CSCI 544 – Applied Natural Language Processing

Homework 1 – Vu Truong Si – 6031936649

Python version used: 3.8.8

New library used: contractions, TfidfVectorizer, train_test_split, Perceptron, LinearSVC, make_pipeline, StandardScaler, LogisticRegression, MultinomialNB.

1. Data Preparation

I downloaded and extracted the dataset from Amazon S3 bucket, then read the dataset into a dataframe using Pandas. During data reading, I skipped bad lines by setting the parameter **error_bad_line** to False.

To keep the Reviews and Ratings, I only selected the two fields **review_body** and **star_rating**. I converted the ratings into readable numerical values, then encoded them as instructed using the following map {1:1, 2:1, 3:2, 4:3, 5:3}.

I randomly sampled 20000 rows from each class in order to create a balanced dataset with a total of 60000 rows.

2. Data Cleaning

I implemented 5 different data cleaning techniques:

- Convert reviews into lowercase: I used **.lower()** to achieve this.
- Remove HTML and URLs from the reviews: I used regular expression to eliminate the patterns of HTML tags and URLs. For example, “**<[^<]+?>**” will target all the HTML tags and I can remove them from the text.
- Expand contractions: I used the library **contractions**, specifically **contractions.fix()** to perform this technique.
- Remove non-alphabetical characters: I also used regular expression to achieve this. I kept all the characters between a-z and A-Z.
- Remove extra spaces: I combined regular expression with **.strip()** to remove extra spaces in the reviews.

Afterward, I counted the number of characters for each review by computing the length of each string using **.len()** on the dataframe, then averaged those results using **.mean()**. For one of the cases, the average before and after cleaning were `269.9424333333333` and `259.18215`.

3. Preprocessing

To remove stopwords, I used **stopwords.words("english")** to get the list of stopwords, then excluded a word in the review if that word belongs to the list.

Note: I performed two experiments, one with stopwords removed, and one with stopwords. The results were noticeably better when stopwords were not removed. I will demonstrate the results down below.

To lemmatize the sentences, I used WordNetLemmatizer. Before lemmatizing, I identified the part-of-speech tag of each word using **nltk.pos_tag()** then put that as an argument for the **lemmatize()** function. I had to convert the tags from nltk to tags that WordNetLemmatizer can understand. For example,

Afterward, I counted the number of characters for each review after preprocessing, averaged those results, then compare it with the result from the previous step. The result for this section is based on the dataset with stopwords retained since the dataset gave a better model performance. I also included the average length of reviews without stopwords in the notebook for comparison. For one of the cases, the average before and after preprocessing (with stopwords removal) were `259.18215` and `151.5338`, while the average before and after preprocessing (without stopwords removal) were `259.18215` and `249.26885`. In the .py file, I only showed the average in the case without stopwords removal, but I kept both cases' results in the notebook.

4. Feature Extraction

I used TfidfVectorizer with **min_df**, **max_df**, and **ngram_range** paramters to extract the features from the reviews. After that, I split the dataset into train and test sets with 48000 instances in the training set and 12000 instances in the test set (80/20 ratio).

5. Perceptron

Note: I implemented GridSearchCV to tune the hyperparameters for all 4 models, however, the best hyperparameters after cross validation did not give good results on the test set, therefore, I did not keep these results. Also, for each model, I used two different datasets, one with stopwords and one without. The results of the models with stopwords retained were clearly better so chose to print them in the .py file.

For perceptron, I used the library Perceptron from sklearn.linear_model. I initialized a model with the following hyperparameters and fit the training data into it:

Perceptron(alpha = 0.0001, tol = 1e-3)

Below are the metrics for Perceptron following the required format, with and without stopwords respectively:

```
0.7006257822277847,0.6899186591077151,0.695230998509687
0.5805063291139241,0.5828673106253177,0.5816844241501776
0.7713933415536375,0.7802444499875281,0.7757936507936508
0.6841751509651154,0.6843434732401871,0.6842363578178384

0.6251508568670046,0.6321698804002929,0.6286407766990292
0.5039239001189061,0.5333501132645356,0.5182196135974566
0.7009857612267251,0.6513994910941476,0.6752835663413348
0.6100201727375452,0.605639828252992,0.6073813188792735
```

We can see that with stopwords retained, Perceptron performed better in all three classes.

6. SVM

I implemented LinearSVC from sklearn.svm with and without normalization to test the results. I found that the model without normalization worked better so I kept that model. The following model was used:

LinearSVC(C = 1.0, tol = 1e-3)

Below are the metrics for SVM following the required format, with and without stopwords respectively:

```
0.732760736196319,0.7360118314025141,0.734382685686178
0.62851929092805,0.6128622267412303,0.6205920205920205
0.797260943996087,0.8131703666749813,0.8051370708816992
0.719513657040152,0.7206814749395752,0.7200372590532993

0.678027556200145,0.6846473029045643,0.6813213504979354
0.5739722440429432,0.5517241379310345,0.5626283367556468
0.7334322453016815,0.7547073791348601,0.743917732631051
0.6618106818482566,0.663692939990153,0.6626224732948778
```

7. Logistic Regression

I used the library LogisticRegression from sklearn.linear_model and set the “max_iter” hyperparameter to 1000 in order for convergence to happen. I used the following settings:

LogisticRegression(C = 1.0, tol = 1e-3, max_iter = 1000)

Below are the metrics for Logistic Regression following the required format, with and without stopwords respectively:

```
0.7514822134387352,0.749815134335716,0.7506477483035164
0.6492796820665673,0.6644636502287747,0.6567839195979899
0.8273051451859399,0.8101771015215764,0.8186515437933206
0.7426890135637475,0.7414852953620223,0.742027737231609

0.710822722820764,0.708567244325116,0.7096931915413763
0.6020304568527919,0.5970299521771961,0.5995197775811956
0.7593058350100603,0.7681933842239186,0.763723754110802
```

```
0.6907196715612054,0.6912635269087435,0.690978907744458
```

8. Naïve Bayes

I implemented a Multinomial Naïve Bayes model with alpha = 1.0 and kept all the default hyperparameters.

MultinomialNB(alpha = 1.0)

Below are the metrics for Logistic Regression following the required format, with and without stopwords respectively:

```
0.7561962569549823,0.7369977816120286,0.7464735988016478
0.6328524895300139,0.6914082358922217,0.6608357628765792
0.8441835645677694,0.7892242454477426,0.815779296119634
0.744410770350922,0.7392100876506643,0.741029552599287

0.7078305519897304,0.6729314132291921,0.6899399399399399
0.5812712275594372,0.6030707274100177,0.5919703520691785
0.7434094903339191,0.7534351145038168,0.7483887274105903
0.6775037566276955,0.6764790850476755,0.6767663398065696
```

In order to calculate the metrics for each model, I created a function called “metrics_calculator”. This function computes the confusion matrix then the precision, recall, f1-score, and the average of those scores. After that, it prints the desired outputs line by line.

Comments on the metrics: It seems like the precision, recall and f1-score for all three classes improved when stopwords were not removed. This can be due to the fact that some stopwords hold important meanings to the instances and removing them worsen the performance of the models. Therefore, I decided to keep the stopwords in the .py file and report the best scores accordingly.

Below is the Jupyter Notebook:

In [1]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet', quiet = True)
import re
from bs4 import BeautifulSoup
import contractions
import warnings
warnings.filterwarnings('ignore')
nltk.download('averaged_perceptron_tagger', quiet = True)
```

Out[1]: True

Read Data

In [2]:
```
# Read the data while skipping bad lines.

dataframe = pd.read_table("amazon_reviews_us_Beauty_v1_00.tsv", error_bad_lines = False, warn_bad_lines=False)
```

In [3]:
```
dataframe
```

Out[3]:

	marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	helpful_vot
0	US	1797882	R3I2DHQBR577SS	B001ANOOOE	2102612	The Naked Bee Vitmin C Moisturizing Sunscreen ...	Beauty	5	(
1	US	18381298	R1QNE9NQFJC2Y4	B0016J22EQ	106393691	Alba Botanica Sunless Tanning Lotion, 4 Ounce	Beauty	5	(
2	US	19242472	R3LIDG2Q4LJBAO	B00HU6UQAG	375449471	Elysee Infusion Skin Therapy Elixir, 2oz.	Beauty	5	(

	marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	helpful_vot
3	US	19551372	R3KSZHPAEVPEAL	B002HWS7RM	255651889	Diane D722 Color, Perm And Conditioner Process...	Beauty	5	(
4	US	14802407	RAI2OIG50KZ43	B00SM99KWU	116158747	Biore UV Aqua Rich Watery Essence SPF50+/PA+++...	Beauty	5	(
...	...	...	...	...	...	...	...	...	
5094302	US	50113639	RZ7RZ02MTP4SL	B000050B70	185454094	Conair NE150NSCS Cordless Nose and Ear Hair Tr...	Beauty	5	1(
5094303	US	52940456	R2IRC0IZ8YCE5T	B000050FF2	678848064	Homedics Envirascape Sound Spa Alarm Clock Radio	Beauty	3	2:
5094304	US	47587881	R1U4ZSXOD228CZ	B000050B6U	862195513	Conair Instant Heat Curling Iron	Beauty	5	8!
5094305	US	53047750	R3SFJLZE09URWM	B000050FDE	195242894	Oral-B Professional Care 1000 Power Toothbrush	Beauty	5	1(
5094306	US	51193940	R1MEWK4I7YS5XK	B000050AUD	190668305	Sonicare PL-4 (4700) Sonic Toothbrush	Beauty	5	2:

5094307 rows × 15 columns

Keep Reviews and Ratings

In [4]:
```
reviews_and_ratings = dataframe[["review_body", "star_rating"]]
```

In [5]:
```
# Convert string to numerical values ("1.0" -> 1.0) and mark as None if the value is invalid.
reviews_and_ratings.loc[: , "star_rating"] = pd.to_numeric(reviews_and_ratings["star_rating"], errors = 'coerce', downcas
```

In [6]:
```
# Drop null values.
reviews_and_ratings = reviews_and_ratings.dropna(how = "any")
```

In [7]:
```
reviews_and_ratings
```

Out[7]:

	review_body	star_rating
0	Love this, excellent sun block!!	5.0
1	The great thing about this cream is that it do...	5.0
2	Great Product, I'm 65 years old and this is al...	5.0
3	I use them as shower caps & conditioning caps....	5.0
4	This is my go-to daily sunblock. It leaves no ...	5.0
...	...	...
5094302	After watching my Dad struggle with his scisso...	5.0
5094303	Like most sound machines, the sounds choices a...	3.0
5094304	I bought this product because it indicated 30 ...	5.0
5094305	We have used Oral-B products for 15 years; thi...	5.0
5094306	I love this toothbrush. It's easy to use, and ...	5.0

5093907 rows × 2 columns

We form three classes and select 20000 reviews randomly from each class.

In [8]:
```
ratings_dict = {1:1, 2:1, 3:2, 4:3, 5:3}
```

In [9]:
```
# Map the ratings to appropriate classes.

reviews_and_ratings.loc[:,"class"] = reviews_and_ratings["star_rating"].map(ratings_dict)
```

In [10]:
```
# Randomly sample 20000 rows from each class.

class_1 = reviews_and_ratings[reviews_and_ratings["class"] == 1].sample(20000)
class_2 = reviews_and_ratings[reviews_and_ratings["class"] == 2].sample(20000)
class_3 = reviews_and_ratings[reviews_and_ratings["class"] == 3].sample(20000)
```

In [11]:
```
balanced_dataset = pd.concat([class_1, class_2, class_3], axis = 0)
```

In [12]:
```
balanced_dataset
```

Out[12]:

	review_body	star_rating	class
4965866	You'll need heavy duty shears to open packages...	1.0	1
2225032	It didn't take the brassiness out of my hair a...	1.0	1
3826255	I have a suspicion that this brush may be the ...	2.0	1
4812345	I was so thrilled to find a cologne that has b...	1.0	1
2084719	While the toothbrush does an excellent job cle...	1.0	1
...	...	...	...
3386104	The picture is deceptive in that you only get ...	4.0	3
4127620	Most of these colors are very natural. A lot o...	4.0	3
2404233	it does work.	4.0	3
279257	Exactly what I expected	5.0	3

	review_body	star_rating	class
2552877	Love the fragrance	5.0	3

60000 rows × 3 columns

In [13]:
```
balanced_dataset["review_length"] = [len(str(x)) for x in balanced_dataset["review_body"]]
```

In [14]:
```
# Calculate the average length of the reviews before cleaning.

length_before_cleaning = balanced_dataset["review_length"].mean()
```

Data Cleaning

In [15]:
```
# Convert text to lowercase
balanced_dataset["review_body"] = balanced_dataset["review_body"].str.lower()
```

In [16]:
```
# Remove HTML tags
balanced_dataset["review_body"] = [re.sub('<[^<]+?>', '', str(x)) for x in balanced_dataset["review_body"]]
```

In [17]:
```
# Remove URLs
balanced_dataset["review_body"] =  [re.sub(r"http\S+","", str(x)) for x in balanced_dataset["review_body"]]
```

In [18]:
```
# Expand contractions
balanced_dataset["review_body"] = [contractions.fix(str(x)) for x in balanced_dataset["review_body"]]
```

In [19]:
```
# Remove non-alphabetical characters
balanced_dataset["review_body"] = [re.sub(r"[^a-zA-Z ]", "", str(x)) for x in balanced_dataset["review_body"]]
```

In [20]:
```
# Remove excess spaces
balanced_dataset["review_body"] = balanced_dataset["review_body"].replace("\s+", " ", regex = True).str.strip()
```

In [21]:
```
balanced_dataset["review_length"] = [len(str(x)) for x in balanced_dataset["review_body"]]
```

In [22]:
```
# Calculate the average length of the reviews after cleaning.

length_after_cleaning = balanced_dataset["review_length"].mean()
```

In [23]:
```
print(str(length_before_cleaning) + "," + str(length_after_cleaning))
```

```
269.7167333333333,258.84695
```

Pre-processing

remove the stop words

In [24]:
```
nltk.download('stopwords', quiet = True)
from nltk.corpus import stopwords
stop = stopwords.words("english")
```

In [25]:
```
balanced_dataset["review_body_no_stopwords"] = balanced_dataset["review_body"].apply(lambda x: ' '.join([word for word in
```

perform lemmatization

In [26]:
```
from nltk.stem import WordNetLemmatizer
```

In [27]:
```
# Function to convert nltk pos tags into tags that WordNetLemmatizer can understand.

def nltk_pos_converter(tag):
    if tag.startswith("J"):
        return "a"
    elif tag.startswith("V"):
        return "v"
    elif tag.startswith("R"):
        return "r"
```

```
    else:
        return "n"
```

In [28]:

```
# Create another column for text without stopwords.

lemmatizer = WordNetLemmatizer()
balanced_dataset["review_body_no_stopwords"] = balanced_dataset["review_body_no_stopwords"].apply(lambda x: ' '.join([lem
```

In [29]:

```
# Lemmatize the text with stopwords.

lemmatizer = WordNetLemmatizer()
balanced_dataset["review_body"] = balanced_dataset["review_body"].apply(lambda x: ' '.join([lemmatizer.lemmatize(word, po
```

In [30]:

```
balanced_dataset
```

Out[30]:

	review_body	star_rating	class	review_length	review_body_no_stopwords
4965866	you will need heavy duty shear to open package...	1.0	1	516	need heavy duty shear open package like packag...
2225032	it do not take the brassiness out of my hair a...	1.0	1	445	take brassiness hair infact add brassiness can...
3826255	i have a suspicion that this brush may be the ...	2.0	1	436	suspicion brush may recent breakage experience...
4812345	i be so thrilled to find a cologne that have b...	1.0	1	472	thrill find cologne discountinued time even am...
2084719	while the toothbrush do an excellent job clean...	1.0	1	253	toothbrush excellent job clean teeth manufactu...
...	...	...	...	...	...
3386104	the picture be deceptive in that you only get ...	4.0	3	566	picture deceptive get little guy arrive adorab...
4127620	most of these color be very natural a lot of b...	4.0	3	183	color natural lot brown grey black white vibra...
2404233	it do work	4.0	3	12	work
279257	exactly what i expect	5.0	3	23	exactly expect
2552877	love the fragrance	5.0	3	18	love fragrance

60000 rows × 5 columns

In [31]:

```
balanced_dataset["review_length"] = [len(str(x)) for x in balanced_dataset["review_body_no_stopwords"]]
```

In [32]:
```
# Calculate the average length of reviews after preprocessing (Stopwords removed).

length_after_preprocessing = balanced_dataset["review_length"].mean()
```

In [33]:
```
print(str(length_after_cleaning) + "," + str(length_after_preprocessing))
```

```
258.84695,151.47631666666666
```

In [34]:
```
balanced_dataset["review_length"] = [len(str(x)) for x in balanced_dataset["review_body"]]
```

In [35]:
```
# Calculate the average length of reviews after preprocessing (Stopwords retained).

length_after_preprocessing = balanced_dataset["review_length"].mean()
```

In [36]:
```
print(str(length_after_cleaning) + "," + str(length_after_preprocessing))
```

```
258.84695,248.88255
```

TF-IDF Feature Extraction

In [37]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [38]:
```
tfidf_vectorizer = TfidfVectorizer(min_df = 0.0001, max_df = 0.5, ngram_range = (1,3))
tfidf_features = tfidf_vectorizer.fit_transform(balanced_dataset["review_body"])
```

In [39]:
```
from sklearn.model_selection import train_test_split
```

In [40]:
```
X = tfidf_features
y = balanced_dataset["class"]
```

In [41]:
```
# Datasets with stopwords.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

In [42]:
```
tfidf_vectorizer = TfidfVectorizer(min_df = 0.0001, max_df = 0.5, ngram_range = (1,3))
tfidf_features_without_stopwords = tfidf_vectorizer.fit_transform(balanced_dataset["review_body_no_stopwords"])
```

In [43]:
```
X = tfidf_features_without_stopwords
```

In [44]:
```
# Datasets without stopwords.

X_train_wos, X_test_wos, y_train_wos, y_test_wos = train_test_split(X, y, test_size = 0.2)
```

Perceptron

In [45]:
```
# Function to calculate the metrics and print them.

def metrics_calculator(y_test, y_pred):
    class_1_cm = dict()
    class_2_cm = dict()
    class_3_cm = dict()

    confusion_matrix = [[0]*3 for i in range(3)]

    for i in range(len(y_test)):
        if y_test[i] == 1:
            if y_pred[i] == 1:
                confusion_matrix[0][0] += 1
            elif y_pred[i] == 2:
                confusion_matrix[0][1] += 1
            else:
                confusion_matrix[0][2] += 1
        elif y_test[i] == 2:
            if y_pred[i] == 1:
                confusion_matrix[1][0] += 1
            elif y_pred[i] == 2:
                confusion_matrix[1][1] += 1
```

```
            else:
                confusion_matrix[1][2] += 1
        else:
            if y_pred[i] == 1:
                confusion_matrix[2][0] += 1
            elif y_pred[i] == 2:
                confusion_matrix[2][1] += 1
            else:
                confusion_matrix[2][2] += 1

    class_1_tp = confusion_matrix[0][0]
    class_1_tn = confusion_matrix[1][1] + confusion_matrix[1][2] + confusion_matrix[2][1] + confusion_matrix[2][2]
    class_1_fp = confusion_matrix[1][0] + confusion_matrix[2][0]
    class_1_fn = confusion_matrix[0][1] + confusion_matrix[0][2]

    class_2_tp = confusion_matrix[1][1]
    class_2_tn = confusion_matrix[0][0] + confusion_matrix[0][2] + confusion_matrix[2][0] + confusion_matrix[2][2]
    class_2_fp = confusion_matrix[0][1] + confusion_matrix[2][1]
    class_2_fn = confusion_matrix[1][0] + confusion_matrix[1][2]

    class_3_tp = confusion_matrix[2][2]
    class_3_tn = confusion_matrix[0][0] + confusion_matrix[0][1] + confusion_matrix[1][0] + confusion_matrix[1][1]
    class_3_fp = confusion_matrix[0][2] + confusion_matrix[1][2]
    class_3_fn = confusion_matrix[2][0] + confusion_matrix[2][1]

    class_1_precision = (class_1_tp) / (class_1_tp + class_1_fp)
    class_1_recall = (class_1_tp) / (class_1_tp + class_1_fn)
    class_1_f1_score = 2 * class_1_precision * class_1_recall / (class_1_precision + class_1_recall)

    class_2_precision = (class_2_tp) / (class_2_tp + class_2_fp)
    class_2_recall = (class_2_tp) / (class_2_tp + class_2_fn)
    class_2_f1_score = 2 * class_2_precision * class_2_recall / (class_2_precision + class_2_recall)

    class_3_precision = (class_3_tp) / (class_3_tp + class_3_fp)
    class_3_recall = (class_3_tp) / (class_3_tp + class_3_fn)
    class_3_f1_score = 2 * class_3_precision * class_3_recall / (class_3_precision + class_3_recall)

    print(str(class_1_precision) + "," + str(class_1_recall) + "," + str(class_1_f1_score))
    print(str(class_2_precision) + "," + str(class_2_recall) + "," + str(class_2_f1_score))
    print(str(class_3_precision) + "," + str(class_3_recall) + "," + str(class_3_f1_score))

    average_precision = (class_1_precision + class_2_precision + class_3_precision)/3
    average_recall = (class_1_recall + class_2_recall + class_3_recall)/3
    average_f1_score = (class_1_f1_score + class_2_f1_score + class_3_f1_score)/3
```

In [46]:

```
from sklearn.linear_model import Perceptron
```

In [47]:

```
# Perceptron without stopwords.

perceptron = Perceptron(alpha = 0.0001, tol = 1e-3)
perceptron.fit(X_train_wos, y_train_wos)
y_pred_wos = perceptron.predict(X_test_wos)
metrics_calculator(y_test_wos.values, y_pred_wos)

# 0.5979827089337176,0.6264150943396226,0.6118687799483966
# 0.5076035658101731,0.49137055837563454,0.4993551715243745
# 0.6954251616111388,0.6847001223990208,0.6900209695325028
# 0.6003371454516765,0.6008285917047593,0.6004149736684247
```

```
0.5979827089337176,0.6264150943396226,0.6118687799483966
0.5076035658101731,0.49137055837563454,0.4993551715243745
0.6954251616111388,0.6847001223990208,0.6900209695325028
0.6003371454516765,0.6008285917047593,0.6004149736684247
```

In [48]:

```
# Perceptron with stopwords.

perceptron = Perceptron(alpha = 0.0001, tol = 1e-3)
perceptron.fit(X_train, y_train)
y_pred = perceptron.predict(X_test)
metrics_calculator(y_test.values, y_pred)
```

```
0.6828846628797234,0.688667496886675,0.685763888888889
0.5745800952619704,0.5727136431784108,0.5736453510198974
0.773346794548208,0.7692693949284459,0.7713027061044683
0.6769371842299673,0.6768835116645106,0.6769039820044181
```

SVM

In [49]:

```
from sklearn.svm import LinearSVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

In [50]:

```
# Linear SVC without scaling and without stopwords.

lsvc = LinearSVC(C = 1.0, tol = 1e-3)
lsvc.fit(X_train_wos, y_train_wos)
y_pred_wos = lsvc.predict(X_test_wos)
metrics_calculator(y_test_wos.values, y_pred_wos)

# 0.662357036300348,0.670188679245283,0.6662498436913843
# 0.577438370846731,0.5469543147208121,0.5617831074035454
# 0.7416391898257183,0.7708690330477356,0.7559716720681791
# 0.6604781989909325,0.6626706756712769,0.6613348743877029
```

```
0.662357036300348,0.670188679245283,0.6662498436913843
0.577438370846731,0.5469543147208121,0.5617831074035454
0.7416391898257183,0.7708690330477356,0.7559716720681791
0.6604781989909325,0.6626706756712769,0.6613348743877029
```

In [51]:

```
# Linear SVC with scaling and without stopwords.

pipeline = make_pipeline(StandardScaler(with_mean = False), LinearSVC(C = 1.0, tol = 1e-3))
pipeline.fit(X_train_wos, y_train_wos)
y_pred = pipeline.predict(X_test_wos)
metrics_calculator(y_test_wos.values, y_pred_wos)

# 0.662357036300348,0.670188679245283,0.6662498436913843
# 0.577438370846731,0.5469543147208121,0.5617831074035454
# 0.7416391898257183,0.7708690330477356,0.7559716720681791
# 0.6604781989909325,0.6626706756712769,0.6613348743877029
```

```
0.662357036300348,0.670188679245283,0.6662498436913843
0.577438370846731,0.5469543147208121,0.5617831074035454
0.7416391898257183,0.7708690330477356,0.7559716720681791
0.6604781989909325,0.6626706756712769,0.6613348743877029
```

In [52]:

```
# Linear SVC without scaling and with stopwords.

lsvc = LinearSVC(C = 1.0, tol = 1e-5)
lsvc.fit(X_train, y_train)
y_pred = lsvc.predict(X_test)
metrics_calculator(y_test.values, y_pred)
```

```
0.7243842364532019,0.7325031133250312,0.7284210526315789
0.6320467242254951,0.6219390304847576,0.6269521410579346
```

```
0.811344327836082,0.8152146623148381,0.8132748904195367
0.7225917628382597,0.7232189353748756,0.7228826947030167
```

In [53]:

```
# Linear SVC with scaling and with stopwords.

pipeline = make_pipeline(StandardScaler(with_mean = False), LinearSVC(C = 1.0, tol = 1e-3))
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
metrics_calculator(y_test.values, y_pred)

# 0.6495160468670402,0.635118306351183,0.6422364941443143
# 0.5422086202499362,0.5312343828085957,0.5366654045184905
# 0.713700939080183,0.7441626914386141,0.7286135693215339
# 0.6351418687323865,0.6368384601994643,0.6358384893281129
```

```
0.6495160468670402,0.635118306351183,0.6422364941443143
0.5422086202499362,0.5312343828085957,0.5366654045184905
0.713700939080183,0.7441626914386141,0.7286135693215339
0.6351418687323865,0.6368384601994643,0.6358384893281129
```

Logistic Regression

In [54]:

```
from sklearn.linear_model import LogisticRegression
```

In [55]:

```
# Logistic Regression without stopwords.

logreg = LogisticRegression(C = 1.0, tol = 1e-3, max_iter = 1000)
logreg.fit(X_train_wos, y_train_wos)
y_pred_wos = logreg.predict(X_test_wos)
metrics_calculator(y_test_wos.values, y_pred_wos)

# 0.6908588648920141,0.6920754716981132,0.6914666331531985
# 0.5923927011051143,0.5850253807106599,0.5886859915719576
# 0.7685970438575236,0.7764993880048959,0.7725280077934729
# 0.6839495366182172,0.6845334134712231,0.6842268775062097
```

```
0.6908588648920141,0.6920754716981132,0.6914666331531985
0.5923927011051143,0.5850253807106599,0.5886859915719576
0.7685970438575236,0.7764993880048959,0.7725280077934729
0.6839495366182172,0.6845334134712231,0.6842268775062097
```

In [56]:

```
# Logistic Regression with stopwords.

logreg = LogisticRegression(C = 1.0, tol = 1e-3, max_iter = 1000)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
metrics_calculator(y_test.values, y_pred)
```

```
0.7475,0.7447073474470735,0.746101060511541
0.6444007858546169,0.655672163918041,0.6499876145652712
0.8294297352342159,0.8179763996987196,0.8236632536973835
0.740443507029611,0.7394519703546113,0.7399173095913986
```

Naive Bayes

In [57]:

```
from sklearn.naive_bayes import MultinomialNB
```

In [58]:

```
# Naive Bayes without stopwords.

mnb = MultinomialNB(alpha = 1.0)
mnb.fit(X_train_wos, y_train_wos)
y_pred_wos = mnb.predict(X_test_wos)
metrics_calculator(y_test_wos.values, y_pred_wos)

# 0.6936582809224319,0.6659119496855346,0.6795019894750353
# 0.5726536445926632,0.6101522842639594,0.5908085524698944
# 0.7601605619668841,0.7417380660954712,0.7508363275926156
# 0.6754908291606597,0.672600766681655,0.6737156231791818
```

```
0.6936582809224319,0.6659119496855346,0.6795019894750353
0.5726536445926632,0.6101522842639594,0.5908085524698944
0.7601605619668841,0.7417380660954712,0.7508363275926156
0.6754908291606597,0.672600766681655,0.6737156231791818
```

In [59]:

```
# Naive Bayes with stopwords.

mnb = MultinomialNB(alpha = 1.0)
mnb.fit(X_train, y_train)
y_pred = mnb.predict(X_test)
metrics_calculator(y_test.values, y_pred)
```

```
0.7457237681899412,0.7275217932752179,0.7365103378719111
0.627000695894224,0.6754122938530734,0.6503067484662577
0.8374867444326617,0.7931207632437861,0.8147001934235976
0.736737069505609,0.7320182834573591,0.7338390932539222
```