Phase 4 Reccomendation Modeling

Authors: Fennec Nighintgale, Matthew Lipman

In this notebook we will be be using the CRISP-DM process to walk through using Faiss K-means algorithm and naive Bayes to predict sentiment on space travel related tweets and YouTube comments.

```
# !pip install faiss-gpu
# !pip install vaderSentiment
```

```
# from google.colab import drive
# drive.mount('/content/drive')
```

```
import faiss
import math
import pandas as pd
import numpy as np
from datetime import datetime
```

```
import scipy
from scipy.linalg import norm
from scipy.sparse import csr_matrix, coo_matrix, hstack, vstack
from imblearn.combine import SMOTEENN
```

```
from textblob import TextBlob
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

```
from sklearn.decomposition import TruncatedSVD
from sklearn.naive_bayes import GaussianNB, BernoulliNB, ComplementNB
from sklearn.preprocessing import MaxAbsScaler

from sklearn.feature_extraction.text import TfidfVectorizer, \
                                              CountVectorizer
from sklearn.metrics import silhouette_samples, confusion_matrix, \
                      accuracy_score, recall_score, precision_score, \
                      roc_curve, auc, roc_auc_score, confusion_matrix, \
                      classification_report, f1_score, r2_score
from sklearn.model_selection import GridSearchCV, cross_val_score, \
                                    train_test_split, RandomizedSearchCV
```

```
from mlxtend.plotting import plot_confusion_matrix
```

```
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib import cm
```

Functions

```
def get_sentiment_an(df):
  """
  Given a pandas dataframe, this will shuffle it & get sentiment
  analysis values from TextBlob & VaderSentiment & assign columns inplace.
  Will return completed dataframe & dataframe columns we'll be using for
  modeling as a coo_matrix.
                Columns returned:
  'polarity' : mean of text sentiment (TextBlob)
  'neg': negative sentiment value (VaderSentiment)
  'pos': positive sentiment value (VaderSentiment)
  'neu': neutral sentiment value (VaderSeniment)
  """
  # shuffling our dataframe so data is no longer sorted
  df = df.sample(frac = 1)
  df = df.reset_index().rename(columns={'index':'id'})
  # getting rid of all unnessecary columns
  df = df[['favorite_count', 'repost_count', 'text', 'id']]
  # feature generation using polarization
  df['polarity'] = df['text'].apply(lambda x:
                                    TextBlob(x).sentiment.polarity)
  # feature generation using intensity analyzer
  analyzer = SentimentIntensityAnalyzer()
  df['pol'] = ''
  for i in range(len(df)):
    sentence = df.at[i, 'text']
    df.at[i, 'pol'] = analyzer.polarity_scores(sentence)
  for i in range(len(df)):
    for h in ['neg', 'neu', 'pos']:
      df.at[i, h] = float(df['pol'][i][h])
  #turning our other variables into a matrix to combine our features
  maxab = MaxAbsScaler()
  drop = ['text', 'pol', 'id']
  hstack = maxab.fit_transform(coo_matrix(df.drop(labels = drop, axis=1)))
  return df, hstack
```

```
def word_transform(df):
  """
  Pass in a pandas dataframe and this will return a coo_matrix
  featuring the TFIDF vectorization of your text, with a maximum
  of 100 features of single words and 100 features of bigrams.
  """
  sing = TfidfVectorizer(max_features=100).fit_transform(df['text'])
```

```
    bi = TfidfVectorizer(ngram_range = (2, 2),
                         max_features=100).fit_transform(df['text'])
    vect = coo_matrix(np.append(bi.toarray(), sing.toarray(), axis=1))
    return vect
```

```
def svd_pca(vect, haystack):
    """
    Pass in your coo_matrix of your word vectors & the scaled
    features from your dataframe this will perform Principal Component
    Analysis in order to keep your most relevant 100 features, calculated
     in chunks to avoid RAM issues. It will return a numpy
     array containing all of the data for your model.
    """
    svd = TruncatedSVD(n_components=95, n_iter=250000,
                       random_state=0, algorithm='arpack').fit(vect)
    # splitting it so we never keep it all in RAM at once as an array
    splits = []
    shape = vect.shape[1]
    tbs = vect
    # since you cant directly index a matrix the easiest way is train_test
    # chose 50000 since my computer can handle it
    while shape > 50000:
        shape -= 50000
        # first, run through the whole matrix
        split1, split2 = train_test_split(tbs, random_state=0,
                                          train_size=shape)
        #all of our leftover rows (not in the 50k) are in split2
        tbs = split2
        # put the chunk of our matrix into a list to iterate over later
        splits.append(split1)
    # append the remainder (whatever was left < 50k) to the end of our list
    splits.append(tbs)
    # start a new matrix by fit transforming our first chunk
    init = coo_matrix(svd.fit_transform(splits[0]))
    # start a loop that will fit transform and stack matricies together
    for item in splits[1:]:
        # fit transform will return it as a dense array
        init = vstack([init, coo_matrix(svd.fit_transform(item))])
    # combine our text matrix & our repost/favorite matrix
    return np.append(haystack.toarray(),
                     init.toarray(), axis=1).astype(np.float32)
```

```
def vect_and_stck(dataframe, first_run):
    """
    Pass in a DataFrame & this function will preprocess and return your
    dataframe & all of the columns nessecary to model as an array
    The first step is to get sentiment analysis scores,
    which is only nessecary the first time you run this, otherwise
    they can be turned to False to save time,
    sentiment analysis is followed by TDIDF vectorizaton on
```

```
  single words & bigrams, and lastly it ends with PCA to reduce
  our dimensions to a reasonable amount. Make drop_clust True to
  return a copy of your dataframe with no cluster column before
  rerunning your model on it.
  """
  if first_run:
    # getting sentiment analysis columns the first time
    dataframe, haystack = get_sentiment_an(dataframe)
  else:
  # keeping only of the columns we need
    clmn = ['neg', 'neu', 'pos', 'polarity',
            'repost_count', 'favorite_count']
    haystack = coo_matrix(dataframe[clmn])
  # word vectorizing
  vect = word_transform(dataframe)
  # return our matrix with PCA done to reduce
  # dimensionality to 100 features as was reccomended
  # for PCA in the documentation
  return svd_pca(vect, haystack), dataframe
```

```
def plt_elbow(matrix, df, display):
  """
  Plots your clusters from k 1-15 and returns the best one as an int,
  if display is true it will display the elbow plot, if not it will
  just return your value.
  """
  max = 15 if display else 5
  K = range(1,max+1)
  # start numpy arrays to store results in
  inrt = np.zeros(max)
  diff = np.zeros(max)
  diff2 = np.zeros(max)
  diff3 = np.zeros(max)
  for k in K:
    kmeans, predictions, not_k = clusterizer(matrix, df, False, k=k)
    inrt[k - 1] = kmeans.obj[-1]
    # first difference
    if k > 1:
        diff[k - 1] = inrt[k - 1] - inrt[k - 2]
    # second difference
    elif k > 2:
        diff2[k - 1] = diff[k - 1] - diff[k - 2]
    # third difference
    elif k > 3:
        diff3[k - 1] = diff2[k - 1] - diff2[k - 2]
  # use differences & numpy argmin to determine best cluster
  elbow = np.argmin(diff3[3:]) + 3
  if display:
    print(f'Elbow {str(elbow)}')
    plt.plot(K, inertias, 'b*-')
    plt.plot(K[elbow-1], inertias[elbow-1], marker='o',
```

```
             markersize=12, markeredgewidth=2, markeredgecolor='r')
    plt.ylabel('Inertia')
    plt.xlabel('K')
    plt.show()
  return int(elbow)
```

```
def chunked_samples(dataframe, stack, fit_predict, drop):
  """
  Given your dataframe, feature array/matrix & predictions
  Computes Silhouette Scores using sklearn and chunking
  the data into sets of 75000 and returns a deep copy of
  your dataframe with the silhouette scores in the column
  'sil' Scores will not be perfect as this function chunks them, and
  therefore cannot get complete pairwise distances
  but without chunking silhouette scores cannot be run on
  any computer or service we can find and it will be pretty close.
  Use drop = True to calculate the z-score of your silhouette scores
  and get rid of the bottom 10%, assuming they're likely to be outliers.
  """
  start = 0
  clust = 75000
  # since this is about the most my computer can handle we'll
  # stick to a cap of 75,000
  for i in range(math.ceil(stack.shape[0]/75000)):
    # gathering our chunks
    df = dataframe[start:start+clust].reset_index(drop=True).copy()
    fp = np.ravel(fit_predict[start:start+clust][0:])
    # making sure our dtypes are correct
    if type(stack) != np.ndarray:
      hstk = stack.toarray()[start:start+clust][0:]
    else:
      hstk = stack[start:start+clust][0:]
    # getting our silhouette comments
    df['sil'] = silhouette_samples(hstk, fp)
    # we want to return our dataframe at the end so
    # on our first ieration we'll start with a chunk
    # and append to it every future iteration
    head = df if i == 0 else head.append(df)
    if drop:
      # if drop = True, this will calculate and drop the
      # lowest ~10th percentile of our silhouette scores
      # assuming they're likely to be outliers
      mean = head['sil'].mean()
      std = head['sil'].std()
      for j in head.index:
        head.at[j, 'silstd'] = (head.at[j, 'sil'] - mean)/std
      head = head.loc[head['silstd'] > -1.28].drop('silstd', 1)
    head.reset_index(drop=True, inplace=True)
    start += 75000
  return head
```

```python
def quick_silhouette(dataframe, stack, predict):
  """
  Pass in your dataframe, your feature array/matrix,
  & your predictions, and this will use chunked_samples
  to get your score & try to quickly form a matplotlib visual
  Scores won't be 100% accurate as you cannot get pairwise distances
  with chunking, but they use a sample size of 75,000 so they should
  be close, and the decrease in accuracy is well worth the 6-7 hours
  in speed it saves
  """
  #first we calculate our silhouette scores
  dataframe = chunked_samples(dataframe, stack, predict, False)
  # convert our predictions and their scores to numpy arrays
  predictions = dataframe.cluster.to_numpy()
  silhouette_vals = dataframe.sil.to_numpy()
  # get our labels
  labels = np.unique(predictions)
  # start plotting
  y_ax_lower, y_ax_upper = 0, 0
  yticks = []
  # go through our labels and assign our predictions & values to them
  for i, c in enumerate(labels):
    c_silhouette_vals = silhouette_vals[predictions == c]
    # sort to keeo our clusters together
    c_silhouette_vals.sort()
    # change plot size & ticks accordingly
    y_ax_upper += len(c_silhouette_vals)
    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    # get colors & plot!
    color = cm.jet(float(i) / labels.shape[0])
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals,
                height=1.0, color=color)
    y_ax_lower += len(c_silhouette_vals)
  # find our average and visualize it
  plt.axvline(np.mean(silhouette_vals), color="red", linestyle="--")
  # fix up our labels and ticks
  plt.yticks(yticks, labels + 1)
  plt.ylabel('Cluster')
  plt.xlabel('Silhouette Coefficient')
  plt.show()
```

```python
def get_imb_clstr(dataframe):
  """
  Pass in your dataframe and get the cluster with the most values in it
  returned as an interger
  """
  clstr = dataframe['cluster'].value_counts()
  # largest cluster gets returned at the first index
  clstr = int(clstr.to_frame().reset_index()['index'][0])
  return clstr
```

```
def reset_clusters(new_predict, dataframe):
  """
  Pass in your new predictions and your dataframe
  and it will renumber all of your clusters.
  Be sure to use this before making visuals, if you don't
  clusters that have been reassigned will just appear to be missing.
  Edits your dataframe inplace. This is intentionally slower in
  the hopes of not crashing my ram by iterating over each line one by one
  """
  # get a list of our starting clusters
  _clusters = [int(x) for x in np.unique(new_predict)]
  # generate a list of new clusters
  _range = list(range(len(_clusters)))
  to_rename = {}
  # generate a dictionary with key value pairs of new : old clusters
  for num in _range:
    to_rename[_clusters[num]] = num + 1
  # go through the dataframe and edit our clusters
  for i in range(len(dataframe)):
    dataframe.at[i, 'cluster'] = to_rename[dataframe.at[i, 'cluster']]
```

```
def add_new_clst(df, dataframe, k, num_clust):
  """
  Pass in your new predictions and your dataframe
  and it will reassign the cluster you modeled to its new
  cluster assignments. Edits dataframe inplace.
  """
  # when generating new clusters after your first modeling
  # you'll need to assign them to new unique cluster numbers
  for i in df.index:
    # use the id's we assigned earlier to match up our clusters
    _id = df.at[i, 'id']
    index = dataframe.loc[dataframe['id'] == _id].index[0]
    # make sure we always get unique clusters, and that numbers being
    # reassigned to the same cluster always turn out the same
    n_cluster = (df.at[i, 'cluster'] + 2) * 10 + num_clust*2
    # reassign our clusters
    dataframe.at[index, 'cluster'] = n_cluster
```

```
def clusterizer(matrix, dataframe, calc_k, k=3, display_elbow=False):
  """
  Pass in your feature array/matrix and your dataframe
  If you would like to have k be calculated for you using the
  elbow method with a maximum number of 5 clusters, do so by
  setting calc_k = True, otherwise, k will be 3
  """
  if calc_k:
    if display_elbow:
      k = plt.elbow(matrix, dataframe, True)
```

```
      k = plt_elbow(matrix, dataframe, True)
    else:
      k = plt_elbow(matrix, dataframe, False)
  # making sure our dtypes always line up
  if type(matrix) != np.ndarray:
    matrix = matrix.toarray()
  # we dont want our clusters to be imbalanced, so we want them
  # to be roughly equal to the number of points/ number of clusters
  shape = matrix.shape[0]
  # start our k-means
  kmeans = faiss.Kmeans(d = matrix.shape[1], k = k, nredo = 100,
                        update_index = True, seed = 42, verbose=True,
                        max_points_per_centroid = math.ceil(shape/k),
                        niter = 20)
  # train our k means
  kmeans.train(matrix)
  # get our predictions
  predict = kmeans.index.search(matrix, 1)[1]
  # assign them inplace to our dataframe
  dataframe['cluster'] = predict
  return kmeans, predict, k
```

```
def get_slice(df, cluster):
  '''
  Pass in your dataframe and return just the values predicted to be in the
  largest cluster.
  '''
  return df.loc[df['cluster'] == cluster].reset_index(drop=True)
```

```
def showconfusionmatrix(y_t, y_hat_t, title):
    """
    Plots confusion matrix for provided y_train, y_hat_train
    OR y_test, y_hat_test using mlxtend
    """
    # assign parameters to dictionary so line wont exceed 78 char
    p = {conf_mat:confusion_matrix(y_t, y_hat_t), colorbar:True,
         show_absolute:True, show_normed:True, cmap:'jet'}
    # gather and plot confusion matrix
    fig, ax = plot_confusion_matrix(**p)
    # set title and axis names
    ax.set_title(f'{title} Model')
    ax.set_ylabel('Actual Data')
    ax.set_xlabel('Predicted Data')
    plt.show()
```

```
def printscores(y_test, y_hat_test, X_test, y_train,
                y_hat_train, X_train, model):
    """Uses sklearn.metrics to compute  testing and training scores
    Results include: Accuracy, Precision, Recall, F1, Residual Counts,
    Residuals, CV Accuracy With Standard Deviation & R^2"""
```

```
print('_____________________________________________________')
print('Training Accuracy Score: ' +
      str(round(accuracy_score(y_train, y_hat_train)* 100, 4)))
train_precision_scores = []
for item in precision_score(y_test, y_hat_test, average=None):
  train_precision_scores.append(round(item, 2))
print('Training Precision Scores: ' + str(train_precision_scores))
train_recall_scores = []
for item in recall_score(y_test, y_hat_test, average=None):
  train_recall_scores.append(round(item, 2))
print('Training Recall Scores: ' + str(train_recall_scores))
train_f1_scores = []
for item in f1_score(y_test, y_hat_test, average=None):
  train_f1_scores.append(round(item, 2))
print('Training F1 Scores: ' + str(train_f1_scores))
print('_____________________________________________________')
trainresiduals = np.abs(y_train - y_hat_train)
print('Training residual counts:')
print(str(pd.Series(trainresiduals).value_counts())[:-29])
print('_____________________________________________________')
print('Testing Accuracy Score: ' +
      str(round(accuracy_score(y_test, y_hat_test) * 100, 4)))
test_precision_scores = []
for item in precision_score(y_test, y_hat_test, average=None):
  test_precision_scores.append(round(item, 2))
print('Testing Precision Scores: ' + str(test_precision_scores))
test_recall_scores = []
for item in recall_score(y_test, y_hat_test, average=None):
  test_recall_scores.append(round(item, 2))
print('Testing Recall Scores: ' + str(test_recall_scores))
test_f1_scores = []
for item in f1_score(y_test, y_hat_test, average=None):
  test_f1_scores.append(round(item, 2))
print('Testing F1 Scores: ' + str(test_f1_scores))
print('_____________________________________________________')
testresiduals = np.abs(y_test - y_hat_test)
print('Testing residual counts:')
print(str(pd.Series(testresiduals).value_counts())[:-29])
print('_____________________________________________________')
r2_scoretst = r2_score(y_test, y_hat_test)
print(f'Testing R2 Score: {round(r2_scoretst, 4)}')
r2_scoretrn = r2_score(y_train, y_hat_train)
print(f'Training R2 Score: {round(r2_scoretrn, 4)}')
print('_____________________________________________________')
print('Cross validated model accuracy:')
scores = cross_val_score(model, X_test, y_test, cv=10)
mean = round(scores.mean(), 4)
std = round(scores.std())
print(f'Testing: {mean} with a standard deviation of {std}')
scores = cross_val_score(model, X_train, y_train, cv=10)
mean = round(scores.mean(), 4)
std = round(scores.std())
```

```
    std = round(scores.std())
    print(f'Training: {mean} with a standard deviation of {std}')
    print('_____________________________________________________')
```

```
def printreports(y_test, y_hat_test, y_train, y_hat_train):
    """
    Provided all input & predicted y values, prints classification
    report for training and testing data right next to each other"""
    print('                                                      ')
    print('               Testing Report:')
    report1 = classification_report(y_test, y_hat_test)
    print(report1)
    print('_____________________________________________________')
    print('               Training Report:')
    report2 = classification_report(y_train, y_hat_train)
    print(report2)
    print('_____________________________________________________')
```

Unsupervised Model Setup

First, we begin by opening up our data and doing some preprocessing, our data has already been cleaned and tokenized and lemmatized, so here we mostly focus on feature generation and dimensionality reduction

```
# open data, low_memory = False because we have some mixed dtype columns,
# they aren't one's we use anyways so not worried sbout it
df = pd.read_csv("Documents/flatiron/hospitality project/data/final_clean_6_word.csv",
                 low_memory=False)
```

```
# tokenize and preprocess our data, turning it into a matrix
matrix, df = vect_and_stck(df, True)
# taking an initial count to see how much we lose by the end
start = [len(df), datetime.now()]
```

First Unsupervised Model

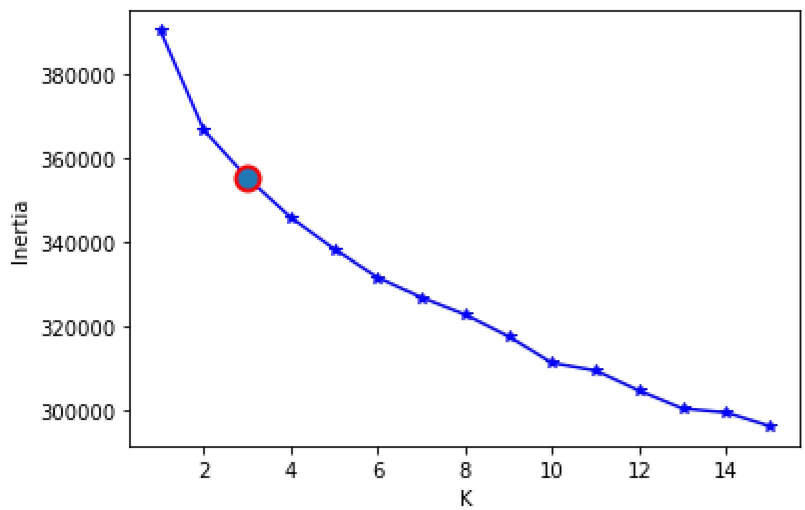
Model Evaluation

Looking at our initial model, our clusters aren't particularly well defined, and even with multiple attempts at what k should be, they're not showing any substantial improvement

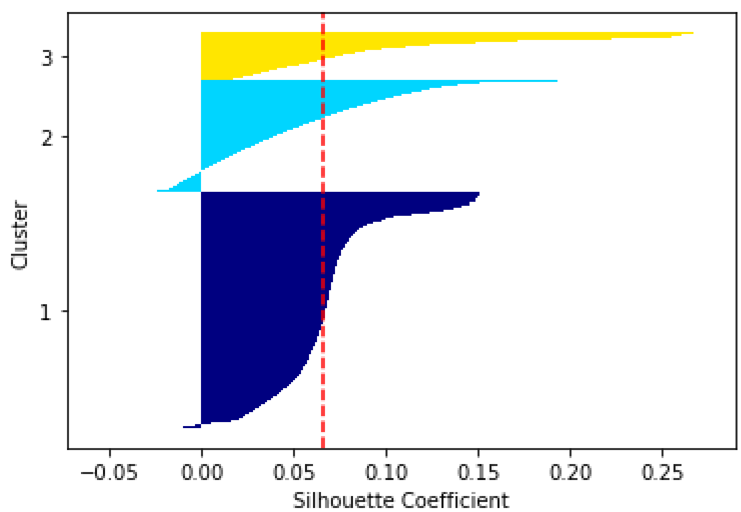
```
# starting a baseline model for our initial model
```

```
first_model = df.copy()
first_means, predict, k = clusterizer(matrix, first_model,
                                      True, display_elbow=True)
```

```
Elbow 3
```




```
# initial silhouette scores very low
quick_silhouette(first_model, matrix, predict)
```



Unsupervised Model Refinement

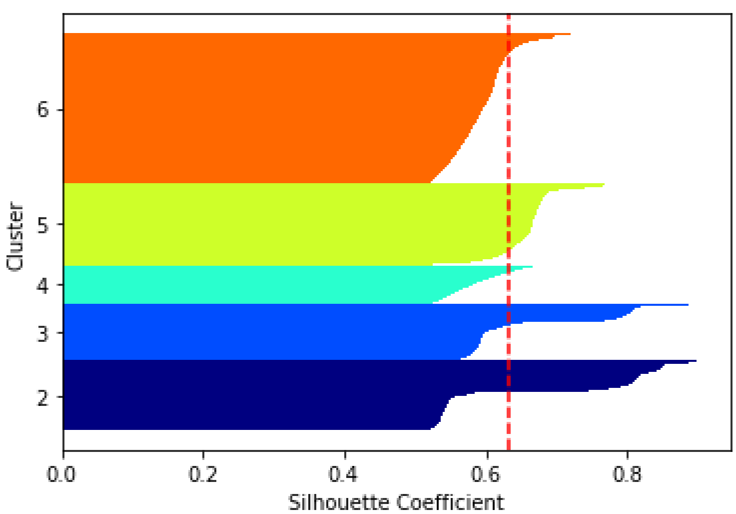
```
# start our model & make our initial predictions
second_means, predict, k = clusterizer(matrix, df, False)
```

```
num_clust = 2
cl = get_imb_clstr(df)
# if our largest cluster is too large, we break it down into more clusters
while len(get_slice(df, cl)) >= len(df)/num_clust-1 and num_clust <= 9:
    # dropping outliers and shuffling our data
```

```
# dropping outliers and shuffling our data
df = chunked_samples(df, matrix, predict, True)
df = df.sample(frac=1).reset_index(drop=True)
# processing our data without outliers
matrix, df = vect_and_stck(df, False)
# re-modeling our data with less outliers
bttr_mn, bttr_prd, k = clusterizer(matrix, df, False, 3, False)
# slicing our dataframe to get disproportionate cluster
cl = get_imb_clstr(df)
cut_matrix, cut_df = vect_and_stck(get_slice(df, cl), False)
# plugging our new k value into a new model optimized for our cluster
clst_mn, clst_prd, k = clusterizer(cut_matrix, cut_df, True, 3, False)
# updating our original datframe with our new clusters clusters
add_new_clst(cut_df, df, k, num_clust)
# updating our predictions
predict = df['cluster'].to_numpy()
# updating our largest cluster
cl = get_imb_clstr(df)
# keeping a count of clusters
num_clust += k-1
```

Model Evaluation

```
# grab our final clusters & plot our silhouettes
predict = df['cluster'].to_numpy()
reset_clusters(predict, df)
quick_silhouette(df, matrix, predict)
```




```
# calculating how long our modeling took & how much data was lost
end = [len(df), datetime.now()]
time = end[1]-start[1]
print(f' {round(end[0]/start[0], 4)}% Unclassifiable')
print(f' Model took {round(time.total_seconds()/60, 2)} min to complete')
```

```
0.5903% Unclassifiable
Model took 126.75 min to complete
```

Our model showed substantial improvement after iterating through & reclustering, losing minimal amounts of data. With our mean silhouette scores looking good and in a reasonable range, we move on to taking a look at visulizing our clusters and descerning the differences between them, which you can find in our notebook /wordclouds.ipynb

Bayesian Modeling

We chose bayesian modeling because it is said to work especially well for natural language processing, 2 of the methods available from Sklearn were not compatible with our data, but the other 2 are tested & refined below.

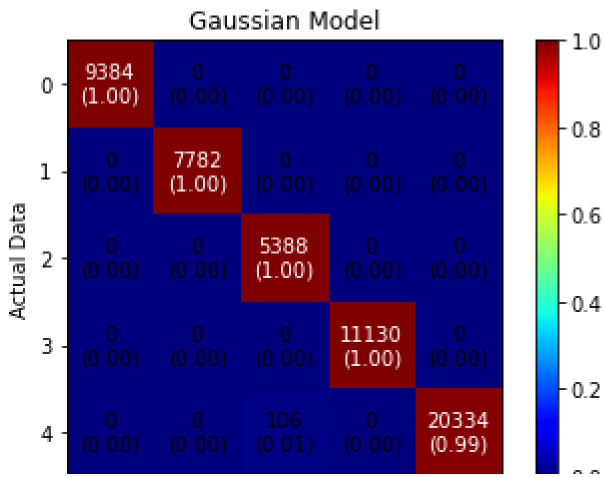
```
# since we have a bit of a class imbalance that we want to fix
# we're going to do a combination of upsampling and unsampling
# to even them out without overdoing it
smote = SMOTEENN()

X_train, X_test, y_train, y_test = train_test_split(matrix,
                                                    predict,
                                                    random_state=42)
# we're only applying this to our training data, so we don't leak
# anything over into our test that could cause overfitting
X_train, y_train = smote.fit_resample(X_train, y_train)
```

Testing models

There are our baseline models for making predictions off of our data

```
# the baseline model is actually incredibly close to our results
# while tuning it we discovered a few variables that did have considerable
# effect, so here is one to help showcase the difference
gau_model = GaussianNB(var_smoothing=1)
y_hat_test = gau_model.fit(X_train, y_train).predict(X_test)
y_hat_train = gau_model.predict(X_train)
```

```
showconfusionmatrix(y_test, y_hat_test, 'Gaussian')
# certainly too good to be true
```



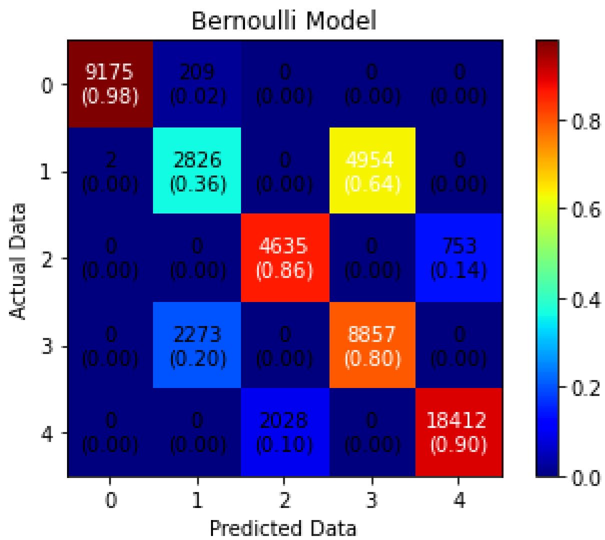

```
print('                Gaussian Model')
print('          __________________________')
printscores(y_test, y_hat_test, X_test, y_train, y_hat_train, X_train, gau_model)
```

```
                Gaussian Model
          __________________________
_____________________________________________________
Training Accuracy Score: 99.9006
Training Precision Scores: [1.0, 1.0, 0.98, 1.0, 1.0]
Training Recall Scores: [1.0, 1.0, 1.0, 1.0, 0.99]
Training F1 Scores: [1.0, 1.0, 0.99, 1.0, 1.0]
_____________________________________________________
Training residual counts:
0    30
_____________________________________________________
Testing Accuracy Score: 99.8042
Testing Precision Scores: [1.0, 1.0, 0.98, 1.0, 1.0]
Testing Recall Scores: [1.0, 1.0, 1.0, 1.0, 0.99]
Testing F1 Scores: [1.0, 1.0, 0.99, 1.0, 1.0]
_____________________________________________________
Testing residual counts:
0
_____________________________________________________
Testing R2 Score: 0.9966
Training R2 Score: 0.998
_____________________________________________________
Cross validated model accuracy:
Testing: 0.7858 with a standard deviation of 0.0
Training: 0.999 with a standard deviation of 0.0
_____________________________________________________
```

As you can see from our scores, this model is way too good to be true and is not producing a consistent output despite its high scores

```
bern_model = BernoulliNB()
y_hat_test = bern_model.fit(X_train, y_train).predict(X_test)
y_hat_train = bern_model.predict(X_train)
```

```
showconfusionmatrix(y_test, y_hat_test, 'Bernoulli')
```




```
print('                  Bernoulli Model')
print('           ______________________________')
printreports(y_test, y_hat_test, y_train, y_hat_train)
```

```
                 Bernoulli Model
           ______________________________

                Testing Report:
               precision    recall  f1-score   support

           1       1.00      0.98      0.99      9384
           2       0.53      0.36      0.43      7782
           3       0.70      0.86      0.77      5388
           4       0.64      0.80      0.71     11130
           5       0.96      0.90      0.93     20440

    accuracy                           0.81     54124
   macro avg       0.77      0.78      0.77     54124
weighted avg       0.81      0.81      0.81     54124

_____________________________________________________
                Training Report:
               precision    recall  f1-score   support

           1       1.00      0.98      0.99     61186
           2       0.62      0.37      0.46     61186
           3       0.91      0.87      0.89     60348
           4       0.56      0.80      0.66     61185
           5       0.88      0.91      0.89     58953

    accuracy                           0.79    302858
   macro avg       0.79      0.79      0.78    302858
weighted avg       0.79      0.79      0.78    302858

_____________________________________________________
```

```
print('                Bernoulli Model')
print('          __________________________')
printscores(y_test, y_hat_test, X_test, y_train, y_hat_train, X_train, bern_model)
```

```
                Bernoulli Model
          __________________________
_____________________________________________________
Training Accuracy Score: 78.5124
Training Precision Scores: [1.0, 0.53, 0.7, 0.64, 0.96]
Training Recall Scores: [0.98, 0.36, 0.86, 0.8, 0.9]
Training F1 Scores: [0.99, 0.43, 0.77, 0.71, 0.93]
_____________________________________________________
Training residual counts:
0    237781
2     6
_____________________________________________________
Testing Accuracy Score: 81.1193
Testing Precision Scores: [1.0, 0.53, 0.7, 0.64, 0.96]
Testing Recall Scores: [0.98, 0.36, 0.86, 0.8, 0.9]
Testing F1 Scores: [0.99, 0.43, 0.77, 0.71, 0.93]
_____________________________________________________
Testing residual counts:
0    43905
2
_____________________________________________________
Testing R2 Score: 0.6812
Training R2 Score: 0.5753
_____________________________________________________
Cross validated model accuracy:
Testing: 0.8185 with a standard deviation of 0.0
Training: 0.7848 with a standard deviation of 0.0
_____________________________________________________
```

This model is only slightly better than random chance overall, and in some clusters, it's even worse! Certainly not the route we want to be going.

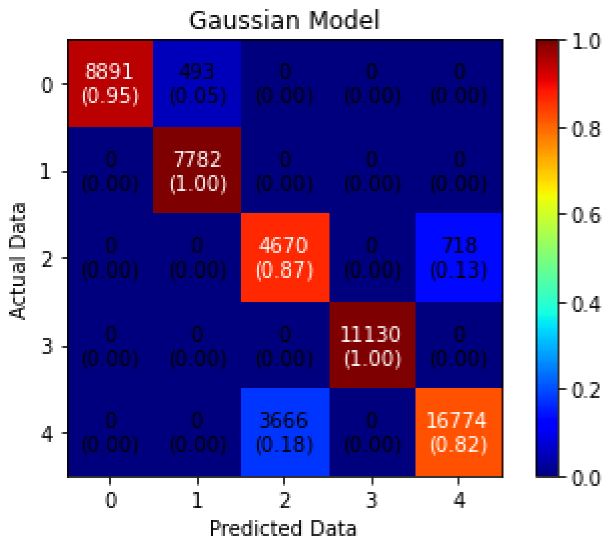
Final Model

Here, with a slight parameter tune, we see our final model come into realization with solid, consistent results.

```
final_model = GaussianNB(var_smoothing= 1e-200)
y_hat_test = final_model.fit(X_train, y_train).predict(X_test)
y_hat_train = final_model.predict(X_train)
```

Evaluation

```
showconfusionmatrix(y_test, y_hat_test, 'Gaussian')
```




```
print('                Final Gaussian Model')
print('              ___________________________')
printreports(y_test, y_hat_test, y_train, y_hat_train)
```

```
                Final Gaussian Model
              ___________________________

                  Testing Report:
                 precision    recall  f1-score   support

           1       1.00      0.95      0.97      9384
           2       0.94      1.00      0.97      7782
           3       0.56      0.87      0.68      5388
           4       1.00      1.00      1.00     11130
           5       0.96      0.82      0.88     20440

    accuracy                           0.91     54124
   macro avg       0.89      0.93      0.90     54124
weighted avg       0.93      0.91      0.92     54124

_____________________________________________________
                  Training Report:
                 precision    recall  f1-score   support

           1       1.00      0.95      0.97     61186
           2       0.95      1.00      0.98     61186
           3       0.84      0.86      0.85     60348
           4       1.00      1.00      1.00     61185
           5       0.86      0.83      0.84     58953

    accuracy                           0.93    302858
   macro avg       0.93      0.93      0.93    302858
weighted avg       0.93      0.93      0.93    302858

_____________________________________________________
```

```
print('                Final Gaussian Model')
```

```
print('          __________________________')
printscores(y_test, y_hat_test, X_test,
            y_train, y_hat_train, X_train, final_model)
```

```
                   Final Gaussian Model
          __________________________
__________________________________________________________
Training Accuracy Score: 93.0278
Training Precision Scores: [1.0, 0.94, 0.56, 1.0, 0.96]
Training Recall Scores: [0.95, 1.0, 0.87, 1.0, 0.82]
Training F1 Scores: [0.97, 0.97, 0.68, 1.0, 0.88]
__________________________________________________________
Training residual counts:
0    281742
2     1
__________________________________________________________
Testing Accuracy Score: 90.9892
Testing Precision Scores: [1.0, 0.94, 0.56, 1.0, 0.96]
Testing Recall Scores: [0.95, 1.0, 0.87, 1.0, 0.82]
Testing F1 Scores: [0.97, 0.97, 0.68, 1.0, 0.88]
__________________________________________________________
Testing residual counts:
0    49247
2
__________________________________________________________
Testing R2 Score: 0.8572
Training R2 Score: 0.875
__________________________________________________________
Cross validated model accuracy:
Testing: 0.9145 with a standard deviation of 0.0
Training: 0.93 with a standard deviation of 0.0
__________________________________________________________
```

In conclusion,

Even though we failed to find ***strong*** sentiment between our clusters, our model itself found clear enough differences between them to make meaningful and replicable predictions and the knowledge we came away from this with has given us great inspiration to make buisness reccomendations that will help your company move forward in the growing space hospitality industry.

Our reccomendations are:

1. Leverage use of the word 'Space' Space is the most commonly found word within the discourse, with that in mind, tagging along a positive sentiment will continue to build a positive arena around space and space travel & extend your reach to the largest audience.

2. Evoke emotion and drive the conversation. Customers think with their hearts, and by utilizing words like “love”, which are some of the most popular non-science words we found, you are likely to create a positive sentiment around your product. That positive emotional response adds tremendous value to your marketing capabilities.

3. Partner with the best Institutions like NASA and SpaceX not only have scientific headway, but they have built reputations. If you want to get people talking about you, working with buisnesses they already support can help extend your reach, as even with the variation between our clusters, the big names in them stay rather consistent.

   We look forward to working with you and appreciate your time. Thank you.