Final-Code-part-2

December 20, 2022

0.0.1 Data import

[11]:
```
import pandas as pd
import swifter
import warnings
warnings.filterwarnings('ignore')

df=pd.read_csv(r'Reviews.csv')

df=df.drop(['Id', 'ProfileName'], axis=1)
df=df.dropna()
display(df, df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 568427 entries, 0 to 568453
Data columns (total 8 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   ProductId               568427 non-null  object
 1   UserId                  568427 non-null  object
 2   HelpfulnessNumerator    568427 non-null  int64
 3   HelpfulnessDenominator  568427 non-null  int64
 4   Score                   568427 non-null  int64
 5   Time                    568427 non-null  int64
 6   Summary                 568427 non-null  object
 7   Text                    568427 non-null  object
dtypes: int64(4), object(4)
memory usage: 39.0+ MB
```

	ProductId	UserId	HelpfulnessNumerator	\
0	B001E4KFG0	A3SGXH7AUHU8GW	1	
1	B00813GRG4	A1D87F6ZCVE5NK	0	
2	B000LQOCH0	ABXLMWJIXXAIN	1	
3	B000UA0QIQ	A395BORC6FGVXV	3	
4	B006K2ZZ7K	A1UQRSCLF8GW1T	0	
…	…	…	…	
568449	B001EO7N10	A28KG5XORO54AY	0	
568450	B003S1WTCU	A3I8AFVPEE8KI5	0	
568451	B004I613EE	A121AA1GQV751Z	2	

```
568452  B004I613EE   A3IBEVCTXKNOH                     1
568453  B001LR2CU2  A3LGQPJCZVL9UC                     0

        HelpfulnessDenominator  Score        Time  \
0                            1      5  1303862400
1                            0      1  1346976000
2                            1      4  1219017600
3                            3      2  1307923200
4                            0      5  1350777600
…                          …    …          …
568449                       0      5  1299628800
568450                       0      2  1331251200
568451                       2      5  1329782400
568452                       1      5  1331596800
568453                       0      5  1338422400

                                 Summary  \
0                    Good Quality Dog Food
1                        Not as Advertised
2                    "Delight" says it all
3                           Cough Medicine
4                              Great taffy
…                                      …
568449                 Will not do without
568450                       disappointed
568451            Perfect for our maltipoo
568452  Favorite Training and reward treat
568453                       Great Honey

                                                     Text
0       I have bought several of the Vitality canned d…
1       Product arrived labeled as Jumbo Salted Peanut…
2       This is a confection that has been around a fe…
3       If you are looking for the secret ingredient i…
4       Great taffy at a great price.  There was a wid…
…                                                     …
568449  Great for sesame chicken..this is a good if no…
568450  I'm disappointed with the flavor. The chocolat…
568451  These stars are small, so you can give 10-15 o…
568452  These are the BEST treats for training and rew…
568453  I am very satisfied ,product is as advertised,…

[568427 rows x 8 columns]

None
```

0.0.2 Data preprocessing

[12]:
```
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import swifter
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

nltk.download('stopwords')

stop_words=set(stopwords.words('english'))

def pre_processing(doc):
    doc = word_tokenize(doc.lower()) #making a list of cell in dataset
    doc = [w for w in doc if not w in stop_words] #removing the stop words
    doc = [re.sub(r'[^a-zA-Z ]+', '', item) for item in doc] #removing all␣
 ↪characters other than character
    doc = [lemmatizer.lemmatize(token) for token in doc] #lemmatization
    doc = ' '.join(doc)
    doc = doc.strip() #removing empty space
    return doc
```

```
[nltk_data] Error loading stopwords: <urlopen error [SSL:
[nltk_data]     CERTIFICATE_VERIFY_FAILED] certificate verify failed:
[nltk_data]     unable to get local issuer certificate (_ssl.c:997)>
```

[13]:
```
import nltk
nltk.download('punkt')
df['Text_pre']=df['Text'].swifter.apply(lambda l: pre_processing(l))
df
```

```
[nltk_data] Error loading punkt: <urlopen error [SSL:
[nltk_data]     CERTIFICATE_VERIFY_FAILED] certificate verify failed:
[nltk_data]     unable to get local issuer certificate (_ssl.c:997)>
```

```
Pandas Apply:   0%|          | 0/568427 [00:00<?, ?it/s]
```

[13]:

	ProductId	UserId	HelpfulnessNumerator	\
0	B001E4KFG0	A3SGXH7AUHU8GW	1	
1	B00813GRG4	A1D87F6ZCVE5NK	0	
2	B000LQOCH0	ABXLMWJIXXAIN	1	
3	B000UA0QIQ	A395BORC6FGVXV	3	
4	B006K2ZZ7K	A1UQRSCLF8GW1T	0	
…	…	…	…	
568449	B001EO7N10	A28KG5XORO54AY	0	
568450	B003S1WTCU	A3I8AFVPEE8KI5	0	

```
568451  B004I613EE  A121AA1GQV751Z                    2
568452  B004I613EE   A3IBEVCTXKNOH                    1
568453  B001LR2CU2  A3LGQPJCZVL9UC                    0

        HelpfulnessDenominator  Score        Time  \
0                            1      5  1303862400
1                            0      1  1346976000
2                            1      4  1219017600
3                            3      2  1307923200
4                            0      5  1350777600
…                          …    …          …
568449                       0      5  1299628800
568450                       0      2  1331251200
568451                       2      5  1329782400
568452                       1      5  1331596800
568453                       0      5  1338422400

                                 Summary  \
0                    Good Quality Dog Food
1                        Not as Advertised
2                    "Delight" says it all
3                           Cough Medicine
4                              Great taffy
…                                      …
568449                 Will not do without
568450                       disappointed
568451            Perfect for our maltipoo
568452  Favorite Training and reward treat
568453                       Great Honey

                                                    Text  \
0       I have bought several of the Vitality canned d…
1       Product arrived labeled as Jumbo Salted Peanut…
2       This is a confection that has been around a fe…
3       If you are looking for the secret ingredient i…
4       Great taffy at a great price.  There was a wid…
…                                                     …
568449  Great for sesame chicken..this is a good if no…
568450  I'm disappointed with the flavor. The chocolat…
568451  These stars are small, so you can give 10-15 o…
568452  These are the BEST treats for training and rew…
568453  I am very satisfied ,product is as advertised,…

                                                Text_pre
0       bought several vitality canned dog food produc…
1       product arrived labeled jumbo salted peanut  p…
2       confection around century  light  pillowy citr…
```

```
3       looking secret ingredient robitussin believe f…
4       great taffy great price  wide assortment yummy…
…                                                     …
568449  great sesame chicken  good better resturants e…
568450  m disappointed flavor  chocolate note especial…
568451  star small  give  one training session  tried …
568452  best treat training rewarding dog good groomin…
568453  satisfied  product advertised  use cereal  raw…

[568427 rows x 9 columns]
```

[14]:
```
df.Score.value_counts()
```

[14]:
```
5    363122
4     80655
1     52268
3     42638
2     29744
Name: Score, dtype: int64
```

[15]:
```
count_5, count_4, count_1, count_3, count_2 = df.Score.value_counts()
avg=int((count_5 + count_4 + count_1 + count_3 + count_2)/5)
```

0.0.3 Balancing dataset

[16]:
```
df_class_1 = df[df['Score']==1]
df_class_2 = df[df['Score']==2]
df_class_3 = df[df['Score']==3]
df_class_4 = df[df['Score']==4]
df_class_5 = df[df['Score']==5]
```

[17]:
```
class_1 = df_class_1.sample(avg, replace=True)
class_2 = df_class_2.sample(avg, replace=True)
class_3 = df_class_3.sample(avg, replace=True)
class_4 = df_class_4.sample(avg, replace=True)
class_5 = df_class_5.sample(avg)
```

[18]:
```
df_balanced= pd.concat([class_1,class_2,class_3,class_4,class_5], axis=0)
df_balanced
```

[18]:
```
         ProductId          UserId  HelpfulnessNumerator  \
206779  B008572JIG  A3GY8NYC3K6JF3                     2
118274  B007RTR89S  A3NWMK532EV42W                     0
314334  B001F9AOR4  A3O7WPXFZMBIKZ                     3
239606  B001E6K6B2  A1K6NAM2EEQ5RR                     1
215663  B0000DGFA9  A1HD2OIUGRGMB3                     5
…              …              …                     …
```

```
123158  B000CQBZOW  A1HZ4TUPLQ1ZEA                    5
554168  B001SBAA3M  A3R1D7H3PIFONU                    0
58451   B001ULH7P4   A5D2SF1EMAAB2                   22
223793  B004VLVIFU  A1LJHCU8O09QCV                    2
505035  B0048IOCFO   AWTXD6UIIN6K4                    0

        HelpfulnessDenominator  Score        Time  \
206779                       4      1  1316390400
118274                       1      1  1338854400
314334                       4      1  1322956800
239606                       2      1  1340928000
215663                       7      1  1166227200
…                          …    …          …
123158                       5      5  1225929600
554168                       0      5  1237161600
58451                       26      5  1264809600
223793                       2      5  1341360000
505035                       0      5  1337299200

                                                Summary  \
206779                              NEVER! NEVER! NEVER!
118274           cheap smell, itchy scalp, animal-tested
314334                             Manufactured in China
239606                             Product of Mexico ???
215663                                    Sloppy Packing
…                                                     …
123158                                 Herbal tea drinker
554168                    Orange, Passionfruit, & Jasmine
58451   Tastes Great but concerned about manufacturing…
223793                                       Nice product
505035                                  Adolph's Marinade

                                                    Text  \
206779  NEVER buy Pedigree. I made a huge mistake tryi…
118274  I tried this shampoo (paired with the strength…
314334  The bag says that the product was manufactured…
239606  Right on the box it says product of Mexico… …
215663  I have never seen such a sloppy and unsuitable…
…                                                     …
123158  Good, full bodied, cup of tea that does not ge…
554168  Another winner from the Lipton Green Tea lineu…
58451   To be completely and 100% fair, I have tried t…
223793  After reading the book "Muscle foods" I set ou…
505035  I have used this product for at least 20 years…

                                                Text_pre
206779  never buy pedigree  made huge mistake trying  …
```

```
118274  tried shampoo  paired strength conditioner  we…
314334  bag say product manufactured china  warning br…
239606  right box say product mexico  nothing race cou…
215663  never seen sloppy unsuitable packing job amazo…
…                                                    …
123158  good  full bodied  cup tea get bitter  fruity …
554168  another winner lipton green tea lineup  tea sw…
58451   completely   fair  tried product enjoyed  part…
223793  reading book  muscle food  set buy product lis…
505035  used product least  year  longer find local gr…

[568425 rows x 9 columns]
```

[19]:
```
df_balanced.Score.value_counts()
```

[19]:
```
1    113685
2    113685
3    113685
4    113685
5    113685
Name: Score, dtype: int64
```

1 Model Preparation

[20]:
```
# removing all words other than nouns from tags feature

def pos(line):
    tempCol = ''
    posTag = nltk.pos_tag(line.split(' '))
    for (word, pos) in posTag:
        if(pos.startswith("NN")):
            tempCol += ' ' + word.strip()
    return tempCol

df_balanced['TextNN'] = df_balanced['Text'].apply(lambda x: pos(x))
df_balanced['TextNN'] = df_balanced['Text'].swifter.apply(lambda l:␣
 ↪pre_processing(l))
df_balanced.head()
```

```
Pandas Apply:   0%|          | 0/568425 [00:00<?, ?it/s]
```

[20]:
```
         ProductId          UserId  HelpfulnessNumerator  \
206779  B008572JIG  A3GY8NYC3K6JF3                     2
118274  B007RTR89S  A3NWMK532EV42W                     0
314334  B001F9AOR4  A3O7WPXFZMBIKZ                     3
239606  B001E6K6B2  A1K6NAM2EEQ5RR                     1
```

```
215663  B0000DGFA9  A1HD2OIUGRGMB3                     5

        HelpfulnessDenominator  Score        Time  \
206779                       4      1  1316390400
118274                       1      1  1338854400
314334                       4      1  1322956800
239606                       2      1  1340928000
215663                       7      1  1166227200

                                     Summary  \
206779                    NEVER! NEVER! NEVER!
118274  cheap smell, itchy scalp, animal-tested
314334                   Manufactured in China
239606                   Product of Mexico ???
215663                          Sloppy Packing

                                                    Text  \
206779  NEVER buy Pedigree. I made a huge mistake tryi…
118274  I tried this shampoo (paired with the strength…
314334  The bag says that the product was manufactured…
239606  Right on the box it says product of Mexico… …
215663  I have never seen such a sloppy and unsuitable…

                                                Text_pre  \
206779  never buy pedigree  made huge mistake trying  …
118274  tried shampoo  paired strength conditioner  we…
314334  bag say product manufactured china  warning br…
239606  right box say product mexico  nothing race cou…
215663  never seen sloppy unsuitable packing job amazo…

                                                  TextNN
206779  never buy pedigree  made huge mistake trying  …
118274  tried shampoo  paired strength conditioner  we…
314334  bag say product manufactured china  warning br…
239606  right box say product mexico  nothing race cou…
215663  never seen sloppy unsuitable packing job amazo…
```

[21]:
```
from sklearn.model_selection import train_test_split

x_text=df_balanced['TextNN']
y_text=df_balanced['Score']

X_train_text, X_test_text, y_train_text, y_test_text = train_test_split(x_text,␣
 ↪y_text)
```

1.1 Random Forest

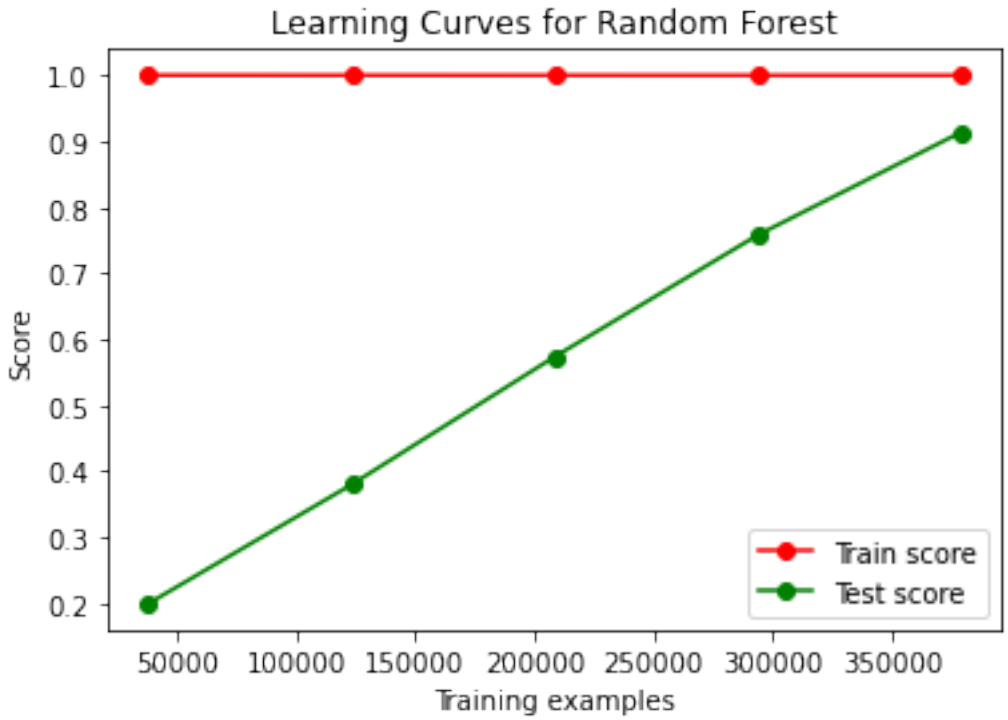
1.1.1 For Text

[18]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer

rf_clf = Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('Random_Forest',␣
 ↪RandomForestClassifier(n_jobs = -1, n_estimators= 105))])
rf_clf.fit(X_train_text, y_train_text)

y_pred_rf_text = rf_clf.predict(X_test_text)

score=rf_clf.score(X_test_text, y_test_text)

print("Accuracy for Random Forest:",score)

print(classification_report(y_test_text, y_pred_rf_text))
```

```
Accuracy for Random Forest: 0.9269353374569866
              precision    recall  f1-score   support

           1       0.95      0.96      0.95     28357
           2       0.99      0.97      0.98     28381
           3       0.97      0.93      0.95     28287
           4       0.93      0.84      0.88     28586
           5       0.81      0.94      0.87     28496

    accuracy                           0.93    142107
   macro avg       0.93      0.93      0.93    142107
weighted avg       0.93      0.93      0.93    142107
```

[21]:
```
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import numpy as np

train_sizes, train_scores, test_scores = learning_curve(rf_clf, x_text, y_text,␣
 ↪cv = 3, n_jobs=-1, train_sizes=np.linspace(0.1, 1.0, 5))
train_scores_mean=np.mean(train_scores, axis=1)
test_scores_mean=np.mean(test_scores, axis=1)

#Labeling the Axis and Table
plt.title('Learning Curves for Random Forest')
plt.xlabel('Training examples')
```

```
plt.ylabel('Score')

plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Train score')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Test score')

plt.legend(loc='best')

plt.show()
```



1.2 K Nearest Neighbour

1.2.1 For Text:

[36]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from sklearn.feature_extraction.text import TfidfVectorizer


clf = Pipeline([('vectorizer_tfidf',TfidfVectorizer(max_features=5000)),('KNN',␣
 ↪KNeighborsClassifier())])

clf.fit(X_train_text, y_train_text)
y_pred_knn_text = clf.predict(X_test_text)
```

```
print(classification_report(y_test_text, y_pred_knn_text))
```

```
              precision    recall  f1-score   support

           1       0.82      0.74      0.78     28565
           2       0.80      0.86      0.83     28192
           3       0.76      0.74      0.75     28550
           4       0.75      0.56      0.64     28369
           5       0.54      0.71      0.61     28431

    accuracy                           0.72    142107
   macro avg       0.74      0.72      0.72    142107
weighted avg       0.74      0.72      0.72    142107
```

1.2.2 For Summary

[37]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from sklearn.feature_extraction.text import TfidfVectorizer


clf = Pipeline([('vectorizer_tfidf',TfidfVectorizer(max_features=5000)),('KNN',␣
 ↪KNeighborsClassifier())])

clf.fit(X_train_text, y_train_text)
y_pred_knn_sum = clf.predict(X_test_text)


print(classification_report(y_test_text, y_pred_knn_sum))
```

```
              precision    recall  f1-score   support

           1       0.66      0.74      0.70     28458
           2       0.61      0.69      0.65     28430
           3       0.64      0.65      0.64     28280
           4       0.62      0.54      0.58     28578
           5       0.64      0.54      0.58     28361

    accuracy                           0.63    142107
   macro avg       0.63      0.63      0.63    142107
weighted avg       0.63      0.63      0.63    142107
```

1.2.3 For Summary

[40]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

rf_clf =␣
 ↪Pipeline([('vectorizer_tfidf',TfidfVectorizer(max_features=5000)),('Random_Forest',␣
 ↪RandomForestClassifier())])
rf_clf.fit(X_train_text, y_train_text)

y_pred_rf_sum = rf_clf.predict(X_test_text)

print("Accuracy for Random Forest:",rf_clf.score(X_test_text, y_test_text))

print(classification_report(y_test_text, y_pred_rf_sum))
```

```
Accuracy for Random Forest: 0.7535448640812908
              precision    recall  f1-score   support

           1       0.81      0.80      0.80     28458
           2       0.72      0.80      0.76     28430
           3       0.80      0.75      0.78     28280
           4       0.72      0.69      0.70     28578
           5       0.73      0.72      0.73     28361

    accuracy                           0.75    142107
   macro avg       0.75      0.75      0.75    142107
weighted avg       0.75      0.75      0.75    142107
```

1.3 Hyperparameter Tuning Random Forest

1.3.1 For Text

[ ]:
```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import RandomizedSearchCV

tfidf_text = TfidfVectorizer(max_features=5000).fit(X_train_text.astype('U').
 ↪values)
text_train = pd.DataFrame(tfidf_text.transform(X_train_text.astype('U').values).
 ↪todense(),columns=tfidf_text.get_feature_names_out())
text_test = pd.DataFrame(tfidf_text.transform(X_test_text.astype('U').values).
 ↪todense(),columns=tfidf_text.get_feature_names_out())

'''
X_train_text = TfidfVectorizer(max_features=5000).fit_transform(X_train_text)
y_train_text = TfidfVectorizer(max_features=5000).fit_transform(y_train_text)
```

```
X_test_text = TfidfVectorizer(max_features=5000).fit_transform(X_test_text)
y_test_text = TfidfVectorizer(max_features=5000).fit_transform(y_test_text)
'''

n_estimators = [int(x) for x in np.linspace(start = 5 , stop = 15, num = 10)] #␣
 ↪returns 10 numbers

max_features = ['auto', 'log2']

max_depth = [int(x) for x in np.linspace(5, 10, num = 2)]

max_depth.append(None)

bootstrap = [True, False]

r_grid = {'Random_Forest__n_estimators': n_estimators,

               'Random_Forest__max_features': max_features,

               'Random_Forest__max_depth': max_depth,

               'Random_Forest__bootstrap': bootstrap}

print(r_grid)

rfc_random = RandomizedSearchCV(estimator=rf_clf, param_distributions=r_grid,␣
 ↪n_iter = 20, scoring='neg_mean_absolute_error', cv = 3, verbose=2,␣
 ↪random_state=42, n_jobs=-1, return_train_score=True)

rfc_random.fit(X_train_text,y_train_text);

#Hyperparameter Tuning
print(rfc_random.best_params_)
```

```
{'Random_Forest__n_estimators': [5, 6, 7, 8, 9, 10, 11, 12, 13, 15],
'Random_Forest__max_features': ['auto', 'log2'], 'Random_Forest__max_depth': [5,
10, None], 'Random_Forest__bootstrap': [True, False]}
Fitting 3 folds for each of 20 candidates, totalling 60 fits
```

1.4 Logistic Regression

1.4.1 For Text

[28]:
```
#same for all reference.
from sklearn.linear_model import LogisticRegression
#logreg = LogisticRegression(fit_intercept = True, C = 1e12, solver =␣
 ↪'liblinear')
```

```
# Fit the model to our X and y training sets
#logreg.fit(X_train, y_train)

clf_logi = Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('logreg',␣
 ↪LogisticRegression(fit_intercept = True, C = 1e12, solver = 'liblinear'))])

clf_logi.fit(X_train_text, y_train_text)
```

[28]:
```
Pipeline(steps=[('vectorizer_tfidf', TfidfVectorizer()),
                ('logreg',
                 LogisticRegression(C=1000000000000.0, solver='liblinear'))])
```

[29]:
```
from sklearn.metrics import accuracy_score
y_pred_logi = clf_logi.predict(X_test_text)

print('Accuracy',accuracy_score(y_test_text, y_pred_logi))
print(classification_report(y_test_text, y_pred_logi))
```

```
Accuracy 0.7558248362149648
              precision    recall  f1-score   support

           1       0.82      0.85      0.83     28357
           2       0.78      0.81      0.79     28553
           3       0.74      0.76      0.75     28512
           4       0.70      0.67      0.68     28266
           5       0.74      0.69      0.71     28419

    accuracy                           0.76    142107
   macro avg       0.75      0.76      0.75    142107
weighted avg       0.75      0.76      0.75    142107
```

1.4.2 For Summary

[43]:
```
#same for all reference.
from sklearn.linear_model import LogisticRegression
#logreg = LogisticRegression(fit_intercept = True, C = 1e12, solver =␣
 ↪'liblinear')

# Fit the model to our X and y training sets
#logreg.fit(X_train, y_train)

clf_logi_sum = Pipeline([('vectorizer_tfidf',TfidfVectorizer()),('logreg',␣
 ↪LogisticRegression(fit_intercept = True, C = 1e12, solver = 'liblinear'))])

clf_logi_sum.fit(X_train_text, y_train_text)
```

```
from sklearn.metrics import accuracy_score
y_pred_logi_sum = clf_logi_sum.predict(X_test_text)

print('Accuracy',accuracy_score(y_test_text, y_pred_logi_sum))
print(classification_report(y_test_text, y_pred_logi_sum))
```

```
Accuracy 0.37555503951248004
              precision    recall  f1-score   support

           1       0.55      0.41      0.47     28357
           2       0.38      0.15      0.21     28553
           3       0.32      0.34      0.33     28512
           4       0.29      0.47      0.36     28266
           5       0.43      0.52      0.47     28419

    accuracy                           0.38    142107
   macro avg       0.39      0.38      0.37    142107
weighted avg       0.39      0.38      0.37    142107
```

[ ]:
```
### References

[1] - https://dal.brightspace.com/d2l/le/content/232269/viewContent/3256445/View

[2] - https://pandas.pydata.org/docs/reference/general_functions.html

[3] - https://miamioh.instructure.com/courses/38817/pages/data-cleaning

[4] - https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

[5] - https://scikit-learn.org/stable/modules/generated/sklearn.
↪feature_extraction.text.TfidfVectorizer.html

[6] - https://www.nltk.org/_modules/nltk/stem/wordnet.html

[7] - https://www.geeksforgeeks.org/bag-of-words-bow-model-in-nlp/

[8] - https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.
↪html
```