File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Load the Yelp data

In [2]:
```
df = pd.read_csv(r"D:\LIUZHICHENG\Udemy\Machine Learning\8 Real World Projects\Natural Language Processing - Yelp Reviews
df = df.drop(columns=["business_id", "date", "review_id", "type", "user_id"])
df.head()
```

Out[2]:

	stars	text	cool	useful	funny
0	5	My wife took me here on my birthday for breakf...	2	5	0
1	5	I have no idea why some people give bad review...	0	0	0
2	4	love the gyro plate. Rice is so good and I als...	0	1	0
3	5	Rosie, Dakota, and I LOVE Chaparral Dog Park!!...	1	2	0
4	5	General Manager Scott Petello is a good egg!!!...	0	0	0

In [3]:
```
df["length"] = df["text"].apply(func=len)
df
```

Out[3]:

	stars	text	cool	useful	funny	length
0	5	My wife took me here on my birthday for breakf...	2	5	0	889
1	5	I have no idea why some people give bad review...	0	0	0	1345
2	4	love the gyro plate. Rice is so good and I als...	0	1	0	76
3	5	Rosie, Dakota, and I LOVE Chaparral Dog Park!!...	1	2	0	419
4	5	General Manager Scott Petello is a good egg!!!...	0	0	0	469
...	...	...	...	...	...	...
9995	3	First visit...Had lunch here today - used my G...	1	2	0	668
9996	4	Should be called house of deliciousness!\n\nI ...	0	0	0	881
9997	4	I recently visited Olive and Ivy for business ...	0	0	0	1425
9998	2	My nephew just moved to Scottsdale recently so...	0	0	0	880
9999	5	4-5 locations.. all 4.5 star average.. I think...	0	0	0	461

10000 rows × 6 columns

EDA [...]

Text Preprocessing

In [21]:
```
%%time

import string
import re
import nltk
from nltk.corpus import stopwords

def remove_punctuation(input_text):

    """ Remove punctuations like '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~' """

    #print("in remove_punctuation\n",input_text)
    # Make translation table
    input_text = str(input_text) # avoid thedange of being a series object
    punct = string.punctuation
    trantab = str.maketrans(punct, len(punct)*' ')  # Every punctuation symbol will be replaced by a space
    return input_text.translate(trantab).encode('ascii', 'ignore').decode('utf8') # -> Final kick to clean up :)


def remove_digits(input_text):

    """ Remove numerical digits ranging from 0-9 """
    #print("in remove_digits\n",input_text)
    return re.sub('\d+', '', input_text)


def to_lower(input_text):

    """ String handling, returns the lowercased strings from the given string """
    #print("in to_lower\n",input_text)
    return input_text.lower()


def remove_stopwords(input_text):

    """ Remove the low-level information from our text in order to give more focus to the important information """
   #print("in remove_stopwords\n",input_text)
    stopwords_list = stopwords.words('english')
    newStopWords = ['citi']
    stopwords_list.extend(newStopWords)

    # Some words which might indicate a certain sentiment are kept via a whitelist
    #whitelist = ["n't", "not", "no"]
    whitelist = ["n't"]
    words = input_text.split()
    clean_words = [word for word in words if (word not in stopwords_list or word in whitelist) and len(word) > 2]
    return " ".join(clean_words) # List -> string
```

```
48
49
50  def expandShortsForms(input_text):
51      #print("in expandShortsForms\n",input_text)
52      return input_text.replace("can't", "can not").replace("won't", "will not")
53
54
55  def lemmatize(input_text):
56
57      """ Return the base or dictionary form of a word, lemma """
58      #print("in lemmatize\n",input_text)
59      outtext= ""
60      # Lemmatize
61      from nltk.stem import WordNetLemmatizer
62      from nltk import pos_tag, word_tokenize, wordnet
63      from nltk.corpus.reader.wordnet import WordNetError
64      lemmatizer = WordNetLemmatizer()
65
66      input_text = input_text.split()
67      for word in input_text:
68          # Get the single character pos constant from pos_tag like this:
69          pos_label = (pos_tag(word_tokenize(word))[0][1][0]).lower()
70
71          # pos_refs = {'n': ['NN', 'NNS', 'NNP', 'NNPS'],
72          #             'v': ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'],
73          #             'r': ['RB', 'RBR', 'RBS'],
74          #             'a': ['JJ', 'JJR', 'JJS']}
75
76          if pos_label == 'j': pos_label = 'a'    # 'j' <--> 'a' reassignment
77
78          if pos_label in ['r']:  # For adverbs it's a bit different
79              try:
80                  if len(wordnet.wordnet.synset(word+'.r.1').lemmas()[0].pertainyms()) > 0:
81                      outtext = outtext + ' ' +  (wordnet.wordnet.synset(word+'.r.1').lemmas()[0].pertainyms()[0].name())
82              except WordNetError:
83                  pass
84              outtext = outtext + ' ' + word # To keep the word in the list
85          elif pos_label in ['a', 's', 'v']: # For adjectives and verbs
86              outtext = outtext +' ' + (lemmatizer.lemmatize(word, pos=pos_label))
87          else:   # For nouns and everything else as it is the default kwarg
88              outtext = outtext +' ' + (lemmatizer.lemmatize(word))
89
90      return outtext
91
92
93  def execute_funcs(input_text, *args):
94      funcs = list(args)
95      for func in funcs:
96          input_text = func(input_text)
97      return input_text
98
99
100 def apply_funcs(input_text):
101     clean_X = execute_funcs(input_text, to_lower, remove_punctuation, remove_digits,
102                             remove_stopwords,
103                             expandShortsForms,
104 #                             lemmatize
105                            )
106     return clean_X
```

Wall time: 0 ns

Pipeline & ColumnTransformer

In [22]:

```
1   %%time
2
3   import warnings
4   warnings.filterwarnings('ignore')
5
6   df["clean_text"] = df["text"].apply(func=apply_funcs)
7   df["length"] = df["clean_text"].apply(func=len)
8
9   feature_columns = ['cool', 'useful', 'funny', 'clean_text', 'length']
10
11  X = df[feature_columns]
12  y = df["stars"]
13
14  from sklearn.model_selection import train_test_split
15
16  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)
17
18  from sklearn.preprocessing import LabelEncoder
19
20  enc = LabelEncoder()
21  y_train = enc.fit_transform(y=y_train)
22  y_test = enc.transform(y=y_test)
23
24  from sklearn.compose import ColumnTransformer
25  from sklearn.feature_extraction.text import TfidfVectorizer
26  from sklearn.preprocessing import OneHotEncoder
27  from sklearn.preprocessing import StandardScaler, RobustScaler, Normalizer
28
29  # Whenever the transformer expects a 1D array as input, the columns were specified as a string ("xxx").
30  # For the transformers which expects 2D data, we need to specify the column as a list of strings (["xxx"]).
31  ct = ColumnTransformer(transformers=[
32      ("TfidfVectorizer", TfidfVectorizer(), ("clean_text")),
33      ("OneHotEncoder", OneHotEncoder(handle_unknown='ignore'), (["cool", "useful", "funny"])),
34      ('Normalizer', Normalizer(), (["length"])),
35      ],
36      n_jobs=-1)
37
38  from sklearn.pipeline import Pipeline
39  from sklearn.naive_bayes import MultinomialNB
40
41  clf = Pipeline(steps=[
42      ("ColumnTransformer", ct),
43      ("MultinomialNB", MultinomialNB())
44  ])
45
46  clf.fit(X_train, y_train)
47
```

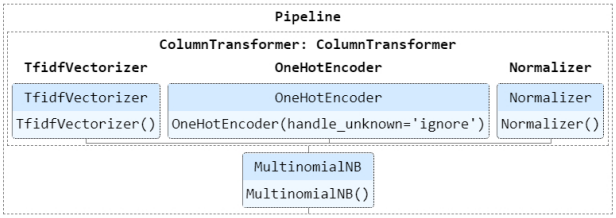
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

predictions = clf.predict(X_test)
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       150
           1       0.00      0.00      0.00       185
           2       0.00      0.00      0.00       292
           3       0.38      0.87      0.53       705
           4       0.62      0.36      0.45       668

    accuracy                           0.43      2000
   macro avg       0.20      0.25      0.20      2000
weighted avg       0.34      0.43      0.34      2000

0.4265
[[  0   0   0 125  25]
 [  0   0   0 174  11]
 [  0   0   0 271  21]
 [  0   0   0 613  92]
 [  0   0   0 428 240]]
Wall time: 10.9 s
```

In [23]:

```
from sklearn import set_config
set_config(display="diagram")

clf
```

Out[23]:



make_pipeline & make_column_transformer

In [24]:

```
%%time

import warnings
warnings.filterwarnings('ignore')

df["clean_text"] = df["text"].apply(func=apply_funcs)
df["length"] = df["clean_text"].apply(func=len)

feature_columns = ['cool', 'useful', 'funny', 'clean_text', 'length']

X = df[feature_columns]
y = df["stars"]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)

from sklearn.preprocessing import LabelEncoder

enc = LabelEncoder()
y_train = enc.fit_transform(y=y_train)
y_test = enc.transform(y=y_test)

from sklearn.compose import make_column_transformer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler, RobustScaler, Normalizer

# This is a shorthand for the ColumnTransformer constructor; it does not require, and does not permit, naming the
# transformers.
ct = make_column_transformer(
    (CountVectorizer(), ("clean_text")),
    (OneHotEncoder(handle_unknown='ignore'), (["cool", "useful", "funny"])),
    (StandardScaler(), (["length"])),
    n_jobs=-1)

from sklearn.pipeline import make_pipeline
from lightgbm import LGBMClassifier
from sklearn.naive_bayes import MultinomialNB

# This is a shorthand for the Pipeline constructor; it does not require, and does not permit, naming the estimators.
# Instead, their names will be set to the lowercase of their types automatically.
clf = make_pipeline(ct, LGBMClassifier())

clf.fit(X_train, y_train)

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

predictions = clf.predict(X_test)
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

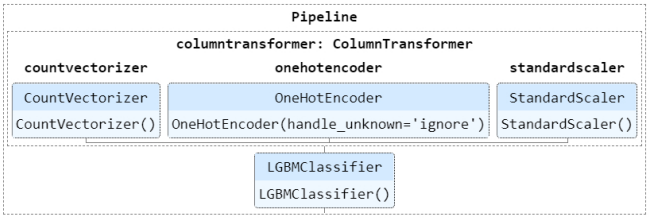
```
              precision    recall  f1-score   support

           0       0.62      0.49      0.54       150
           1       0.46      0.26      0.34       185
           2       0.39      0.22      0.28       292
           3       0.49      0.60      0.54       705
           4       0.59      0.65      0.62       668

    accuracy                           0.52      2000
   macro avg       0.51      0.45      0.46      2000
weighted avg       0.52      0.52      0.51      2000

0.524
```

```
[[ 73  24  14  21  18]
 [ 27  49  37  55  17]
 [  5  20  64 157  46]
 [  7  11  42 425 220]
 [  6   2   9 214 437]]
Wall time: 11.2 s
```

In [25]:

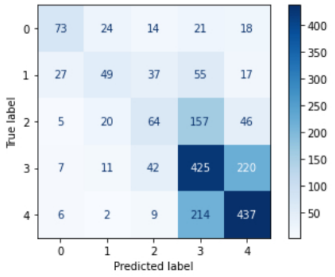
```
from sklearn import set_config
set_config(display="diagram")

clf
```

Out[25]:



In [26]:

```
from sklearn.metrics import plot_confusion_matrix

display = plot_confusion_matrix(estimator=clf, X=X_test, y_true=y_test, cmap="Blues", values_format='.3g')
```



In [27]:

```
display.confusion_matrix
```

Out[27]:

```
array([[ 73,  24,  14,  21,  18],
       [ 27,  49,  37,  55,  17],
       [  5,  20,  64, 157,  46],
       [  7,  11,  42, 425, 220],
       [  6,   2,   9, 214, 437]], dtype=int64)
```