LIME implementation on text data

In [1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import time
```

In [2]:
```
train = pd.read_csv(r"C:\Users\Jaswanth Reddy\Desktop\Image dataset\nlp(lime)\train.csv")
test = pd.read_csv(r"C:\Users\Jaswanth Reddy\Desktop\Image dataset\nlp(lime)\test.csv")
```

In [3]:
```
train.head()
```

Out[3]:

	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

In [4]:
```
test.head()
```

Out[4]:

	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...
2	3	NaN	NaN	there is a forest fire at spot pond, geese are...
3	9	NaN	NaN	Apocalypse lighting. #Spokane #wildfires
4	11	NaN	NaN	Typhoon Soudelor kills 28 in China and Taiwan

In [5]:
```
# To remove URL and HTML
import re


def remove_URL(text):
    url = re.compile(r"https?://\S+|www\.\S+")
    return url.sub(r"", text)


def remove_html(text):
    html = re.compile(r"<.*?>")
    return html.sub(r"", text)
```

In [9]:
```
# To remove punctuation
import string
def remove_punct(text):
    table = str.maketrans("", "", string.punctuation)
    return text.translate(table)
```

In [10]:
```
train["text"] = train.text.map(lambda x: remove_URL(x))
train["text"] = train.text.map(lambda x: remove_html(x))
train["text"] = train.text.map(lambda x: remove_punct(x))
```

In [13]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to C:\Users\Jaswanth
[nltk_data]     Reddy\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

Out[13]:
```
True
```

In [14]:
```
# To remove stopwords
from nltk.corpus import stopwords

stop = set(stopwords.words("english"))


def remove_stopwords(text):
    text = [word.lower() for word in text.split() if word.lower() not in stop]

    return " ".join(text)
```

In [15]:
```
train["text"] = train["text"].map(remove_stopwords)
```

In [16]:
```
# Stemming

from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()


def stemming(text):
    text = [stemmer.stem(word) for word in text.split()]

    return " ".join(text)

train["text"] = train["text"].map(stemming)
```

Bag of words

In [19]:
```
from sklearn.feature_extraction.text import CountVectorizer


def count_vect(data, ngrams=(1, 1)):
    count_vectorizer = CountVectorizer(ngram_range=ngrams)
    emb = count_vectorizer.fit_transform(data)

    return emb, count_vectorizer
```

In [20]:
```
train_counts, count_vectorizer = count_vect(train["text"])
test_counts = count_vectorizer.transform(test["text"])
```

In [21]:
```
train.shape
```

Out[21]:
```
(7613, 6)
```

In [27]:
```
print(train_counts)
```

```
  (0, 4486)     1
  (0, 12815)    1
  (0, 5259)     1
  (0, 9952)     1
  (0, 1144)     1
  (0, 6376)     1
  (0, 16508)    1
  (1, 6366)     1
  (1, 6168)     1
  (1, 10784)    1
  (1, 9035)     1
  (1, 13370)    1
  (1, 13657)    1
  (1, 2948)     1
  (2, 13086)    1
  (2, 1596)     1
  (2, 14052)    2
  (2, 11994)    2
  (2, 11055)    1
  (2, 11221)    1
  (2, 5696)     1
  (2, 11417)    1
  (2, 5792)     1
  (3, 5696)     1
  (3, 11417)    1
  :     :
  (7610, 16775) 1
  (7610, 7319)  1
  (7610, 6)     1
  (7610, 16552) 1
  (7611, 3010)  1
  (7611, 12094) 1
  (7611, 13212) 1
  (7611, 9414)  1
  (7611, 13931) 1
  (7611, 8268)  1
  (7611, 15724) 1
  (7611, 3642)  1
  (7611, 5282)  2
  (7611, 12147) 1
```

```
  (7611, 15082) 1
  (7611, 10994) 1
  (7611, 8140)  1
  (7612, 2900)  1
  (7612, 10857) 1
  (7612, 9125)  1
  (7612, 11023) 1
  (7612, 17136) 1
  (7612, 7618)  1
  (7612, 766)   1
  (7612, 12754) 1
```

TFIDF

In [22]:
```
from sklearn.feature_extraction.text import TfidfVectorizer


def tfidf(data, ngrams=(1, 1)):
    tfidf_vectorizer = TfidfVectorizer(ngram_range=ngrams)
    train = tfidf_vectorizer.fit_transform(data)

    return train, tfidf_vectorizer


train_tfidf, tfidf_vectorizer = tfidf(train["text"])
test_tfidf = tfidf_vectorizer.transform(test["text"])
```

In [23]:
```
[x for x in train_tfidf.todense()[0][0:].tolist()[0] if x != 0]
```

Out[23]:
```
[0.40602120282994786,
 0.47004586769653584,
 0.3260376260282067,
 0.47004586769653584,
 0.29098037199188415,
 0.3665665825282048,
 0.26061346373181266]
```

Logistic regression

In [28]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
```

Using count vectorizer

In [54]:
```
X = train_counts
y = train["target"].values

X_train_counts, X_test, y_train_counts, y_test = train_test_split(
    X, y, test_size=0.2, random_state=2020
)
```

In [55]:
```
model1 = LogisticRegression(class_weight="balanced")
model1.fit(X_train_counts, y_train_counts)
```

Out[55]:
```
LogisticRegression(class_weight='balanced')
```

In [56]:
```
y_pred = model.predict(X_test)

f1score = f1_score(y_test, y_pred)
print(f"Counts Model Score: {f1score * 100} %")
```

```
Counts Model Score: 77.12230215827338 %
```

Using TF-IDF vectorizer

In [32]:
```
X = train_tfidf
y = train["target"].values

X_train_tfidf, X_test, y_train_tfidf, y_test = train_test_split(
    X, y, test_size=0.2, random_state=2020
)
```

In [33]:
```
model = LogisticRegression(class_weight="balanced")
model.fit(X_train_tfidf, y_train_tfidf)
```

Out[33]:
```
LogisticRegression(class_weight='balanced')
```

In [34]:
```
y_pred = model.predict(X_test)

f1score = f1_score(y_test, y_pred)
print(f"Tfidf Model Score: {f1score * 100} %")
```

```
Tfidf Model Score: 76.64576802507837 %
```

LIME implementation using TFIDF

In [35]:
```
from sklearn.pipeline import make_pipeline

pipe = make_pipeline(tfidf_vectorizer, model)
```

In [36]:
```
test.text.iloc[0]
```

Out[36]:
```
'Just happened a terrible car crash'
```

In [37]:
```
print(pipe.predict_proba([test.text.iloc[0]]))
```

```
[[0.33891714 0.66108286]]
```

In [38]:
```
from lime.lime_text import LimeTextExplainer

explainer = LimeTextExplainer(class_names=["irrelevant", "disaster"])
```

In [64]:
```
# number of words in a large sentence
max_features = test.text.str.split().map(lambda x: len(x)).max()
print(max_features)
```

```
31
```

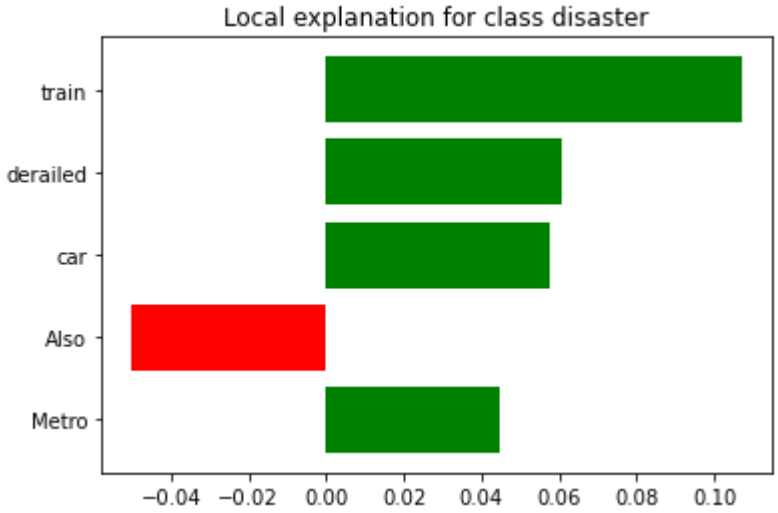
In [52]:
```
import random
#random seed represents to select particular row from a table
random.seed(13)
idx = random.randint(0, len(test))

exp = explainer.explain_instance(
    test.text.iloc[idx], pipe.predict_proba, num_features=5 # number of features
)
```

In [53]:
```
print(
    f"Probability of the sample to be a disaster is: {pipe.predict_proba([test.text.iloc[0]])[0, 1]}\n"
)
print(f"Explanation as a list of weighted features:")
exp.as_list()
```

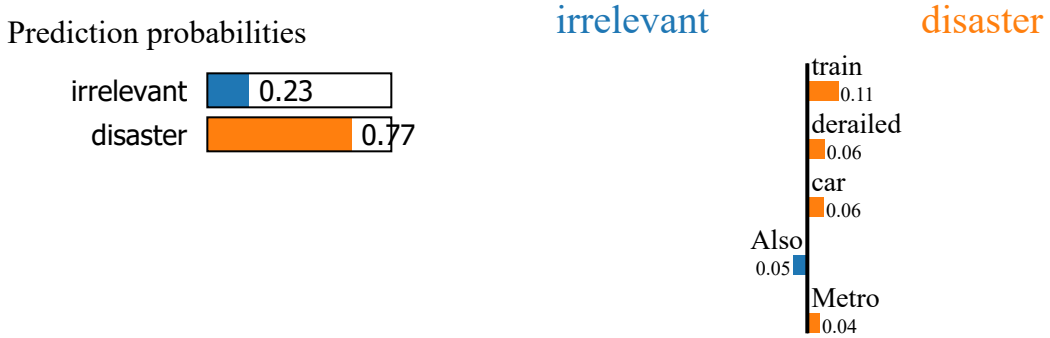
```
Probability of the sample to be a disaster is: 0.661082857740497

Explanation as a list of weighted features:
```

Out[53]:
```
[('train', 0.10523086675973924),
 ('derailed', 0.06157057284528654),
 ('car', 0.05765422179480754),
 ('Also', -0.05340760475256339),
 ('Metro', 0.04653502292034725)]
```

In [41]:
```
f = exp.as_pyplot_figure()
```



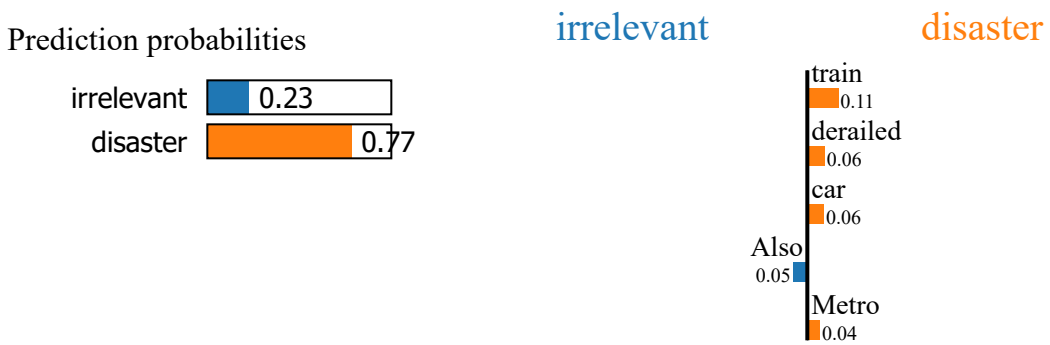
In [42]:
```
exp.show_in_notebook(text=True) # text=True to represent text with highlighted words
```



Text with highlighted words

Also there is no estimate of damage yet from #WMATA/#Metro on the six-car train that derailed ~5 a.m. Shuttle service available. @CQnow

In [47]:
```
exp.show_in_notebook(text=False)
```



In [ ]:
```
### From the above result we can conclude that "also" keyword is effecting the statement that it is not a disast
```

Lime implementaion(Bag of words)

In [57]:
```
from sklearn.pipeline import make_pipeline

pipe = make_pipeline(count_vectorizer, model1)
```

In [58]:
```
test.text.iloc[0]
```

Out[58]:
```
'Just happened a terrible car crash'
```

In [59]:
```
from lime.lime_text import LimeTextExplainer

explainer = LimeTextExplainer(class_names=["irrelevant", "disaster"])
```

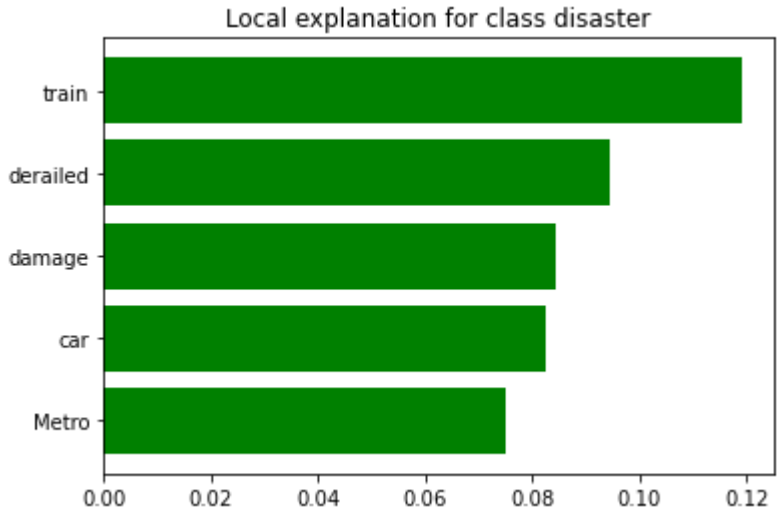
In [60]:
```
import random

random.seed(13)
idx = random.randint(0, len(test))

exp = explainer.explain_instance(
    test.text.iloc[idx], pipe.predict_proba, num_features=5   # number of features
)
```

In [61]:
```
print(
    f"Probability of the sample to be a disaster is: {pipe.predict_proba([test.text.iloc[0]])[0, 1]}\n"
)
print(f"Explanation as a list of weighted features:")
exp.as_list()
```

```
Probability of the sample to be a disaster is: 0.6621196784743593

Explanation as a list of weighted features:
```

Out[61]:
```
[('train', 0.11917705613639987),
 ('derailed', 0.09447202622730055),
 ('damage', 0.08421561083857573),
 ('car', 0.08259027297462833),
 ('Metro', 0.07518087796770903)]
```

In [62]:
```
f = exp.as_pyplot_figure()
```



In [63]:
```
exp.show_in_notebook(text=True)
```

Prediction probabilities

irrelevant 0.01

disaster 0.99

irrelevant disaster

train 0.12

derailed 0.09

damage 0.08

car 0.08

Metro 0.08

Text with highlighted words

Also there is no estimate of damage yet from #WMATA/#Metro on the six-car train that derailed ~5 a.m. Shuttle service available. @CQnow

In [ ]: