Project – Amazon Sentiment and Product Type Analysis

October 19, 2020

[781]:
```python
# import important dependencies
import random, math, time, datetime, os, json, re, pickle

# data manipulation
import numpy as np
import pandas as pd
from itertools import chain, cycle

# visualization
import matplotlib.pyplot as plt
import seaborn as sns
import missingno
plt.style.use('seaborn-whitegrid')

%matplotlib inline

# machine learning
from sklearn.calibration import CalibratedClassifierCV
from sklearn.multiclass import OneVsRestClassifier
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn import model_selection, tree, preprocessing, metrics
from sklearn.model_selection import train_test_split, cross_validate,␣
↪GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

# import warnings
# warnings.filterwarnings('ignore')
```

[782]:
```python
# converted to a function modified code from Keith Galli's GitHub - thanks␣
↪Keith!
# function that extracts 1000 random reviews from the year 2018 for each of the␣
↪5 categories
```

```python
def get_file_extract(file):
    data_2018 = []
    file_name = file
    with open(f'./Documents/project-data/{file_name}.json', 'r') as rf:
        for line in rf:
            review = json.loads(line)
            if int(review['reviewTime'].split(',')[1]) == 2018:
                data_2018.append(review)

    data_extract_2018 = random.sample(data_2018, 1000)

    with open(f'./Documents/project-data/{file_name}_sample.json', 'w') as wf:
        for review in data_extract_2018:
            wf.write(json.dumps(review)+'\n')

# get_file_extract('Electronics')
# get_file_extract('Home_and_Kitchen')
# get_file_extract('Movies_and_TV')
# get_file_extract('Pet_Supplies')
# get_file_extract('Office_Products')
```

[783]:
```python
class Category:
    ELECTRONICS = "ELECTRONICS"
    HOME = "HOME"
    MOVIES = "MOVIES"
    PETS = "PETS"
    OFFICE = "OFFICE"

class Sentiment_word:
    POSITIVE = "POSITIVE"
    NEUTRAL = "NEUTRAL"
    NEGATIVE = "NEGATIVE"

class Review:
    def __init__(self, category, text, score):
        self.category = category
        self.text = text
        self.score = score
        self.sentiment = self.get_sentiment()

    def get_sentiment(self):
        if self.score <= 2:
            return Sentiment_word.NEGATIVE
        elif self.score == 3:
            return Sentiment_word.NEUTRAL
        else:
```

```
            return Sentiment_word.POSITIVE
```

[784]:
```
files = [
             './Documents/Amazon Data Classification/Electronics_extract.json',
             './Documents/Amazon Data Classification/Home_and_Kitchen_extract.
 ↪json',
             './Documents/Amazon Data Classification/Movies_and_TV_extract.
 ↪json',
             './Documents/Amazon Data Classification/Pet_Supplies_extract.json',
             './Documents/Amazon Data Classification/Office_Products_extract.
 ↪json'
]
categories = [
              Category.ELECTRONICS,
              Category.HOME,
              Category.MOVIES,
              Category.PETS,
              Category.OFFICE
]

# Reading in the files - thanks again Keith!

reviews = []
for i in range(len(files)):
    name = files[i]
    category = categories[i]
    with open(name) as f:
        for line in f:
            review_json = json.loads(line)
            review = Review(category, review_json['reviewText'],␣
 ↪review_json['overall'])
            reviews.append(review)

len(reviews)
```

[784]: 5000

[785]:
```
train, test = train_test_split(reviews, test_size = 0.18, random_state = 1267)

train_text = [x.text for x in train]
train_sentiment = [x.sentiment for x in train]
train_category = [x.category for x in train]
```

[786]:
```
""" spell check """

from textblob import TextBlob
```

```
tb_train_text = [TextBlob(train_text_unit) for train_text_unit in train_text]
tb_train_text_correct = [tb_train_text_unit.correct() for tb_train_text_unit in␣
↪tb_train_text]
```

[787]:
```
""" stopwords removal"""

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stop_words = stopwords.words('english')
stop_words

# phrase = str('\t' + str(tb_train_text_correct[5]))
# type(phrase)

# type('\t'  str(phrase))

# word_pilot = word_tokenize(phrase)

words = [word_tokenize(str(tb_train_text_correct_unit)) for␣
↪tb_train_text_correct_unit in tb_train_text_correct]

train_text_stripped = []
[train_text_stripped.append(str(word)) for word in words if word not in␣
↪stop_words]
train_text_stripped[3]
```

[787]:
```
"['I', 'originally', 'decided', 'to', 'begin', 'my', 'cats', 'on', 'a', 'grain',
'free', 'diet', 'because', 'of', 'their', 'digestive', 'issues', '.', 'There',
'were', 'lots', 'of', 'incidents', 'of', 'throwing', 'up', 'in', 'recent',
'months', 'and', 'the', 'frequency', 'between', 'events', 'seem', 'to', 'be',
'on', 'the', 'rise', '.', 'Since', 'grain', 'is', 'not', 'a', 'part', 'of', 'a',
'cats', 'diet', 'in', 'the', 'wild', ',', 'it', 'does', 'seem', 'curious',
'that', 'it', 'would', 'be', 'a', 'part', 'of', 'a', 'cat', 'diet', 'in', 'a',
'household', 'environment', '…', 'seems', 'that', 'it', 'is', 'a', 'filler',
'.', 'It', 'took', 'a', 'big', 'longer', 'to', 'transition', 'our', 'cats',
'food', 'than', 'the', 'recommended', 'time', 'frame', 'because', 'they', 'are',
'pick', '.', 'I', 'started', 'to', 'switch', 'them', 'in', 'March', '2018',
'and', 'at', 'this', 'point', '(', 'July', ')', 'they', 'have', 'been', 'on',
'approximately', 'three', 'weeks', 'of', 'only', 'the', 'Tutor', 'brand',
'food', '.', 'Not', 'only', 'have', 'I', 'gotten', 'my', 'desired', 'result',
'with', 'the', 'decrease', 'in', 'the', 'number', 'of', 'hairballs', 'and',
'throw', 'up', 'situations', 'Am', 'having', 'to', 'clean', 'up', ',', 'but',
'an', 'amazing', 'side', 'effect', 'is', 'that', 'my', 'long-haired', 'cats',
'have', 'soft', 'and', 'silk', 'fur', '.', 'Its', 'beautiful', '!', 'Prior',
'to', 'twitching', 'their', 'food', 'to', 'a', 'grain', 'free', 'diet', ',',
'their', 'fur', 'was', 'a', 'little', 'oily', 'with', 'dandruff', '(', 'if',
```

```
'that', 'makes', 'any', 'sense', 'LOL', ')', '.', 'It', 'this', 'time', ',',
'neither', 'cat', 'has', 'oily', 'fur', 'nor', 'do', 'they', 'have', 'any',
'pet', 'danger', 'that', 'is', 'visible', '.', 'Am', 'a', 'huge', 'fan', 'and',
'Am', 'planning', 'to', 'switch', 'my', 'puppy', 'over', 'to', 'a', 'grain',
'free', 'diet', 'as', 'soon', 'as', 'I', 'can', '.']"
```

[788]:
```
""" now that we have removed all the typos and kept only the important words,␣
↪let's create a tfidf vectorizer
and pass the words into the vectorizer to create a train_test_vector of␣
↪numbers"""

vectorizer = TfidfVectorizer()
vectorizer.fit(train_text_stripped)
train_text_vector = vectorizer.transform(train_text_stripped)
```

[789]:
```
train_text_vector[0]
```

[789]:
```
<1x9463 sparse matrix of type '<class 'numpy.float64'>'
        with 30 stored elements in Compressed Sparse Row format>
```

[790]:
```
labels_sentiment = [Sentiment_word.POSITIVE, Sentiment_word.NEGATIVE,␣
↪Sentiment_word.NEUTRAL]
labels_category = categories
```

[791]:
```
# define function to get the metrics for the predicted values

def get_all_metrics(algorithm, X, y, cv, labels):

    model = algorithm()
#     fitted_model = model.fit(X, y)

    train_pred = model_selection.cross_val_predict(model,
                                                    X,
                                                    y,
                                                    cv=cv)
#     Accuracy
    accuracy = metrics.accuracy_score(y, train_pred)

#     Precision
    precision = metrics.precision_score(y, train_pred, average = None, labels =␣
↪labels)

#     Recall
    recall = metrics.recall_score(y, train_pred, average = None, labels =␣
↪labels)

#     F1
```

```
    f1 = metrics.f1_score(y, train_pred, average = None, labels = labels)

    all_metrics = [accuracy
                   , precision
                   , recall
                   , f1
                  ]

    metric_labels = ['Accuracy','Precision', 'Recall', 'F1']
    metrics_vals = [metric for metric in all_metrics]
    zip_iterator = zip(metric_labels, metrics_vals)
    dict_metrics = dict(zip_iterator)

    return train_pred, dict_metrics

# define function to get the run time

def get_time(algorithm, X, y, cv, labels):

    start_time = time.time()
    predictions, dict_metrics = get_all_metrics(algorithm, X,
                                                y, cv, labels)
    log_time = (time.time() - start_time)
    return ('Running Time: %s' % datetime.timedelta(seconds = log_time))
```

0.0.1 Linear SVC

Sentiment

[792]:
```
svc_sentiment_pred, svc_sentiment_metrics = get_all_metrics(LinearSVC,
                                       train_text_vector,
                                       train_sentiment,
                                       10, labels_sentiment)

svc_sentiment_time = get_time(LinearSVC, train_text_vector, train_sentiment,␣
 ↪10, labels_sentiment)

print(svc_sentiment_metrics)
svc_sentiment_time
```

```
{'Accuracy': 0.8402439024390244, 'Precision': array([0.86422977, 0.60732984,
0.24050633]), 'Recall': array([0.97784343, 0.28501229, 0.06168831]), 'F1':
array([0.91753292, 0.38795987, 0.09819121])}
```

[792]: 'Running Time: 0:00:00.393328'

Category

[793]:
```
svc_category_pred, svc_category_metrics = get_all_metrics(LinearSVC,
                                        train_text_vector,
                                        train_category,
                                        10, labels_category)

svc_category_time = get_time(LinearSVC, train_text_vector, train_category, 10,␣
 ↪labels_category)

print(svc_category_metrics)
svc_category_time
```

```
{'Accuracy': 0.694390243902439, 'Precision': array([0.67020024, 0.61214374,
0.78959276, 0.78869448, 0.6119951 ]), 'Recall': array([0.69221411, 0.59951456,
0.85121951, 0.72345679, 0.60679612]), 'F1': array([0.68102932, 0.60576334,
0.81924883, 0.75466838, 0.60938452])}
```

[793]: 'Running Time: 0:00:00.694495'

0.0.2 Gaussian Naive Bayes

Sentiment

[794]:
```
gnb_sentiment_pred, gnb_sentiment_metrics = get_all_metrics(GaussianNB,
                                        train_text_vector.toarray(),
                                        train_sentiment,
                                        10, labels_sentiment)

gnb_sentiment_time = get_time(GaussianNB, train_text_vector.toarray(),␣
 ↪train_sentiment, 10, labels_sentiment)

print(gnb_sentiment_metrics)
gnb_sentiment_time
```

```
{'Accuracy': 0.48560975609756096, 'Precision': array([0.79446809, 0.11045365,
0.05470636]), 'Recall': array([0.55155096, 0.13759214, 0.22077922]), 'F1':
array([0.6510898 , 0.12253829, 0.08768536])}
```

[794]: 'Running Time: 0:00:09.725766'

Category

[795]:
```
gnb_category_pred, gnb_category_metrics = get_all_metrics(GaussianNB,
                                        train_text_vector.toarray(),
                                        train_category,
                                        10, labels_category)

gnb_category_time = get_time(GaussianNB, train_text_vector.toarray(),␣
 ↪train_category, 10, labels_category)
```

```
print(gnb_category_metrics)
gnb_category_time
```

{'Accuracy': 0.5490243902439025, 'Precision': array([0.60806452, 0.52023121, 0.77346278, 0.68960469, 0.37995965]), 'Recall': array([0.45863747, 0.4368932 , 0.58292683, 0.58148148, 0.68567961]), 'F1': array([0.52288488, 0.47493404, 0.66481224, 0.63094441, 0.48896582])}

[795]: 'Running Time: 0:00:08.679279'

0.0.3 K-Neighbors

Sentiment

[796]:
```
knc_sentiment_pred, knc_sentiment_metrics =␣
↪get_all_metrics(KNeighborsClassifier,
                                              train_text_vector,
                                              train_sentiment,
                                              10, labels_sentiment)

knc_sentiment_time = get_time(KNeighborsClassifier, train_text_vector,␣
↪train_sentiment, 10, labels_sentiment)

print(knc_sentiment_metrics)
knc_sentiment_time
```

{'Accuracy': 0.8265853658536585, 'Precision': array([0.82713936, 0.   , 0.66666667]), 'Recall': array([0.99940916, 0.   , 0.01948052]), 'F1': array([0.9051505 , 0.   , 0.03785489])}

[796]: 'Running Time: 0:00:00.580344'

Category

[797]:
```
knc_category_pred, knc_category_metrics = get_all_metrics(KNeighborsClassifier,
                                              train_text_vector,
                                              train_category,
                                              10, labels_category)

knc_category_time = get_time(KNeighborsClassifier, train_text_vector,␣
↪train_category, 10, labels_category)

print(knc_category_metrics)
knc_category_time
```

{'Accuracy': 0.2375609756097561, 'Precision': array([0.22367515, 0.19675926, 0.2394822 , 0.252   , 0.31190476]), 'Recall': array([0.39537713, 0.10315534,

0.45121951, 0.07777778, 0.15898058]), 'F1': array([0.28571429, 0.13535032,
0.31289641, 0.11886792, 0.21061093])}

[797]: 'Running Time: 0:00:00.648043'

0.0.4 Decision Tree

Sentiment

[798]:
```
dtc_sentiment_pred, dtc_sentiment_metrics =␣
↪get_all_metrics(DecisionTreeClassifier,
                                        train_text_vector,
                                        train_sentiment,
                                        10, labels_sentiment)

dtc_sentiment_time = get_time(DecisionTreeClassifier, train_text_vector,␣
↪train_sentiment, 10, labels_sentiment)

print(dtc_sentiment_metrics)
dtc_sentiment_time
```

{'Accuracy': 0.7746341463414634, 'Precision': array([0.86725917, 0.2877907 ,
0.19402985]), 'Recall': array([0.89364845, 0.24324324, 0.16883117]), 'F1':
array([0.88025607, 0.26364847, 0.18055556])}

[798]: 'Running Time: 0:00:04.902174'

Category

[799]:
```
dtc_category_pred, dtc_category_metrics =␣
↪get_all_metrics(DecisionTreeClassifier,
                                        train_text_vector,
                                        train_category,
                                        10, labels_category)

dtc_category_time = get_time(DecisionTreeClassifier, train_text_vector,␣
↪train_category, 10, labels_category)

print(dtc_category_metrics)
dtc_category_time
```

{'Accuracy': 0.4892682926829268, 'Precision': array([0.40898876, 0.35316456,
0.63537118, 0.60576923, 0.43814433]), 'Recall': array([0.44282238, 0.33859223,
0.7097561 , 0.54444444, 0.41262136]), 'F1': array([0.42523364, 0.34572491,
0.67050691, 0.57347204, 0.425   ])}

[799]: 'Running Time: 0:00:04.981727'

0.0.5 Gradient Boost

Sentiment

[800]:
```
gbc_sentiment_pred, gbc_sentiment_metrics =␣
↪get_all_metrics(GradientBoostingClassifier,
                                               train_text_vector,
                                               train_sentiment,
                                               10, labels_sentiment)

gbc_sentiment_time = get_time(GradientBoostingClassifier, train_text_vector,␣
↪train_sentiment, 10, labels_sentiment)

print(gbc_sentiment_metrics)
gbc_sentiment_time
```

```
{'Accuracy': 0.8341463414634146, 'Precision': array([0.84720812, 0.625      ,
0.30357143]), 'Recall': array([0.98611521, 0.15970516, 0.05519481]), 'F1':
array([0.91139932, 0.25440313, 0.09340659])}
```

[800]: 'Running Time: 0:01:45.131691'

Category

[801]:
```
gbc_category_pred, gbc_category_metrics =␣
↪get_all_metrics(GradientBoostingClassifier,
                                               train_text_vector,
                                               train_category,
                                               10, labels_category)

gbc_category_time = get_time(GradientBoostingClassifier, train_text_vector,␣
↪train_category, 10, labels_category)

print(gbc_category_metrics)
gbc_category_time
```

```
{'Accuracy': 0.6163414634146341, 'Precision': array([0.61892247, 0.41603631,
0.84698609, 0.84991568, 0.58429858]), 'Recall': array([0.5729927 , 0.66747573,
0.66829268, 0.62222222, 0.55097087]), 'F1': array([0.59507265, 0.51258155,
0.74710293, 0.71846044, 0.56714553])}
```

[801]: 'Running Time: 0:03:02.872013'

0.0.6 Balanced Train Set

Sentiment

[912]:
```
train_set_sentiment = pd.Series(train_sentiment)
train_set_neutral = train_set_sentiment[train_set_sentiment == 'NEUTRAL']
```

```
train_set_positive = train_set_sentiment[train_set_sentiment == 'POSITIVE'][0:
 ↪len(train_set_neutral)]
train_set_negative = train_set_sentiment[train_set_sentiment == 'NEGATIVE'][0:
 ↪len(train_set_neutral)]
train_sentiment_final = list(chain(train_set_positive, train_set_negative,␣
 ↪train_set_neutral))
```

[913]:
```
len(train_sentiment_final)
```

[913]: 924

[914]:
```
# same thing for the text
train_text_series = pd.Series(train_text_stripped)
train_text_neutral = train_text_series[train_set_sentiment == 'NEUTRAL']
train_text_positive = train_text_series[train_set_sentiment == 'POSITIVE'][0:
 ↪len(train_set_neutral)]
train_text_negative = train_text_series[train_set_sentiment == 'NEGATIVE'][0:
 ↪len(train_set_neutral)]
train_text_sentiment = list(chain(train_text_positive, train_text_negative,␣
 ↪train_text_neutral))
```

[915]:
```
len(train_text_sentiment)
```

[915]: 924

[916]:
```
vectorizer_sentiment_final = TfidfVectorizer()
train_sentiment_vector = vectorizer_sentiment_final.
 ↪fit_transform(train_text_sentiment)
```

[917]:
```
train_sentiment_vector
```

[917]:
```
<924x4784 sparse matrix of type '<class 'numpy.float64'>'
        with 27382 stored elements in Compressed Sparse Row format>
```

Category

[918]:
```
# train_category less unbalanced, but for the sake of completeness, let's do␣
 ↪the same with category
pd.Series(train_category).value_counts()
```

[918]:
```
OFFICE         824
HOME           824
ELECTRONICS    822
MOVIES         820
PETS           810
dtype: int64
```

[919]:
```
train_set_category = pd.Series(train_category)
train_set_pets = train_set_category[train_set_category == 'PETS']
train_set_movies = train_set_category[train_set_category == 'MOVIES'][0:
 ↪len(train_set_pets)]
train_set_electronics = train_set_category[train_set_category ==␣
 ↪'ELECTRONICS'][0:len(train_set_pets)]
train_set_home = train_set_category[train_set_category == 'HOME'][0:
 ↪len(train_set_pets)]
train_set_office = train_set_category[train_set_category == 'OFFICE'][0:
 ↪len(train_set_pets)]
train_category_final = list(chain(train_set_pets, train_set_movies,␣
 ↪train_set_electronics, train_set_home,
                                  train_set_office))
```

[920]:
```
len(train_category_final)
```

[920]: 4050

[921]:
```
# same thing for the text
# train_text_series = pd.Series(train_text_stripped)
train_text_pets = train_text_series[train_set_category == 'PETS']
train_text_movies = train_text_series[train_set_category == 'MOVIES'][0:
 ↪len(train_set_pets)]
train_text_electronics = train_text_series[train_set_category ==␣
 ↪'ELECTRONICS'][0:len(train_set_pets)]
train_text_home = train_text_series[train_set_category == 'HOME'][0:
 ↪len(train_set_pets)]
train_text_office = train_text_series[train_set_category == 'OFFICE'][0:
 ↪len(train_set_pets)]
train_text_category = list(chain(train_text_pets, train_text_movies,␣
 ↪train_text_electronics, train_text_home,
                                train_text_office))
```

[922]:
```
len(train_text_category)
```

[922]: 4050

[923]:
```
vectorizer_category_final = TfidfVectorizer()
train_category_vector = vectorizer_category_final.
 ↪fit_transform(train_text_category)
```

[924]:
```
train_category_vector
```

[924]:
```
<4050x9428 sparse matrix of type '<class 'numpy.float64'>'
        with 100589 stored elements in Compressed Sparse Row format>
```

0.0.7 Test Set

[925]:
```
test_text = [x.text for x in test]
test_sentiment = [x.sentiment for x in test]
test_category = [x.category for x in test]
```

[926]:
```
""" spell check """

tb_test_text = [TextBlob(test_text_unit) for test_text_unit in test_text]
tb_test_text_correct = [tb_test_text_unit.correct() for tb_test_text_unit in␣
↪tb_test_text]
```

[927]:
```
""" stopwords removal"""

test_words = [word_tokenize(str(tb_test_text_correct_unit)) for␣
↪tb_test_text_correct_unit in tb_test_text_correct]
test_text_stripped = []
[test_text_stripped.append(str(word)) for word in test_words if word not in␣
↪stop_words]
test_text_stripped[1]
```

[927]:
```
"['good', 'one']"
```

0.0.8 Create Balanced Test Set

Sentiment

[928]:
```
pd.Series(test_sentiment).value_counts()
```

[928]:
```
POSITIVE    739
NEGATIVE     98
NEUTRAL      63
dtype: int64
```

[929]:
```
test_set_sentiment = pd.Series(test_sentiment)
test_set_neutral = test_set_sentiment[test_set_sentiment == 'NEUTRAL']
test_set_positive = test_set_sentiment[test_set_sentiment == 'POSITIVE'][0:
↪len(test_set_neutral)]
test_set_negative = test_set_sentiment[test_set_sentiment == 'NEGATIVE'][0:
↪len(test_set_neutral)]
test_sentiment_final = list(chain(test_set_positive, test_set_negative,␣
↪test_set_neutral))
```

[930]:
```
len(test_sentiment_final)
```

[930]:
```
189
```

[931]:
```python
# same thing for the text
test_text_series = pd.Series(test_text_stripped)
test_text_neutral = test_text_series[test_set_sentiment == 'NEUTRAL']
test_text_positive = test_text_series[test_set_sentiment == 'POSITIVE'][0:
 ↪len(test_set_neutral)]
test_text_negative = test_text_series[test_set_sentiment == 'NEGATIVE'][0:
 ↪len(test_set_neutral)]
test_text_sentiment = list(chain(test_text_positive, test_text_negative,␣
 ↪test_text_neutral))
```

[932]:
```python
len(test_text_sentiment)
```

[932]: 189

[933]:
```python
test_sentiment_vector = vectorizer_sentiment_final.
 ↪transform(test_text_sentiment)
```

[934]:
```python
test_sentiment_vector
```

[934]:
```
<189x4784 sparse matrix of type '<class 'numpy.float64'>'
        with 4748 stored elements in Compressed Sparse Row format>
```

Category

[935]:
```python
pd.Series(test_category).value_counts()
```

[935]:
```
PETS           190
MOVIES         180
ELECTRONICS    178
OFFICE         176
HOME           176
dtype: int64
```

[936]:
```python
test_set_category = pd.Series(test_category)
test_set_home = test_set_category[test_set_category == 'HOME']
test_set_movies = test_set_category[test_set_category == 'MOVIES'][0:
 ↪len(test_set_home)]
test_set_electronics = test_set_category[test_set_category == 'ELECTRONICS'][0:
 ↪len(test_set_home)]
test_set_pets = test_set_category[test_set_category == 'PETS'][0:
 ↪len(test_set_home)]
test_set_office = test_set_category[test_set_category == 'OFFICE'][0:
 ↪len(test_set_home)]
test_category_final = list(chain(test_set_home, test_set_movies,␣
 ↪test_set_electronics, test_set_pets,
                                test_set_office))
```

[937]:
```
len(test_category_final)
```

[937]: 880

[938]:
```
# same for text
# test_text_series = pd.Series(test_text_stripped)
test_text_home = test_text_series[test_set_category == 'HOME']
test_text_movies = test_text_series[test_set_category == 'MOVIES'][0:
↪len(test_set_home)]
test_text_electronics = test_text_series[test_set_category == 'ELECTRONICS'][0:
↪len(test_set_home)]
test_text_pets = test_text_series[test_set_category == 'PETS'][0:
↪len(test_set_home)]
test_text_office = test_text_series[test_set_category == 'OFFICE'][0:
↪len(test_set_home)]
test_text_category = list(chain(test_text_home, test_text_movies,␣
↪test_text_electronics, test_text_pets,
                               test_text_office))
```

[939]:
```
len(test_text_category)
```

[939]: 880

[940]:
```
test_category_vector = vectorizer_category_final.transform(test_text_category)
test_category_vector
```

[940]:
```
<880x9428 sparse matrix of type '<class 'numpy.float64'>'
        with 19926 stored elements in Compressed Sparse Row format>
```

0.0.9 Improve Performance

Sentiment

[941]:
```
# while the LinearSVC model was used above, included the option of
# a radial-based model, with different C parameters

parameters = {'kernel':('linear', 'rbf'), 'C':[0.5,1,1.5,2,4,8,12,16,24,32]}
svc_model = svm.SVC(gamma = 'auto')
new_classifier_sentiment = GridSearchCV(svc_model, parameters, cv=10)
new_classifier_sentiment.fit(train_sentiment_vector, train_sentiment_final)
```

[941]:
```
GridSearchCV(cv=10, estimator=SVC(gamma='auto'),
             param_grid={'C': [0.5, 1, 1.5, 2, 4, 8, 12, 16, 24, 32],
                         'kernel': ('linear', 'rbf')})
```

[942]:
```
results_sentiment = new_classifier_sentiment.cv_results_
df_sentiment = pd.DataFrame(results)
new_classifier_sentiment.score
```

```
df_sentiment
```

[942]:

```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time param_C  \
0        1.571508      0.091935         0.133049        0.010316     0.5
1        3.021482      0.194080         0.212313        0.016729     0.5
2        1.683724      0.114735         0.144912        0.012823       1
3        2.744954      0.083488         0.207093        0.007159       1
4        1.596129      0.039458         0.133008        0.003731     1.5
5        3.132797      0.173041         0.218473        0.013525     1.5
6        1.622083      0.064521         0.135691        0.011898       2
7        3.162214      0.071986         0.206866        0.009852       2
8        1.577540      0.038767         0.129638        0.003455       4
9        3.257411      0.144345         0.219786        0.020770       4
10       1.685450      0.096046         0.136994        0.008419       8
11       3.178028      0.132876         0.201677        0.013755       8
12       1.504958      0.020787         0.123612        0.001820      12
13       3.185897      0.130706         0.200711        0.006994      12
14       1.602226      0.069948         0.132286        0.007956      16
15       3.260216      0.186983         0.215495        0.023361      16
16       1.649522      0.098039         0.132492        0.012175      24
17       3.249617      0.165405         0.208534        0.013842      24
18       1.568800      0.080353         0.127187        0.005424      32
19       3.296683      0.176661         0.218499        0.037127      32

   param_kernel                              params  split0_test_score  \
0        linear  {'C': 0.5, 'kernel': 'linear'}           0.834146
1           rbf     {'C': 0.5, 'kernel': 'rbf'}           0.826829
2        linear    {'C': 1, 'kernel': 'linear'}           0.843902
3           rbf       {'C': 1, 'kernel': 'rbf'}           0.834146
4        linear  {'C': 1.5, 'kernel': 'linear'}           0.843902
5           rbf     {'C': 1.5, 'kernel': 'rbf'}           0.836585
6        linear    {'C': 2, 'kernel': 'linear'}           0.826829
7           rbf       {'C': 2, 'kernel': 'rbf'}           0.839024
8        linear    {'C': 4, 'kernel': 'linear'}           0.809756
9           rbf       {'C': 4, 'kernel': 'rbf'}           0.836585
10       linear    {'C': 8, 'kernel': 'linear'}           0.819512
11          rbf       {'C': 8, 'kernel': 'rbf'}           0.836585
12       linear   {'C': 12, 'kernel': 'linear'}           0.819512
13          rbf      {'C': 12, 'kernel': 'rbf'}           0.836585
14       linear   {'C': 16, 'kernel': 'linear'}           0.821951
15          rbf      {'C': 16, 'kernel': 'rbf'}           0.836585
16       linear   {'C': 24, 'kernel': 'linear'}           0.819512
17          rbf      {'C': 24, 'kernel': 'rbf'}           0.836585
18       linear   {'C': 32, 'kernel': 'linear'}           0.812195
19          rbf      {'C': 32, 'kernel': 'rbf'}           0.836585

    split1_test_score  split2_test_score  split3_test_score  \
```

0	0.829268	0.831707	0.831707
1	0.826829	0.826829	0.826829
2	0.848780	0.839024	0.856098
3	0.829268	0.831707	0.831707
4	0.856098	0.839024	0.856098
5	0.836585	0.841463	0.846341
6	0.846341	0.843902	0.853659
7	0.841463	0.841463	0.851220
8	0.836585	0.819512	0.843902
9	0.843902	0.841463	0.848780
10	0.821951	0.817073	0.836585
11	0.843902	0.841463	0.848780
12	0.819512	0.817073	0.834146
13	0.843902	0.841463	0.848780
14	0.821951	0.817073	0.836585
15	0.843902	0.841463	0.848780
16	0.826829	0.814634	0.843902
17	0.843902	0.841463	0.848780
18	0.821951	0.812195	0.839024
19	0.843902	0.841463	0.848780

	split4_test_score	split5_test_score	split6_test_score \
0	0.826829	0.831707	0.824390
1	0.826829	0.821951	0.824390
2	0.836585	0.843902	0.836585
3	0.829268	0.826829	0.826829
4	0.836585	0.853659	0.846341
5	0.831707	0.834146	0.829268
6	0.834146	0.843902	0.841463
7	0.834146	0.836585	0.831707
8	0.804878	0.831707	0.836585
9	0.831707	0.836585	0.836585
10	0.790244	0.836585	0.824390
11	0.831707	0.836585	0.836585
12	0.792683	0.834146	0.824390
13	0.831707	0.836585	0.836585
14	0.790244	0.831707	0.824390
15	0.831707	0.836585	0.836585
16	0.790244	0.843902	0.819512
17	0.831707	0.836585	0.836585
18	0.785366	0.841463	0.821951
19	0.831707	0.836585	0.836585

	split7_test_score	split8_test_score	split9_test_score	mean_test_score \
0	0.824390	0.821951	0.821951	0.827805
1	0.824390	0.824390	0.824390	0.825366
2	0.829268	0.831707	0.831707	0.839756

3	0.826829	0.824390	0.824390	0.828537
4	0.843902	0.834146	0.846341	0.845610
5	0.831707	0.821951	0.829268	0.833902
6	0.831707	0.829268	0.848780	0.840000
7	0.834146	0.826829	0.829268	0.836585
8	0.809756	0.819512	0.851220	0.826341
9	0.834146	0.824390	0.829268	0.836341
10	0.804878	0.809756	0.841463	0.820244
11	0.834146	0.824390	0.829268	0.836341
12	0.802439	0.809756	0.831707	0.818537
13	0.834146	0.824390	0.829268	0.836341
14	0.802439	0.809756	0.829268	0.818537
15	0.834146	0.824390	0.829268	0.836341
16	0.809756	0.807317	0.826829	0.820244
17	0.834146	0.824390	0.829268	0.836341
18	0.812195	0.809756	0.829268	0.818537
19	0.834146	0.824390	0.829268	0.836341

	std_test_score	rank_test_score
0	0.004253	13
1	0.001618	15
2	0.008019	3
3	0.003095	12
4	0.007402	1
5	0.006494	11
6	0.008533	2
7	0.006724	4
8	0.015067	14
9	0.006764	5
10	0.015037	17
11	0.006764	5
12	0.013053	18
13	0.006764	5
14	0.013457	18
15	0.006764	5
16	0.015543	16
17	0.006764	5
18	0.015433	18
19	0.006764	5

[943]:
```
df['params'][df['rank_test_score'] == 1]
```

[943]:
```
4    {'C': 1.5, 'kernel': 'linear'}
Name: params, dtype: object
```

Category

[944]:
```
new_classifier_category = GridSearchCV(svc_model, parameters, cv=10)

new_classifier_category.fit(train_category_vector, train_category_final)
results_category = new_classifier_category.cv_results_
df_category = pd.DataFrame(results_category)
new_classifier_category.score
df_category['params'][df_category['rank_test_score'] == 1]
```

[944]:
```
2    {'C': 1, 'kernel': 'linear'}
Name: params, dtype: object
```

0.0.10 Fit Final Train Data on Fine-tuned Models

Sentiment

[945]:
```
svc_sentiment_final = svm.SVC(gamma = 'auto', C = 1.5, kernel = 'linear')
svc_sentiment_final.fit(train_sentiment_vector, train_sentiment_final)
```

[945]:
```
SVC(C=1.5, gamma='auto', kernel='linear')
```

Category

[946]:
```
svc_category_final = svm.SVC(gamma ='auto', C = 1, kernel = 'linear')
svc_category_final.fit(train_category_vector, train_category_final)
```

[946]:
```
SVC(C=1, gamma='auto', kernel='linear')
```

0.0.11 Evaluation

[947]:
```
# Create a function using adapted code from sklearn's website and seralouk's
# stackoverflow's post on 07-19-18 - thank you!

def plot_multiple_roc(model,X_train, y_train, X_test, y_test, labels,␣
 ↪n_classes):

    classifier = OneVsRestClassifier(model)
    y_train_binary = preprocessing.label_binarize(y_train,
                                                  classes = labels)
    y_test_binary = preprocessing.label_binarize(y_test,
                                                 classes = labels)
    y_pred_dec = classifier.fit(X_train, y_train_binary).
 ↪decision_function(X_test)

    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(0,n_classes):
```

```
        fpr[i], tpr[i], _ = metrics.roc_curve(y_test_binary[:,i], y_pred_dec[:
↪,i])
        roc_auc[i] = metrics.auc(fpr[i], tpr[i])
    colors = cycle(['blue', 'red', 'green', 'orange', 'black'])
    for i, color in zip(range(0,n_classes), colors):
        plt.plot(fpr[i], tpr[i], color = color,
                label= 'ROC curve of class {0} (area = {1:0.2f})'
                ''.format(i, roc_auc[i]))
    plt.plot([0,1], [0,1], 'k--')
    plt.xlim([-0.05, 1.0])
    plt.ylim([0, 1.05])
    plt.title('ROC Curve for Each Class')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc = 'lower right')
    plt.show()
```

Sentiment

[948]:
```
sentiment_prediction = svc_sentiment_final.predict(test_sentiment_vector)
metrics.f1_score(test_sentiment_final,sentiment_prediction, average = None)
```

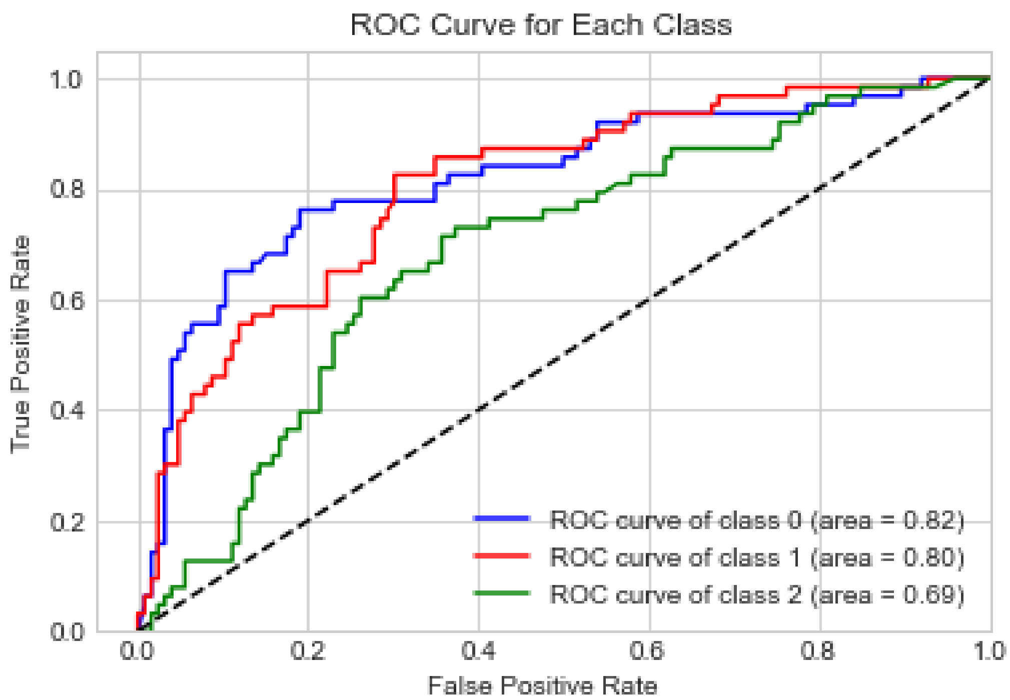
[948]:
```
array([0.58267717, 0.53125   , 0.66666667])
```

[949]:
```
svc_sentiment_final_cal = CalibratedClassifierCV(svc_sentiment_final)
svc_sentiment_final_cal.fit(train_sentiment_vector, train_sentiment_final)
sentiment_pred_prob = svc_sentiment_final_cal.
↪predict_proba(test_sentiment_vector)
```

[950]:
```
metrics.roc_auc_score(test_sentiment_final, sentiment_pred_prob, multi_class =␣
↪'ovr')
```

[950]:
```
0.7633324934912236
```

[951]:
```
plot_multiple_roc(svm.SVC(gamma = 'auto', C = 1.5, kernel = 'linear',
                          probability = True),
                          train_sentiment_vector, train_sentiment_final,
                          test_sentiment_vector, test_sentiment_final,
                          labels_sentiment, 3)
```



Category

[952]:
```
category_prediction = svc_category_final.predict(test_category_vector)
metrics.f1_score(test_category_final,category_prediction, average = None)#␣
↪metrics.score
```

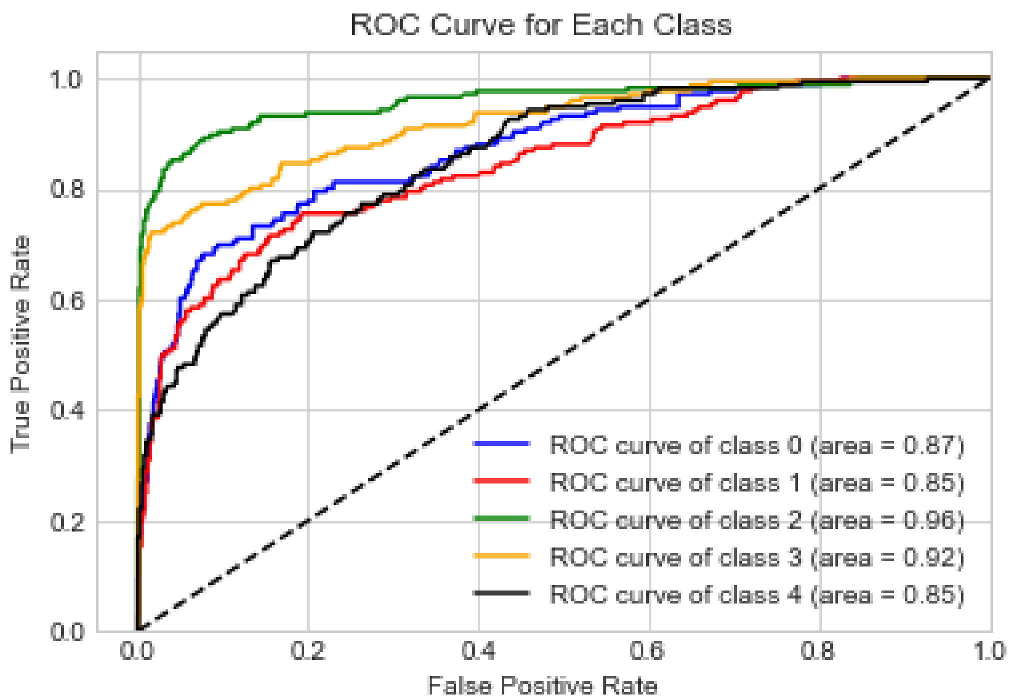
[952]: array([0.63586957, 0.60055096, 0.80779944, 0.57381616, 0.79099678])

[953]:
```
svc_category_final_cal = CalibratedClassifierCV(svc_category_final)
svc_category_final_cal.fit(train_category_vector, train_category_final)
category_pred_prob = svc_category_final_cal.predict_proba(test_category_vector)
```

[954]:
```
metrics.roc_auc_score(test_category_final, category_pred_prob, multi_class =␣
↪'ovr')
```

[954]: 0.872527117768595

[955]:
```
plot_multiple_roc(svm.SVC(gamma = 'auto', C = 1, kernel = 'linear',
                          probability = True),
                          train_category_vector, train_category_final,
                          test_category_vector, test_category_final,
                          labels_category, 5)
```

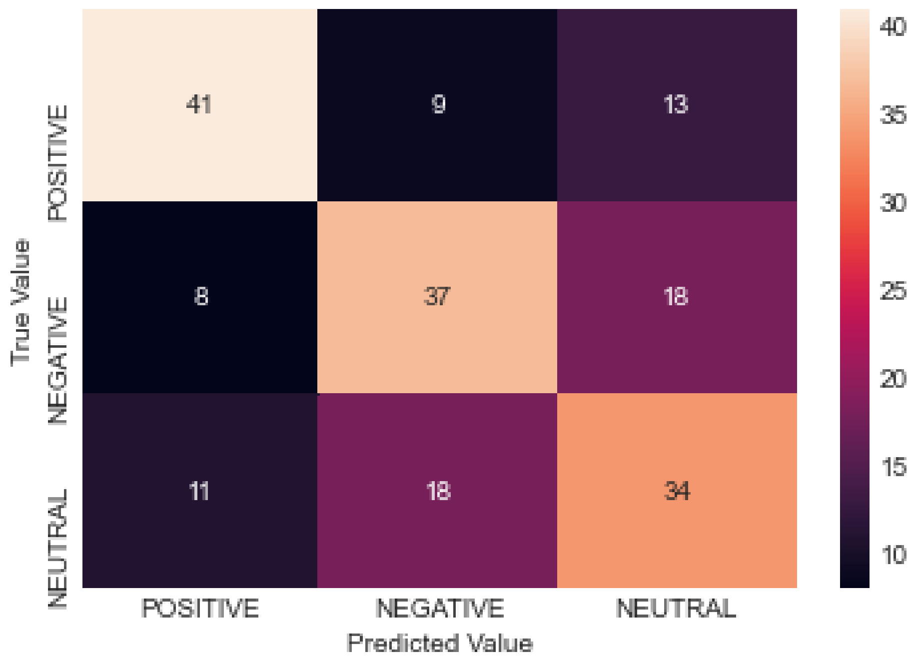


0.0.12 Confusion Matrix

Sentiment

[956]:
```
cm_sentiment = confusion_matrix(test_sentiment_final, sentiment_prediction,␣
 ↪labels=labels_sentiment)
df_cm_sentiment = pd.DataFrame(cm_sentiment, index=labels_sentiment,␣
 ↪columns=labels_sentiment)
sns.heatmap(df_cm_sentiment, annot=True, fmt='d')
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
```

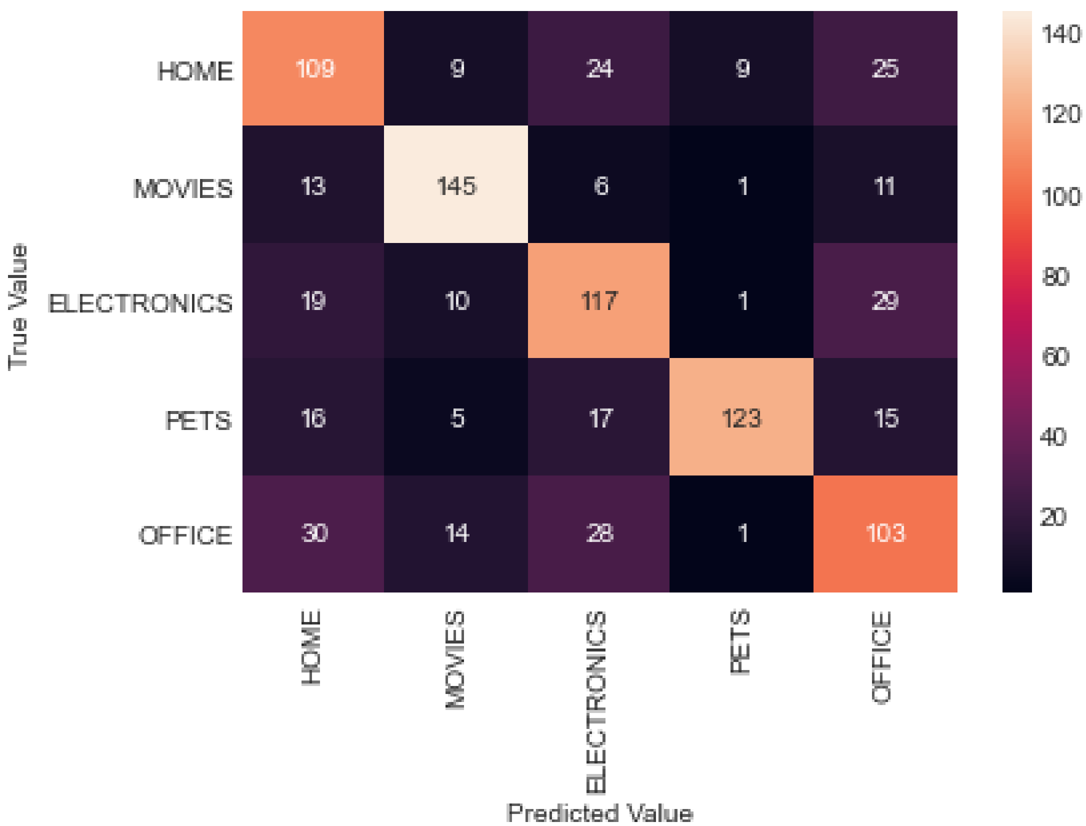
[956]: Text(37.5, 0.5, 'True Value')



Category

[957]:
```
cm_category = confusion_matrix(test_category_final, category_prediction,␣
↪labels=['HOME', 'MOVIES', 'ELECTRONICS',
                                                                        ␣
↪'PETS', 'OFFICE'])
df_cm_category = pd.DataFrame(cm_category, index=['HOME', 'MOVIES',␣
↪'ELECTRONICS', 'PETS', 'OFFICE'],
                              columns=['HOME', 'MOVIES', 'ELECTRONICS', 'PETS',␣
↪'OFFICE'])
sns.heatmap(df_cm_category, annot=True, fmt='d')
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
```

[957]: Text(37.5, 0.5, 'True Value')



0.0.13 Possible Improvements

[958]:
```
# Big problem is the lack of data used
# Due to limited memory on my laptop, did not use as many reviews as I would␣
↪have liked
```

[959]:
```
# Maybe if I had removed the neutral reviews, I would have obtained a more␣
↪clear positive/negative classification
```

[960]:
```
# Incorporate some more advanced NLP features such as word vectors from␣
↪spaCy(although might not be ideal
# for very lengthy reviews) and the removal of specific characters like␣
↪punctuation marks using regex or spaCy's Matcher
```

[961]:
```
# An important improvement would certainly be to create a balanced train set␣
↪BEFORE
# training each of the models, especially for the sentiment analysis
```

[962]:
```
# In addtion, another potential issue that comes to mind is that I used␣
↪accuracy as
```

```
# the metric to decide which model I should go ahead with, despite having
# used unbalanced data while training the models
# The f1 would have been a more appropriate metric would have been a more
# appropriate metric in this regard.
```