Assignment #4 - Luke Schwenke

Create classification model, predicting the outcome of food safety inspection based on the inspectors' comments

In [ ]:
```
import pandas as pd
import requests
import re
import numpy as np
from pandarallel import pandarallel
from textblob import TextBlob
import sklearn
from textblob.sentiments import NaiveBayesAnalyzer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, HashingVectoriz
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix, accurac
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn import metrics
from sklearn.pipeline import make_pipeline
import numpy as np
import eli5
```

In [ ]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
import multiprocessing
num_processors = multiprocessing.cpu_count()
print(f'Available CPUs: {num_processors}')
```

```
Available CPUs: 8
```

In [ ]:
```
pandarallel.initialize(nb_workers=num_processors-1, use_memory_fs=False)
```

```
INFO: Pandarallel will run on 7 workers.
INFO: Pandarallel will use standard multiprocessing data transfer (pipe) to
transfer data between the main process and workers.
```

In [ ]:
```
%%time

# Define the API endpoint and parameters
url = "https://data.cityofchicago.org/resource/cwig-ma7x.json"

# Fetch the total count of records
def get_total_count(url):
    params = {
        "$select": "count(*)"
    }
    response = requests.get(url, params=params)
    data = response.json()
    return int(data[0]['count'])

total_count = get_total_count(url)
print(f"Total number of records available: {total_count:,.0f}")
```

```
Total number of records available: 266,602
CPU times: user 30 ms, sys: 3.48 ms, total: 33.5 ms
Wall time: 784 ms
```

In [ ]:
```
%%time

# Fetch data and load it into a pandas DataFrame
def fetch_data(url, params):
    response = requests.get(url, params=params)
    data = response.json()
    return pd.DataFrame(data)

# Set the limit parameter equal to the number of available records
params = {
    "$limit": total_count
}

# Fetch the data and load it into a DataFrame
df = fetch_data(url, params)

print(f'Number of records retrieved: {df.shape[0]:,.0f}')
```

```
Number of records retrieved: 266,602
CPU times: user 5.3 s, sys: 554 ms, total: 5.85 s
Wall time: 25.8 s
```

In [ ]:
```
df.head(1)
```

Out[ ]:

	inspection_id	dba_name	aka_name	license_	facility_type	risk	address	city
0	2588404	SAMI BOY 2	SAMI BOY 2	2951502	Grocery Store	Risk 2 (Medium)	3730 W DIVISION ST	CHICAGO

1 rows × 22 columns

Extract free-form text comments from inspectors

In [ ]:
```
#df_clean = df[df['results'] == 'Fail']

# Drop rows with NaN in the 'violations' column
df_clean = df.dropna(subset=['violations'])
```

In [ ]:
```
comments_pattern = r'Comments: (.*?)(?:\||$)'
df_clean['text_comments'] = df_clean['violations'].apply(lambda x: re.findal
```

In [ ]:
```
print("Here is a sample comment:", df_clean['text_comments'][0])
```

```
Here is a sample comment: ['NO WRITTEN EMPLOYEE HEALTH POLICY ON SITE FOR AL
L EMPLOYEES. MUST PROVIDE. PRIORITY FOUNDATION VIOLATION. NO CITATION ISSUE
D. ', 'NOTED THE REAR PREP/DISH HAND WASHING SINK NOT ACCESSIBLE FOR USE. SI
NK COMPLETELY BLOCKED BY A BOX FREEZER AND MANY STORED KITCHEN ITEMS ON THE
FLOOR UNDER AND AROUND THE SINK. MUST MAKE SINK ACCESSIBLE AND REMOVE ALL ST
ORED ITEMS UNDER AND AROUND THE SINK. PRIORITY FOUNDATION VIOLATION 7-38-03
0(C) CITATION ISSUED. ', 'ALL BULK FOODS NOT STORED IN THE ORIGINAL CONTAINE
R MUST BE LABELED WITH THE PRODUCT COMMON NAME. ', 'OBSERVED THE FOLLOWING R
AT DROPPINGS IN THE FOLLOWING AREAS: APPROXIMATELY 50 DROPPINGS ALONG THE BA
SEBOARD BEHIND THE DINING ROOM LEATHER SOFA, 15 TO 20 DROPPINGS THROUGHOUT T
HE REAR STORAGE AREA NEAR THE CUSTOMER WASHROOM, 10 TO 15 DROPPINGS BEHIND B
OX FREEZER, COOLER AND COOKS LINE EQUIPMENT, 5 TO 10 DROPPINGS UNDER THE REA
R THREE COMPARTMENT SINK AND HAND SINK. INSTRUCTED TO REMOVE ALL DROPPINGS,
SANITIZE ALL AFFECTED AREAS. PRIORITY FOUNDATION VIOLATION 7-38-020(A) CITAT
ION ISSUED. ', 'FRONT EXIT/ENTRANCE DOOR WITH OPEN 1/4 INCH GAP ALONG THE BO
TTOM THRESHOLD AND MISSING RODENT PROOF STRIP. MUST PROVIDE A TIGHT FITTING
SEAL ON ALL AREAS OF THE DOOR. ', 'ALL POTS, PANS AND COOKING EQUIPMENT, BUL
K FOOD BUCKETS MUST BE STORED ELEVATED FROM THE FLOORS. ', 'NOTED KITCHEN CU
TTING BOARD IN EXTREMELY POOR CONDITION WITH DEEP BLACKENED GROOVES. MUST RE
PLACE. ', 'MUST NOT USE GROCERY BAGS AS FOOD CONTAINERS THROUGHOUT ALL FREEZ
ERS AND COOLERS. ', 'MUST NOT USE RAW WOOD AS LINER UNDER BULK FOOD BUCKETS
AND UNDER KITCHEN BOX FREEZERS.----MUST NOT USE TIN FOIL AS LINER FOR SHELVI
NG IN THE KITCHEN. MUST BE SMOOTH, CLEANABLE AND NON-ABSORBENT. ', 'INTERIOR
OF THE OVEN AND SOME COOLERS DIRTY WITH FOOD DEBRIS, FOOD CONTACT SHELVING T
HROUGHOUT THE KITCHEN NOTED DIRTY. MUST CLEAN AND MAINTAIN ALL. ', 'NOTED A
RUBBER HOSE ATTACHED TO THE UTILITY SERVICE SINK AND SINK WITHOUT A BACKFLOW
PREVENTION DEVICE. MUST PROVIDE A BACKFLOW PREVENTION DEVICE FOR THE SINK.
', 'CRACKED/DAMAGED/MISSING FLOOR TILES ON THE FLOOR AT THE ENTRANCE TO THE
REAR PREP/DISH AREA. MUST REPLACE. ', 'NOTED DIRTY FLOORS IN THE FOLLOWING A
REAS WITH GREASE AND DEBRIS: UNDER THE THREE COMPARTMENT SINK, AROUND AND BE
HIND THE GREASE TRAP, BEHIND ALL BOX FREEZERS AND KITCHEN COOLERS, UNDER COO
KS LINE EQUIPMENT. MUST CLEAN AND MAINTAIN ALL.----RAW WOODEN WALL ON THE KI
TCHEN WEST WALL MUST BE PAINTED/SEALED.---- WATER DAMAGED CEILING TILES THRO
UGHOUT THE KITCHEN MUST BE REPLACED.----NOTED TWO HOLES IN THE WALL BASEBOAR
D BEHIND THE DINING ROOM LEATHER SOFA. INSTRUCTED TO SEAL ALL OPEN NOTED HOL
ES. ', 'ALL UNNECESSARY ITEMS STORED IN THE FOLLOWING AREAS MUST BE REMOVED
FROM THE PREMISES TO PREVENT PEST HARBORING AND ALL OTHER NEEDED ITEMS MUST
BE STORED ELEVATED AND ORGANIZED: THROUGHOUT THE FRONT CASHIER AREA, REAR EX
IT DOOR AREA AT CUSTOMER WASHROOM, REAR PREP/DISH ROOM.-----DIRTY, UNUSED TA
LL REACH-IN COOLER AT THE FRONT CASHIER AREA MUST BE REMOVED FROM THE PREMIS
ES. ', 'MUST PROVIDE FOOD HANDLER TRAINING FOR ALL REQUIRED EMPLOYEES.']
```

In [ ]:
```
def list_to_string(text_list):
    return ' '.join(text_list)

# Apply the function to each element in the 'text_comments' column
df_clean['text_comments_single_string'] = df_clean['text_comments'].apply(li
```

In [ ]:
```
df_clean['text_comments_single_string'] = df_clean['text_comments_single_str
```

In [ ]:
```
print("Here is a CLEANED sample comment:", df_clean['text_comments_single_st
```

```
Here is a CLEANED sample comment: observed no verifiable employee health pol
icy on site at time of inspection left template and instructed to maintain c
opies of verifiable health policy signed by all employees on site at all tim
es priority foundation violation 7 38 010 no citation issued observed the fr
ont exit door not completely rodent proofed as required must completely rode
nt proof door by sealing 1 4 gap at the bottom of the door observed washroom
door not self closing instructed manager to repair self closing device on wa
shroom door observed soiled mop heads not properly stored instructed manager
to hang mop heads to prevent insect breeding
```

Alternatively, split into individual comments which will increase the number of rows / samples

I will use this as the *X* variable

In [ ]:
```
df_exploded = df_clean.explode('text_comments').reset_index()
```

In [ ]:
```
df_exploded['text_comments'][0]
```

Out[ ]:
```
'OBSERVED NO VERIFIABLE EMPLOYEE HEALTH POLICY ON SITE AT TIME OF INSPECTIO
N. LEFT TEMPLATE AND INSTRUCTED TO MAINTAIN COPIES OF VERIFIABLE HEALTH POLI
CY SIGNED BY ALL EMPLOYEES ON SITE AT ALL TIMES. PRIORITY FOUNDATION VIOLATI
ON 7-38-010. NO CITATION ISSUED '
```

Model Preparation for Multinomial Classification

In [ ]:
```
print("The different possible inspection outcomes are as follows:\n", list(d
```

```
The different possible inspection outcomes are as follows:
 ['Pass', 'Fail', 'Pass w/ Conditions', 'No Entry', 'Not Ready', 'Out of Bus
iness']
```

I will encode the labelled *y* variable as:

- 1 = Fail
- 2 = Pass
- 3 = Pass w/ Conditions
- 4 = No Entry
- 5 = Not Ready
- 6 = Out of Business

In [ ]:
```
encoding = {
    'Fail': 1,
    'Pass': 2,
    'Pass w/ Conditions': 3,
    'No Entry': 4,
    'Not Ready': 5,
    'Out of Business': 6
}

df_exploded['results_labels'] = df_exploded['results'].map(encoding)
```

In [ ]:
```
# define X and y
X = df_exploded['text_comments']
y = df_exploded['results_labels']

print(f"X Shape: {X.shape}")
print(f"y Shape: {y.shape}")
```

```
X Shape: (873757,)
y Shape: (873757,)
```

In [ ]:
```
y.value_counts()
```

Out[ ]:
```
2    332892
1    321370
3    215259
4      3654
5       454
6       128
Name: results_labels, dtype: int64
```

Note: the above results indicate the results are highly imbalanced

In [ ]:
```
# split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=12312

print(f"Training records, X_train: {X_train.shape} y_train: {y_train.shape}"
print(f"Testing records, X_test: {X_test.shape} y_test: {y_test.shape}")
```

```
Training records, X_train: (655317,) y_train: (655317,)
Testing records, X_test: (218440,) y_test: (218440,)
```

In [ ]:
```
# Reset the indices of X_train and y_train
X_train.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
# Remove NaN values from X_train
X_train = X_train.dropna()
# Remove corresponding rows from y_train based on the NaN removal from X_tra
y_train = y_train[X_train.index]

# Reset the indices of X_test and y_test
X_test.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
# Remove NaN values from X_test
X_test = X_test.dropna()
# Remove corresponding rows from y_test based on the NaN removal from X_test
y_test = y_test[X_test.index]
```

In [ ]:
```
# Verify shapes are equal for train and test sets
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape, '\n')
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (655193,)
y_train shape: (655193,)

X_test shape: (218392,)
y_test shape: (218392,)
```

Model #1: Naive-Bayes

In [ ]:
```
# Instantiate NB pipeline
pipe_nb = make_pipeline(
    CountVectorizer(lowercase=False, stop_words='english', max_df=0.8, min_d
    MultinomialNB()
)
```

In [ ]:
```
# Verify there are no NaN before model training
nan_count_X_train = X_train.isna().sum()
nan_count_y_train = y_train.isna().sum()
print("Number of NaN values in X_train:", nan_count_X_train)
print("Number of NaN values in y_train:", nan_count_y_train)
```

```
Number of NaN values in X_train: 0
Number of NaN values in y_train: 0
```

In [ ]:
```
#%time nb.fit(countvectorizer.fit_transform(X_train), y_train)
%time pipe_nb.fit(X_train, y_train);
```

```
CPU times: user 1min 3s, sys: 1.68 s, total: 1min 5s
Wall time: 1min 6s
```

In [ ]:
```
# make class predictions
%time y_pred = pipe_nb.predict(X_test)
```

```
CPU times: user 10 s, sys: 34.2 ms, total: 10.1 s
Wall time: 10.1 s
```

In [ ]:
```
# calculate accuracy of class predictions
print(f"Test Accuracy: {metrics.accuracy_score(y_test, y_pred) * 100:.1f}%")
```

```
Test Accuracy: 44.4%
```

In [ ]:
```
# calculate precision and recall
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.43      0.38      0.40     80728
           2       0.47      0.66      0.55     82948
           3       0.38      0.22      0.28     53657
           4       0.00      0.00      0.00       910
           5       0.00      0.00      0.00       117
           6       0.00      0.00      0.00        32

    accuracy                           0.44    218392
   macro avg       0.21      0.21      0.20    218392
weighted avg       0.43      0.44      0.43    218392
```

In [ ]:
```
# calculate the confusion matrix
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[30625 38043 12060     0     0     0]
 [21722 54591  6635     0     0     0]
 [18427 23532 11698     0     0     0]
 [  289   464   157     0     0     0]
 [   41    57    19     0     0     0]
 [   10    17     5     0     0     0]]
```

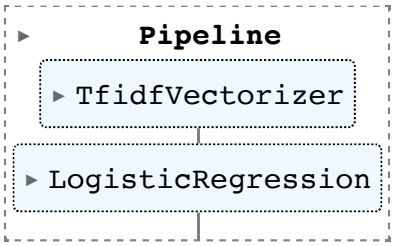
Model #2: Multinomial Logistic Regression

In [ ]:
```
pipe_logreg = make_pipeline(
    CountVectorizer(lowercase=False, stop_words='english', max_df=0.8, min_d
    #TfidfVectorizer(lowercase=False, stop_words='english', max_df=0.8, min_
    LogisticRegression(max_iter=5000, multi_class='multinomial', class_weigh
)
```

In [ ]:
```
%time pipe_logreg.fit(X_train, y_train)
```

```
CPU times: user 1min 6s, sys: 2.55 s, total: 1min 8s
Wall time: 1min 19s
```

Out[ ]:



In [ ]:
```
%time y_pred = pipe_logreg.predict(X_test)
```

```
CPU times: user 10 s, sys: 115 ms, total: 10.1 s
Wall time: 10.2 s
```

In [ ]:
```
pd.Series(y_pred).value_counts()
```

Out[ ]:
```
6    55968
5    50281
2    49115
3    24851
1    24748
4    13429
dtype: int64
```

In [ ]:
```
print(f"Test Accuracy: {metrics.accuracy_score(y_test, y_pred) * 100:.1f}%")
```

```
Test Accuracy: 20.9%
```

In [ ]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.45      0.14      0.21     80728
           2       0.52      0.31      0.38     82948
           3       0.36      0.17      0.23     53657
           4       0.01      0.08      0.01       910
           5       0.00      0.36      0.00       117
           6       0.00      0.38      0.00        32

    accuracy                           0.21    218392
   macro avg       0.22      0.24      0.14    218392
weighted avg       0.45      0.21      0.28    218392
```

Model #3: Support Vector

In [ ]:
```
from sklearn.svm import SVC
```

The SGDClassifier in scikit-learn can be used for multinomial regression, but it typically works better for binary classification or linear models. For multinomial regression, especially when dealing with multiple classes and non-linear decision boundaries, using a Support Vector Machine (SVM) with a kernel like RBF (SVC with kernel='rbf') is often a better choice.

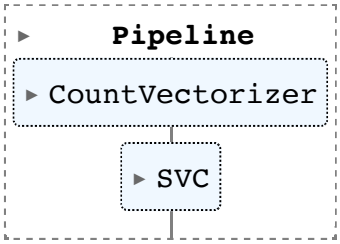
In [ ]:
```
myCountVectorizer = CountVectorizer(lowercase=False, stop_words='english', m
mySVC = SVC(max_iter=300, kernel='rbf')#, class_weight='balanced')

pipe_svc = make_pipeline(
    myCountVectorizer,
    mySVC
)
```

In [ ]:
```
%time pipe_svc.fit(X_train, y_train)
```

```
CPU times: user 2min 35s, sys: 2min 28s, total: 5min 3s
Wall time: 5min 28s
```

Out[ ]:



In [ ]:
```
%time y_pred = pipe_svc.predict(X_test)
```

```
CPU times: user 15.9 s, sys: 200 ms, total: 16.1 s
Wall time: 16.1 s
```

In [ ]:
```
print(f"SVC RBF Kernel - Test Accuracy: {metrics.accuracy_score(y_test, y_pr
print(f"SVC RBF Kernel - Test F1-score: {metrics.f1_score(y_test, y_pred, av
```

```
SVC RBF Kernel - Test Accuracy: 55.4%
SVC RBF Kernel - Test F1-score: 52.7%
```

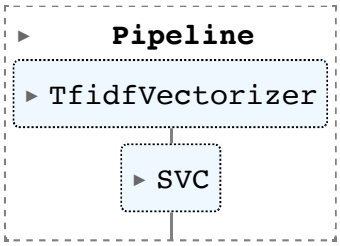
Try the best performing SVC RBF kernel with the TF-IDF vectorizer:

In [ ]:
```
pipe_svc = make_pipeline(
    TfidfVectorizer(lowercase=False, stop_words='english', max_df=0.8, min_d
    SVC(max_iter=300, kernel='rbf')
)
```

In [ ]:
```
%time pipe_svc.fit(X_train, y_train)
```

```
CPU times: user 4min 42s, sys: 2min 29s, total: 7min 12s
Wall time: 21min 54s
```

Out[ ]:



In [ ]:
```
%time y_pred = pipe_svc.predict(X_test)
```

```
CPU times: user 1min 3s, sys: 152 ms, total: 1min 3s
Wall time: 1min 4s
```

In [ ]:
```
print(f"TF-IDF Vectorizer - SVC RBF Kernel - Test Accuracy: {metrics.accurac
```

```
TF-IDF Vectorizer - SVC RBF Kernel - Test Accuracy: 37.1%
```

Note: the above ML classifiers perform very poorly when the prompted to balance the classes during model fitting

Summary:

The vectorizer I used in all instances of my models was CountVectorizer; however, I did try the TF-IDF vectorizer on my best performing model and it dropped the test accuracy. For this reason I decided to keep the regular CountVectorizer as my best pre-processing technique.

In addition to using the CountVectorizer I kept capitalizations since some words that are capitalized could have more meaning. The stop words were removed to reduce noise. I removed words that appear in more than 80% of the comments will be ignored since they probably do not have a lot of meaning. I did the same for words under 20%. I limited the terms to 1,000 maximum to help with computation and dimensionality, and lastly I set the ngram range to 1-5 or 1-10 to allow the text to capture more patterns/context.

The first Naive Bayes model had a test accuracy of 44.4%, the multinomial Logistic Regression had a test accuracy of 20.9%, and the Support Vector Classifier with an RBF kernel performed the best with a test accuracy of 55.4%. This is not very strong performance which leads me to believe there are ML models that could capture the patterns in the dataset better than the 3 algorithms I tried.

The challenge of this problem was also made more difficult due to the results class being imbalanced. Even when accounting for this when balancing the class_weight parameter the test accuracy was still not 60%+. Future steps would be to try more models like Neural Networks, further tune model hyperparameters, and consider various ways to handle the class imbalance.

(experiment) How do Logistic Regression and Support Vector Classification perform with binary classification?

In [ ]:
```
pipe_svc = make_pipeline(
    CountVectorizer(lowercase=False, stop_words='english', max_df=0.8, min_d
    SVC(max_iter=300, kernel='linear')#, class_weight='balanced')
)
```

In [ ]:
```
encoding_binary = {
    'Fail': 0,
    'Pass': 1
}

df_exploded['results_labels_binary'] = df_exploded['results'].map(encoding_b
```

In [ ]:
```
df_exploded['results_labels_binary'].value_counts()
```

Out[ ]:
```
1.0    332892
0.0    321370
Name: results_labels_binary, dtype: int64
```

In [ ]:
```
# define X and y
X = df_exploded['text_comments']
y = df_exploded['results_labels_binary']

print(f"X Shape: {X.shape}")
print(f"y Shape: {y.shape}")
```

```
X Shape: (873757,)
y Shape: (873757,)
```

In [ ]:
```
# split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=12312

print(f"Training records, X_train: {X_train.shape} y_train: {y_train.shape}"
print(f"Testing records, X_test: {X_test.shape} y_test: {y_test.shape}")
```

```
Training records, X_train: (655317,) y_train: (655317,)
Testing records, X_test: (218440,) y_test: (218440,)
```

In [ ]:
```
# Reset the indices of X_train and y_train
X_train.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
# Remove NaN values from X_train
X_train = X_train.dropna()
# Remove corresponding rows from y_train based on the NaN removal from X_tra
y_train = y_train[X_train.index]

# Reset the indices of X_test and y_test
X_test.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
# Remove NaN values from X_test
X_test = X_test.dropna()
# Remove corresponding rows from y_test based on the NaN removal from X_test
y_test = y_test[X_test.index]

###

X_train.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
# Remove NaN values from X_train
y_train = y_train.dropna()
# Remove corresponding rows from y_train based on the NaN removal from X_tra
X_train = X_train[y_train.index]

# Reset the indices of X_test and y_test
X_test.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
# Remove NaN values from X_test
y_test = y_test.dropna()
# Remove corresponding rows from y_test based on the NaN removal from X_test
X_test = X_test[y_test.index]
```

In [ ]:
```
# Verify shapes are equal for train and test sets
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape, '\n')
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (490436,)
y_train shape: (490436,)

X_test shape: (163676,)
y_test shape: (163676,)
```

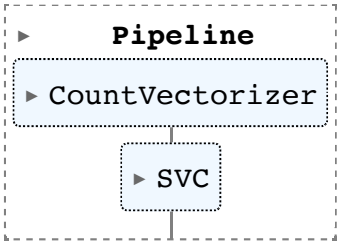
In [ ]:
```
# Verify there are no NaN before model training
nan_count_X_train = X_train.isna().sum()
nan_count_y_train = y_train.isna().sum()
print("Number of NaN values in X_train:", nan_count_X_train)
print("Number of NaN values in y_train:", nan_count_y_train)
```

```
Number of NaN values in X_train: 0
Number of NaN values in y_train: 0
```

In [ ]:
```
%time pipe_svc.fit(X_train, y_train)
```

```
CPU times: user 49.7 s, sys: 972 ms, total: 50.7 s
Wall time: 51 s
```

Out[ ]:



In [ ]:
```
%time y_pred = pipe_svc.predict(X_test)
```

```
CPU times: user 7.82 s, sys: 37.9 ms, total: 7.86 s
Wall time: 7.9 s
```

In [ ]:
```
print(f"SVC Linear Kernel (Binary Classification) - Test Accuracy: {metrics.
print(f"SVC Linear Kernel (Binary Classification) - Test F1-score: {metrics.
```

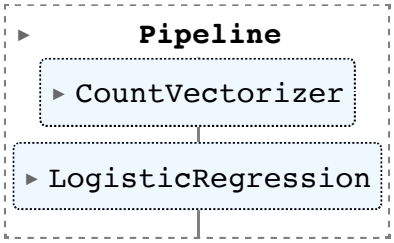
```
SVC Linear Kernel (Binary Classification) - Test Accuracy: 61.1%
SVC Linear Kernel (Binary Classification) - Test F1-score: 60.9%
```

In [ ]:
```
pipe_logreg_binary = make_pipeline(
    CountVectorizer(lowercase=False, stop_words='english', max_df=0.8, min_d
    #TfidfVectorizer(lowercase=False, stop_words='english', max_df=0.8, min_
    LogisticRegression(max_iter=5000, multi_class='auto', class_weight='bala
)
```

In [ ]:
```
%time pipe_logreg_binary.fit(X_train, y_train)
```

```
CPU times: user 52 s, sys: 1.57 s, total: 53.6 s
Wall time: 56.7 s
```

Out[ ]:



In [ ]:
```
%time y_pred = pipe_logreg_binary.predict(X_test)
```

```
CPU times: user 7.23 s, sys: 123 ms, total: 7.36 s
Wall time: 7.39 s
```

In [ ]:
```
print(f"Logistic Regression (Binary Classification) - Test Accuracy: {metric
print(f"Logistic Regression (Binary Classification) - Test F1-score: {metric
```

```
Logistic Regression (Binary Classification) - Test Accuracy: 61.1%
Logistic Regression (Binary Classification) - Test F1-score: 60.9%
```