Tweets

March 6, 2021

0.1 Sentiment Analysis with Disaster Tweets (Kaggle Competition Notebook)

0.1.1 Author: Georg Zhelev

This natural language processing projekt looks to determine when a natural disaster is taking place based on tweets from Twitter. SciKit Learn's bag of words method is applied coupled with a logistic regression, the parameters of which are chosen using grid search. Because the sample is about 60:40 unbalanced, a startified cross validation is used to adjust prediction expectations. This model serves as a baseline for comparison with more advanced prediction methods.

0.1.2 Mount Drive

```
[ ]: from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

0.1.3 Import Tools

```
[ ]: import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.feature_extraction.text import CountVectorizer

     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import cross_val_score
     from sklearn.model_selection import StratifiedKFold
     from sklearn.model_selection import GridSearchCV

     import numpy as np
     from google.colab import files
```

0.1.4 Load Data

```
[ ]: df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Econ/Disaster Tweets/
      ↪train.csv")
```

"df" is the test data

0.1.5 Explore Data

```
[ ]: df.head()
```

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

     [5 rows x 5 columns]
```

```
[ ]: df.shape
```

```
[ ]: (7613, 5)
```

1: positive(forest fire) 0: negative (no forest fire)

```
[ ]: round(df.target.mean(),2)
```

```
[ ]: 0.43
```

```
[ ]: df.target.value_counts(normalize=True)
```

```
[ ]: 0    0.57034
     1    0.42966
     Name: target, dtype: float64
```

More tweets are not about forest fires (57%)

```
[ ]: df.isna().any()
```

```
[ ]: id          False
     keyword      True
     location     True
     text        False
     target      False
     dtype: bool
```

```
[ ]: print("Missing:")
     print("keyword missing:", df.keyword.isna().sum())
     print("location missing:", df.location.isna().sum())
```

```
Missing:
keyword missing: 61
location missing: 2533
```

What to do with missing location?

- For now don't use it as an explanation variable (worse)
- Use it as explanation variable, but drop those who are missing (better)
- Impute it somehow (best)

0.1.6 Pre-Process

```
[ ]: df1=df.dropna()
```

```
[ ]: 7613-(2533+61)
```

```
[ ]: 5019
```

Not exactly 5080 rowls, because some are both NA for Location and Keyword

```
[ ]: df1.isna().any()
```

```
[ ]: id          False
     keyword     False
     location    False
     text        False
     target      False
     dtype: bool
```

0.1.7 Split Data

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(df["text"], df["target"],␣
      ↪test_size=0.33, stratify=df['target'], random_state=1)
```

```
[ ]: print(y_train.value_counts(y_train[1]))
```

```
0    0.570392
1    0.429608
Name: target, dtype: float64
```

```
[ ]: print(y_test.value_counts(y_test[0]))
```

```
0    0.570235
1    0.429765
Name: target, dtype: float64
```

```
[ ]: print(y_train.shape)
     print(y_test.shape)
```

```
(5100,)
(2513,)
```

```
[ ]: print(X_train.shape)
```

```
(5100,)
```

```
[ ]: X_train.head()
```

```
[ ]: 7360    'My Fifty Online Dates and why I'm still singl…
     826     Tomorrow's Announcement VODs http://t.co/cUbze…
     1877    @jaureguiswisdom lmao well i only know one and…
     7596    RT @LivingSafely: #NWS issues Severe #Thunders…
     4330    Criminals Who Hijack Lorries And Buses Arreste…
     Name: text, dtype: object
```

0.1.8 Bag of Words

```
[ ]: vect = CountVectorizer(min_df=3)
     vect.fit(X_train)
     X_train_bag = vect.transform(X_train)
     print('X_train:', repr(X_train_bag))
```

```
X_train: <5100x3330 sparse matrix of type '<class 'numpy.int64'>'
        with 59830 stored elements in Compressed Sparse Row format>
```

```
[ ]: features = vect.get_feature_names()
     print('First 10 features:', features[:10])
     print('Features 400 to 401', features[400:401])
     print('Every 100th feature:', features[::100])
```

```
First 10 features: ['00', '01', '02', '03', '04', '05', '06', '07', '08', '09']
Features 400 to 401 ['blanket']
Every 100th feature: ['00', 'aba', 'among', 'autumn', 'blanket', 'business',
'chile', 'cool', 'debate', 'dozens', 'eq', 'feat', 'friend', 'grown', 'hobbit',
'injured', 'kiernan', 'list', 'max', 'moral', 'north', 'page', 'plot', 'quake',
'rescue', 'salvador', 'shelter', 'southbound', 'sunday', 'thing', 'trfc',
'vancouver', 'whales', 'you']
```

0.1.9 Fit Model

```
[ ]: stkfold = StratifiedKFold(n_splits=5, shuffle=True)

     param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}

     grid = GridSearchCV(LogisticRegression(max_iter=100000), param_grid, cv=stkfold)
     grid.fit(X_train_bag, y_train)
```

```
[ ]: GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=True),
                  error_score=nan,
                  estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                               fit_intercept=True,
                                               intercept_scaling=1, l1_ratio=None,
                                               max_iter=100000, multi_class='auto',
                                               n_jobs=None, penalty='l2',
                                               random_state=None, solver='lbfgs',
                                               tol=0.0001, verbose=0,
                                               warm_start=False),
                  iid='deprecated', n_jobs=None,
                  param_grid={'C': [0.001, 0.01, 0.1, 1, 10]},
                  pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                  scoring=None, verbose=0)
```

[ ]:
```
print('Best cross-validation score:', grid.best_score_)
print('Best parameters:', grid.best_params_)
X_test_bag = vect.transform(X_test)
print(X_test_bag.shape)

print('Test accuracy', grid.score(X_test_bag, y_test))
```

```
Best cross-validation score: 0.7888235294117647
Best parameters: {'C': 0.1}
(2513, 3330)
Test accuracy 0.8038201352964585
```

Load Test Data

[ ]:
```
test = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Econ/Disaster Tweets/
 ↪test.csv")
```

[ ]:
```
test.shape
```

[ ]:
```
(3263, 4)
```

[ ]:
```
test.isna().any()
```

[ ]:
```
id          False
keyword      True
location     True
text        False
dtype: bool
```

[ ]:
```
print("keyword missing:", test.keyword.isna().sum())
print("location missing:", test.location.isna().sum())
```

```
keyword missing: 26
location missing: 1105
```

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

```
[ ]:    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
```

```
[ ]: id=test.id
```

```
[ ]: id.head()
```

```
[ ]: 0     0
     1     2
     2     3
     3     9
     4    11
     Name: id, dtype: int64
```

```
[ ]: X_test_k=test["text"]
     print(X_test_k.head())
```

```
0                   Just happened a terrible car crash
1    Heard about #earthquake is different cities, s…
2    there is a forest fire at spot pond, geese are…
3             Apocalypse lighting. #Spokane #wildfires
4        Typhoon Soudelor kills 28 in China and Taiwan
Name: text, dtype: object
```

here it is important to use the already fitted vocabulary using the count vectorizer and not fit a new one

```
[ ]: #vect.fit(X_test_k)
     X_test_k_bag = vect.transform(X_test_k)
     print('X_test_k:', repr(X_test_k_bag))
```

```
X_test_k: <3263x3330 sparse matrix of type '<class 'numpy.int64'>'
        with 36883 stored elements in Compressed Sparse Row format>
```

0.1.10 Make Predictions

```
[ ]: X_test_k_bag.shape
```

```
[ ]: (3263, 3330)
```

```
[ ]: X_train_bag.shape
```

[ ]:
```
(5100, 3330)
```

[ ]:
```
# predict results
predictions = grid.predict(X_test_k_bag)
```

[ ]:
```
predictions.shape
```

[ ]:
```
(3263,)
```

[ ]:
```
predictions.head()
```

[ ]:
```
0    0
1    0
2    1
3    0
4    1
Name: target, dtype: int64
```

[ ]:
```
predictions = pd.Series(predictions, name="target")
submission = pd.concat([pd.Series(id),predictions], axis = 1)
```

[ ]:
```
submission.shape
```

[ ]:
```
(3263, 2)
```

[ ]:
```
submission.head()
```

[ ]:
```
   id  target
0   0       0
1   2       0
2   3       1
3   9       0
4  11       1
```

[ ]:
```
submission.to_csv("tweets_pred.csv",index=False)
files.download("tweets_pred.csv")
```

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


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

[ ]:
```
#Kaggle Competition Result

#Accuracy on unlabeled test set: 0.78332
```