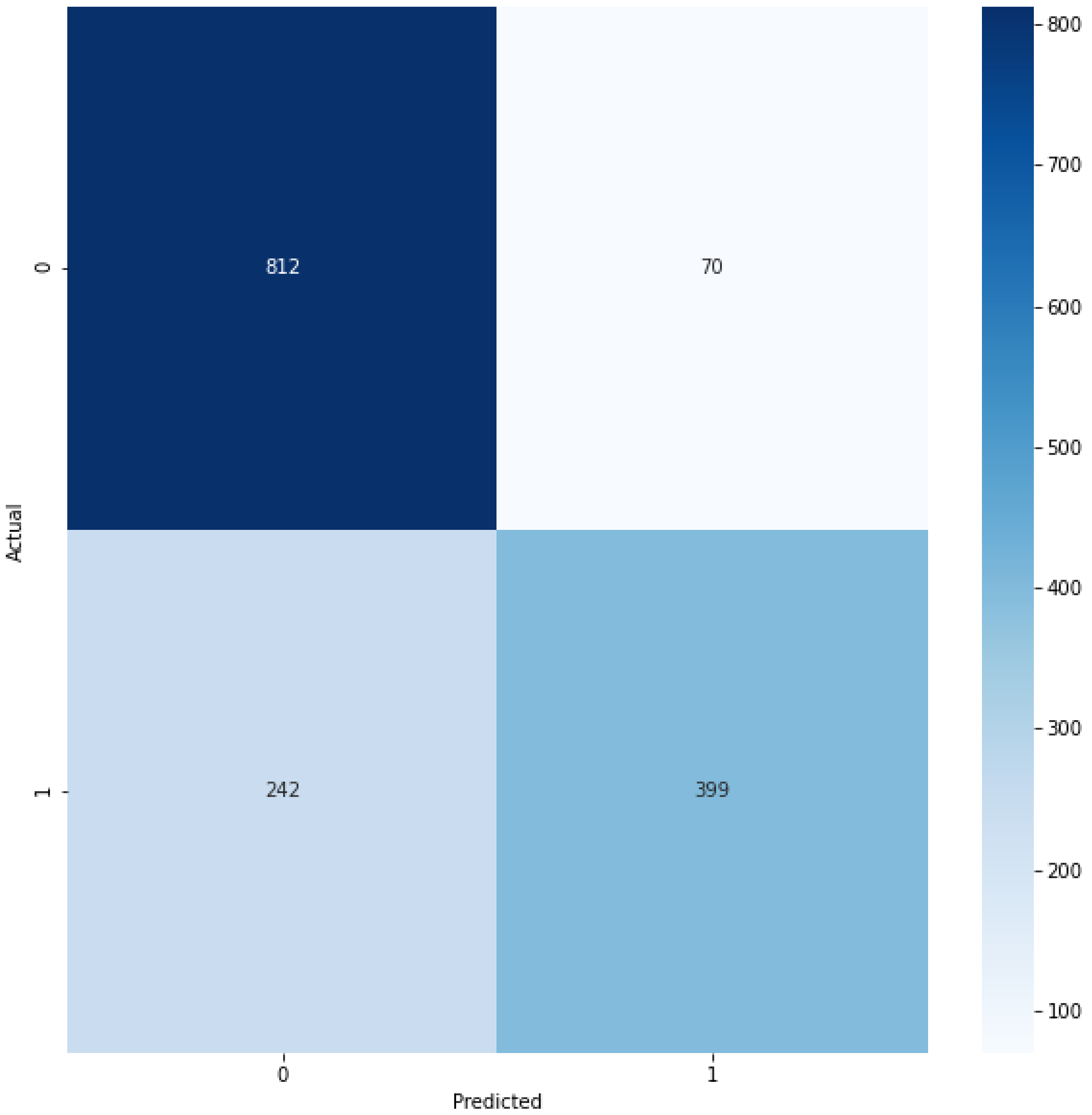
```
# Tuning hyper-parameters for accuracy

Best parameters set found on development set:

{'loss': 'log', 'max_iter': 1000, 'tol': 0.0001}

Grid scores on development set:

0.782 (+/-0.026) for {'loss': 'hinge', 'max_iter': 100, 'tol': 0.001}
0.771 (+/-0.029) for {'loss': 'hinge', 'max_iter': 100, 'tol': 0.01}
0.774 (+/-0.035) for {'loss': 'hinge', 'max_iter': 100, 'tol': 0.0001}
0.773 (+/-0.021) for {'loss': 'hinge', 'max_iter': 1000, 'tol': 0.001}
0.768 (+/-0.027) for {'loss': 'hinge', 'max_iter': 1000, 'tol': 0.01}
0.777 (+/-0.023) for {'loss': 'hinge', 'max_iter': 1000, 'tol': 0.0001}
0.781 (+/-0.010) for {'loss': 'hinge', 'max_iter': 2000, 'tol': 0.001}
0.772 (+/-0.027) for {'loss': 'hinge', 'max_iter': 2000, 'tol': 0.01}
0.784 (+/-0.014) for {'loss': 'hinge', 'max_iter': 2000, 'tol': 0.0001}
0.776 (+/-0.023) for {'loss': 'log', 'max_iter': 100, 'tol': 0.001}
0.765 (+/-0.033) for {'loss': 'log', 'max_iter': 100, 'tol': 0.01}
0.783 (+/-0.008) for {'loss': 'log', 'max_iter': 100, 'tol': 0.0001}
0.783 (+/-0.019) for {'loss': 'log', 'max_iter': 1000, 'tol': 0.001}
0.765 (+/-0.026) for {'loss': 'log', 'max_iter': 1000, 'tol': 0.01}
0.787 (+/-0.013) for {'loss': 'log', 'max_iter': 1000, 'tol': 0.0001}
0.776 (+/-0.017) for {'loss': 'log', 'max_iter': 2000, 'tol': 0.001}
0.746 (+/-0.097) for {'loss': 'log', 'max_iter': 2000, 'tol': 0.01}
0.782 (+/-0.009) for {'loss': 'log', 'max_iter': 2000, 'tol': 0.0001}
0.762 (+/-0.038) for {'loss': 'modified_huber', 'max_iter': 100, 'tol': 0.00
1}
0.753 (+/-0.042) for {'loss': 'modified_huber', 'max_iter': 100, 'tol': 0.0
1}
0.733 (+/-0.140) for {'loss': 'modified_huber', 'max_iter': 100, 'tol': 0.00
01}
0.743 (+/-0.041) for {'loss': 'modified_huber', 'max_iter': 1000, 'tol': 0.0
01}
0.756 (+/-0.024) for {'loss': 'modified_huber', 'max_iter': 1000, 'tol': 0.0
1}
0.768 (+/-0.009) for {'loss': 'modified_huber', 'max_iter': 1000, 'tol': 0.0
001}
0.765 (+/-0.029) for {'loss': 'modified_huber', 'max_iter': 2000, 'tol': 0.0
01}
0.733 (+/-0.149) for {'loss': 'modified_huber', 'max_iter': 2000, 'tol': 0.0
1}
0.750 (+/-0.021) for {'loss': 'modified_huber', 'max_iter': 2000, 'tol': 0.0
001}
0.672 (+/-0.036) for {'loss': 'squared_hinge', 'max_iter': 100, 'tol': 0.00
1}
0.668 (+/-0.043) for {'loss': 'squared_hinge', 'max_iter': 100, 'tol': 0.01}
0.682 (+/-0.031) for {'loss': 'squared_hinge', 'max_iter': 100, 'tol': 0.000
1}
0.674 (+/-0.008) for {'loss': 'squared_hinge', 'max_iter': 1000, 'tol': 0.00
1}
0.678 (+/-0.008) for {'loss': 'squared_hinge', 'max_iter': 1000, 'tol': 0.0
1}
```

```
0.679 (+/-0.028) for {'loss': 'squared_hinge', 'max_iter': 1000, 'tol': 0.00
01}
0.678 (+/-0.015) for {'loss': 'squared_hinge', 'max_iter': 2000, 'tol': 0.00
1}
0.678 (+/-0.013) for {'loss': 'squared_hinge', 'max_iter': 2000, 'tol': 0.0
1}
0.675 (+/-0.015) for {'loss': 'squared_hinge', 'max_iter': 2000, 'tol': 0.00
01}
0.709 (+/-0.107) for {'loss': 'perceptron', 'max_iter': 100, 'tol': 0.001}
0.750 (+/-0.025) for {'loss': 'perceptron', 'max_iter': 100, 'tol': 0.01}
0.657 (+/-0.179) for {'loss': 'perceptron', 'max_iter': 100, 'tol': 0.0001}
0.711 (+/-0.082) for {'loss': 'perceptron', 'max_iter': 1000, 'tol': 0.001}
0.695 (+/-0.109) for {'loss': 'perceptron', 'max_iter': 1000, 'tol': 0.01}
0.720 (+/-0.045) for {'loss': 'perceptron', 'max_iter': 1000, 'tol': 0.0001}
0.711 (+/-0.078) for {'loss': 'perceptron', 'max_iter': 2000, 'tol': 0.001}
0.718 (+/-0.076) for {'loss': 'perceptron', 'max_iter': 2000, 'tol': 0.01}
0.722 (+/-0.072) for {'loss': 'perceptron', 'max_iter': 2000, 'tol': 0.0001}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       0.79      0.88      0.83       882
           1       0.80      0.68      0.74       641

    accuracy                           0.79      1523
   macro avg       0.80      0.78      0.78      1523
weighted avg       0.79      0.79      0.79      1523


Wall time: 2min 12s
```

Ввод [225]:

```
bestmod = fitModel(SGDClassifier(loss=best_params_SGD['loss'],max_iter=best_params_SGD['max
                    x_train=X_train_emb,y_train=y_train_emb,x_test=X_test_emb,y_test=y_test_

best_params_of_all_models.append(bestmod)
```

```
---time_fit model 1.0 seconds ---

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```



Ввод [226]:

```
bestmod['accuracy_score']
```

Out[226]:

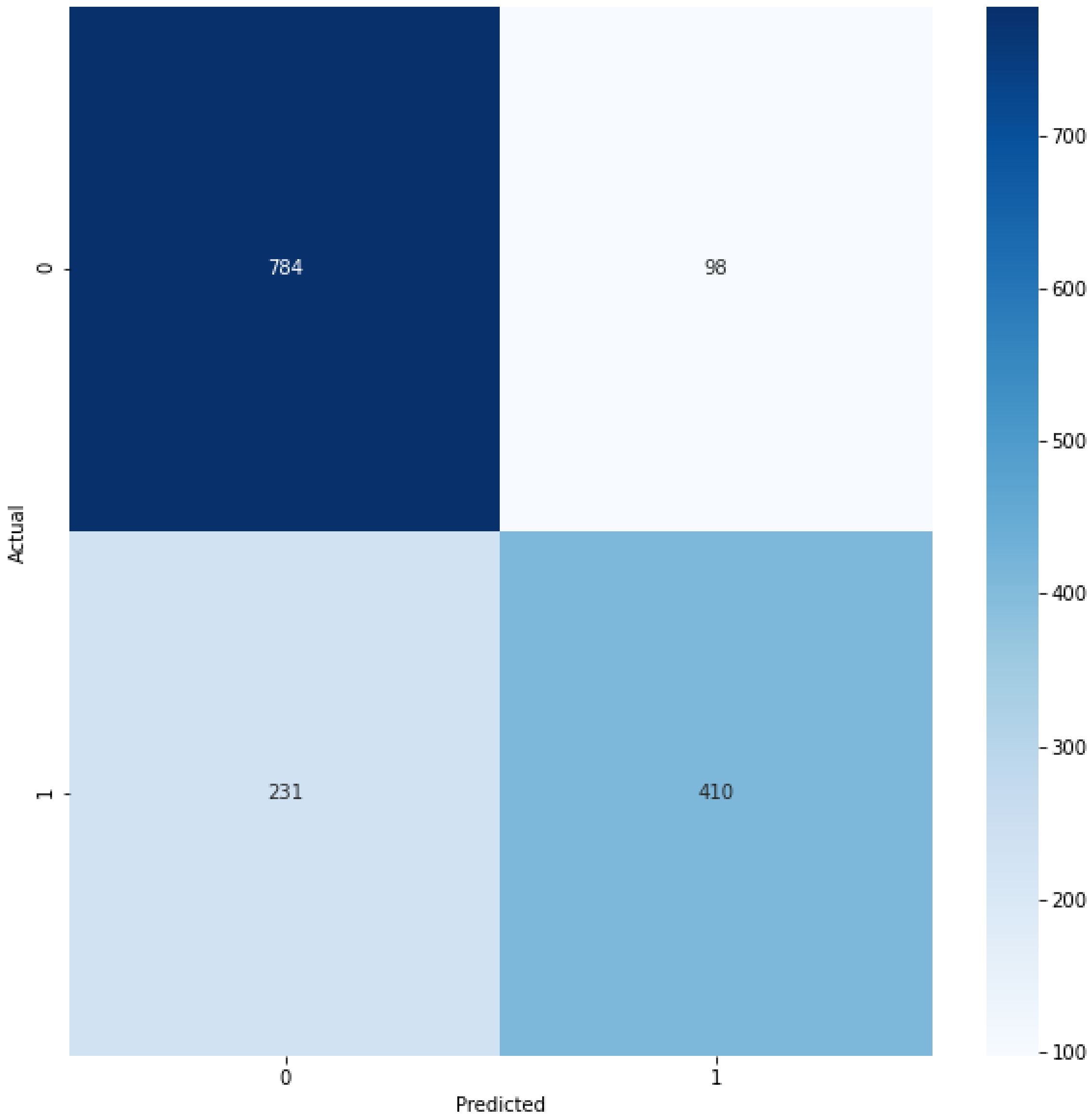
```
0.7951411687458962
```

3.6 RandomForestClassifier model

Ввод [229]:

```
bestmod = fitModel(RandomForestClassifier(n_estimators=50,criterion='entropy',random_state=
                   x_train=X_train_emb,y_train=y_train_emb,x_test=X_test_emb,y_test=y_test_

best_params_of_all_models.append(bestmod)
```

```
---time_fit model 12.0 seconds ---

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```



3.7 XGB Classifier model

Ввод [252]:

```
%%time
XGB = xgb.XGBClassifier()
XGB.fit(X_train_emb, y_train_emb)
pred_emb = XGB.predict(X_test_emb)
XGB_accuracy = accuracy_score(y_test_emb, pred_emb)

print("Accuracy: ",XGB_accuracy)
```

```
[20:46:16] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the d
efault evaluation metric used with the objective 'binary:logistic' was chang
ed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to re
store the old behavior.
Accuracy:  0.7997373604727511
Wall time: 50.7 s
```