5-Class Sentiments Analysis using Ada Boosting

May 16, 2020

Data Exploration

[1]:
```
# importing require libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# Reading dataset of train and test
train_df = pd.read_csv('data/sentiment_5_class_train.csv')
test_df = pd.read_csv('data/sentiment_5_class_test.csv')
```

[3]:
```
train_df.head()
```

[3]:

	Phrase	Sentiment
0	the prisoner	2
1	The sheer joy and pride they took in their wor…	3
2	has never made a more sheerly beautiful film t…	3
3	the story has the sizzle of old news that has …	3
4	far superior	4

[4]:
```
test_df.head()
```

[4]:

	Phrase	Sentiment
0	makes for a touching love story , mainly becau…	3
1	a truly magical movie	4
2	check	3
3	is a remarkably accessible and haunting film .	4
4	are too cute	3

[5]:
```
# checking the shape of dataset
print('train_df shape: ', train_df.shape)
print('test_df shape: ', test_df.shape)
```

```
train_df shape:  (14711, 2)
test_df shape:  (3678, 2)
```

[6]:
```
# checking the null values
train_df.isnull().sum()
```

[6]:
```
Phrase       0
Sentiment    0
dtype: int64
```

[7]:
```
test_df.isnull().sum()
```

[7]:
```
Phrase       0
Sentiment    0
dtype: int64
```

[8]:
```
# checking the number of sentiments
train_df.Sentiment.unique()
```

[8]:
```
array([2, 3, 4, 0, 1])
```

Here, above sentiments represents:
0 : Very Bad
1 : Bad
2 : Normal
3 : Good
4 : Very Good

[9]:
```
# checking if there is empty phrase or not
empty_train = train_df[train_df.Phrase.apply(lambda x: len(x.split()) == 0)]
empyt_test = test_df[test_df.Phrase.apply(lambda x: len(x.split()) == 0)]

print('Empty train Phrase: \n', empty_train)
print('Empty test Phrase: \n', empyt_test)
```

```
Empty train Phrase:
      Phrase  Sentiment
6189                 1
Empty test Phrase:
 Empty DataFrame
Columns: [Phrase, Sentiment]
Index: []
```

Here, we have empty `Phrase` at training set but not in test set.

[10]:
```
# removing empty Phrase index
train_df.drop(empty_train.index, inplace=True)
```

[11]:
```
# Total number of Phrase with sentiments in train set
sentiments_collection_train = train_df.groupby('Sentiment').size()
sentiments_collection_test = test_df.groupby('Sentiment').size()
```

```
print('In train set: \n', sentiments_collection_train)
print('\n In test set: \n', sentiments_collection_test)
```

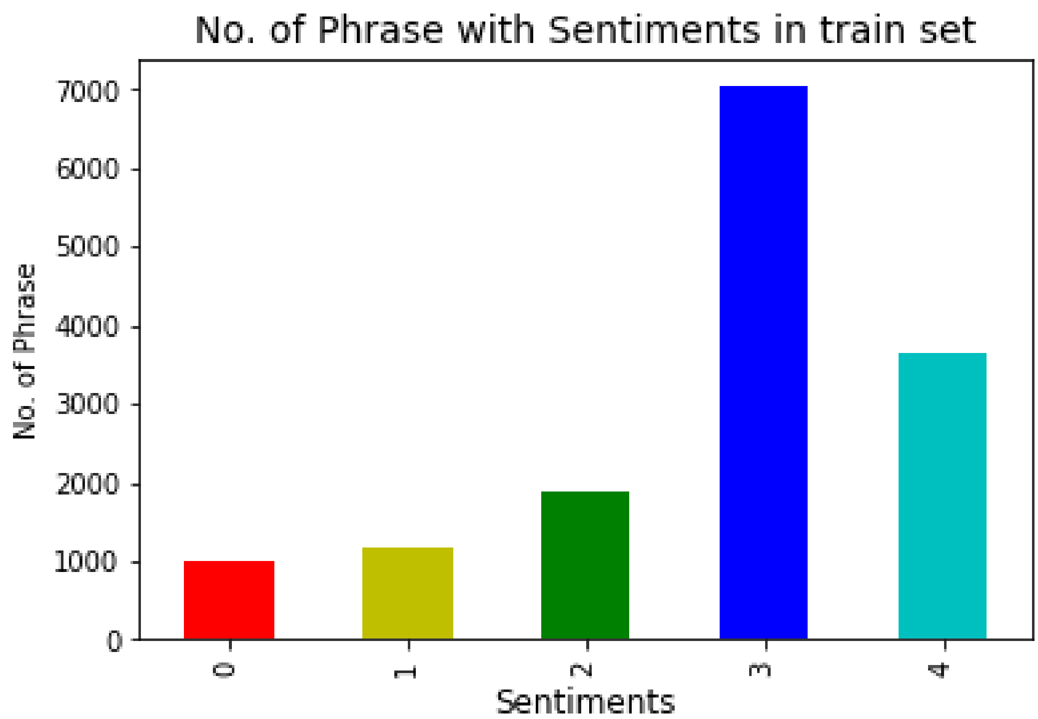
```
In train set:
 Sentiment
0     988
1    1164
2    1876
3    7033
4    3649
dtype: int64

 In test set:
 Sentiment
0     247
1     291
2     469
3    1759
4     912
dtype: int64
```

[12]:
```
# barplot of no.of phrase with sentiments in train set
sentiments_collection_train.plot(kind='bar', color=['r', 'y', 'g', 'b', 'c'])

plt.title('No. of Phrase with Sentiments in train set', fontsize=14)
plt.xlabel('Sentiments', fontsize=12)
plt.ylabel('No. of Phrase')

plt.show()
```

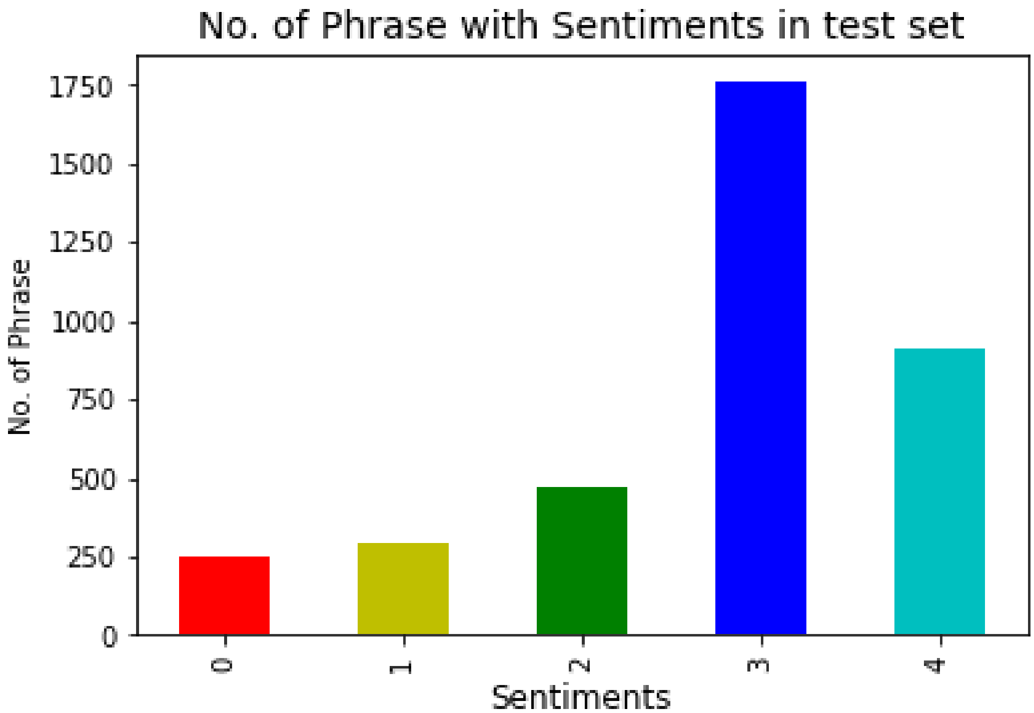


[13]:

```python
# barplot of no.of phrase with sentiments in test set
sentiments_collection_test.plot(kind='bar', color=['r', 'y', 'g', 'b', 'c'])

plt.title('No. of Phrase with Sentiments in test set', fontsize=14)
plt.xlabel('Sentiments', fontsize=12)
plt.ylabel('No. of Phrase')

plt.show()
```

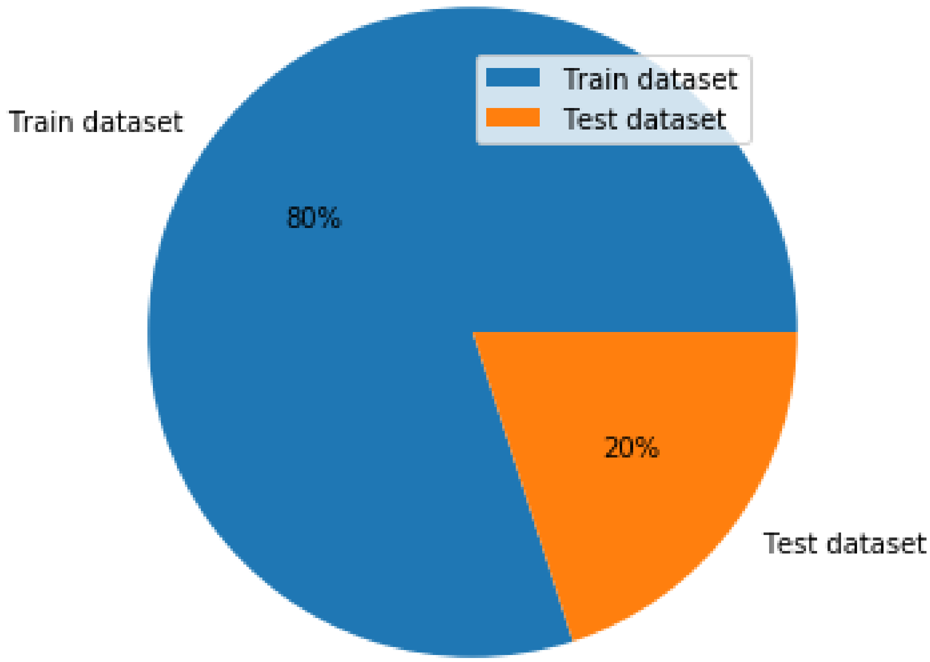


This diagram shows that, we have more number of positive phrase than the negative phrase. We can also observe that the data is inbalanced because label 3 is higer than other. so, to make a balanced dataset we can use `SMOOTEING` method.

[14]:
```
# lets check the distribution of train and test set
train_len = len(train_df)
test_len = len(test_df)

plt.pie(x=[train_len, test_len], labels=['Train dataset', 'Test dataset'],␣
 ↪autopct='%1.0f%%', radius=1.4)
plt.legend()

plt.show()
```



Feature Extraction and Preprocessing

Splitting train and test set.

[15]:
```
X_train = train_df['Phrase'].tolist()
y_train = train_df['Sentiment']

X_test = test_df['Phrase'].tolist()
y_test = test_df['Sentiment']

print('X_train length: ', len(X_train))
print('X_test length: ', len(X_test))
print('\n')

print('y_train length: ', len(y_train))
print('y_test length: ', len(y_test))
```

```
X_train length:  14710
X_test length:  3678


y_train length:  14710
y_test length:  3678
```

Using `Term Frequency - Inverse Dense Frequency (TF-IDF)` for vectorizatioin.

[16]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
vectorizer.fit(X_train)
```

[16]:
```
TfidfVectorizer()
```

[17]:
```
X_train_v = vectorizer.transform(X_train)
X_test_v = vectorizer.transform(X_test)

print('X_train_v shape: ', X_train_v.shape)
print('X_test_v shape: ', X_test_v.shape)
```

```
X_train_v shape:  (14710, 7115)
X_test_v shape:  (3678, 7115)
```

We, know that the data is imbalanced. So, we are using `SMOOTE Oversampling` to make the data balanced.

[18]:
```
from collections import Counter
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42, sampling_strategy='auto')
X_train_smote, y_train_smote = smote.fit_resample(X_train_v, y_train)
```

```
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.neighbors.base module is  deprecated in version 0.22 and will be removed
in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.neighbors. Anything that cannot be imported from
sklearn.neighbors is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.ensemble.bagging module is  deprecated in version 0.22 and will be
removed in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.ensemble. Anything that cannot be imported from
sklearn.ensemble is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.ensemble.base module is  deprecated in version 0.22 and will be removed
in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.ensemble. Anything that cannot be imported from
sklearn.ensemble is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
```

```
sklearn.ensemble.forest module is  deprecated in version 0.22 and will be
removed in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.ensemble. Anything that cannot be imported from
sklearn.ensemble is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.utils.testing module is  deprecated in version 0.22 and will be removed
in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.utils. Anything that cannot be imported from sklearn.utils
is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.metrics.classification module is  deprecated in version 0.22 and will be
removed in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.metrics. Anything that cannot be imported from
sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:86: FutureWarning: Function safe_indexing
is deprecated; safe_indexing is deprecated in version 0.22 and will be removed
in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

[19]:
```
Counter(y_train_smote)
```

[19]:
```
Counter({2: 7033, 3: 7033, 4: 7033, 0: 7033, 1: 7033})
```

Now, our dataset is balanced.

Grid Search

[20]:
```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(random_state=1,max_depth=5,min_samples_split=10)

grid_params={
    'n_estimators':(5,8,11,13),
    'learning_rate':(0.1, 0.001, 0.5),

}
```

[22]:
```
from sklearn.svm import SVC
from sklearn.metrics import make_scorer,f1_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import GridSearchCV

scorer = make_scorer(f1_score, average='macro')
```

```
clf = GridSearchCV(AdaBoostClassifier(base_estimator=dt), grid_params,␣
↪scoring=scorer)
clf.fit(X_train_smote, y_train_smote)
```

[22]:

```
GridSearchCV(estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassifier(
max_depth=5,
min_samples_split=10,
random_state=1)),
             param_grid={'learning_rate': (0.1, 0.001, 0.5),
                         'n_estimators': (5, 8, 11, 13)},
             scoring=make_scorer(f1_score, average=macro))
```

[23]:

```
best_score = clf.best_score_

print(best_score)
```

```
0.5031366322858036
```

[24]:

```
best_params = clf.best_params_

print(best_params)
```

```
{'learning_rate': 0.5, 'n_estimators': 13}
```

[25]:

```
print('The best score is: {} with params {}'.format(best_score, best_params))
```

```
The best score is: 0.5031366322858036 with params {'learning_rate': 0.5,
'n_estimators': 13}
```

Model Evaluation

[26]:

```
from sklearn import metrics

model=AdaBoostClassifier(random_state=1,learning_rate=0.
↪5,n_estimators=13,base_estimator=dt)
model.fit(X_train_smote, y_train_smote)

y_pred = model.predict(X_test_v)
print(metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.49      0.34      0.40       247
           1       0.22      0.13      0.16       291
           2       0.23      0.78      0.35       469
           3       0.53      0.24      0.33      1759
           4       0.45      0.46      0.45       912

    accuracy                           0.36      3678
```

```
   macro avg       0.38      0.39      0.34      3678
weighted avg       0.44      0.36      0.35      3678
```

[ ]: