Reviews

1. **Problem Statement**

You are working in an e-commerce company, and your company has put forward a task to analyze the customer reviews for various products. You are supposed to create a report that classifies the products based on the customer reviews.

2. **Project Objective**

Find various trends and patterns in the reviews data, create useful insights that best describe the product quality.

Classify each review based on the sentiment associated with the same

3. **Data Description**

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm   # tqdm is for printing the status bar
from bs4 import BeautifulSoup

# library for splitting the dataset
from sklearn.model_selection import train_test_split

# libraries for featurization
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import Normalizer

# library for modeling
from sklearn.naive_bayes import MultinomialNB

#  library for hyperparameter tuning
from sklearn.model_selection import GridSearchCV

# evaluation of model
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

import re
```

In [2]:
```
data = pd.read_csv('Reviews (1) (1).csv')
data.head()
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

In [44]:
```
data.shape
```

Out[44]:
```
(525814, 10)
```

In [45]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 525814 entries, 0 to 568453
Data columns (total 10 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Id                      525814 non-null  int64
 1   ProductId               525814 non-null  object
 2   UserId                  525814 non-null  object
 3   ProfileName             525798 non-null  object
 4   HelpfulnessNumerator    525814 non-null  int64
 5   HelpfulnessDenominator  525814 non-null  int64
 6   Score                   525814 non-null  int64
 7   Time                    525814 non-null  int64
 8   Summary                 525789 non-null  object
 9   Text                    525814 non-null  object
dtypes: int64(5), object(5)
memory usage: 44.1+ MB
```

In [46]:
```
data.describe()
```

Out[46]:

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
count	525814.000000	525814.000000	525814.000000	525814.000000	5.258140e+05
mean	284599.060038	1.747293	2.209544	0.843981	1.295943e+09
std	163984.038077	7.575819	8.195329	0.362874	4.828129e+07
min	1.000000	0.000000	0.000000	0.000000	9.393408e+08
25%	142730.250000	0.000000	0.000000	1.000000	1.270598e+09
50%	284989.500000	0.000000	1.000000	1.000000	1.310861e+09
75%	426446.750000	2.000000	2.000000	1.000000	1.332634e+09
max	568454.000000	866.000000	878.000000	1.000000	1.351210e+09

Data Pre-processing Steps & Inspiration

In [3]:
```
data=data[data['Score']!=3]  #  we will not consider reviews with 'Score' 3, so we are droping all the rows with 'Score' feature equals 3
data.shape
```

Out[3]:
```
(525814, 10)
```

In [4]:
```
def partition(x):  # given x it returns 1 if x>3 else returns 0
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be negative(0) and greater the 3 to be positive(1)
actual_score = data['Score']
positive_negative = actual_score.map(partition)
data['Score'] = positive_negative
data.head()
```

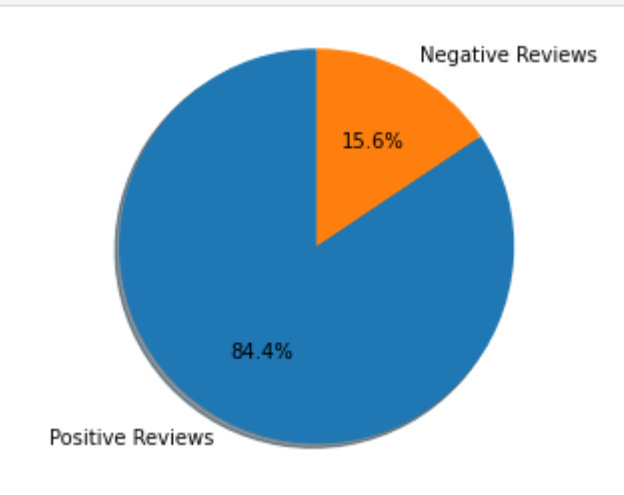
Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	0	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	1	1350777600	Great taffy	Great taffy at a great price. There was a wid...

In [5]:
```
data['Score'].value_counts()
```

Out[5]:
```
1    443777
0     82037
Name: Score, dtype: int64
```

In [6]:
```
labels = 'Positive Reviews', 'Negative Reviews'
sizes = [443777, 82037]
fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



In [7]:
```
sorted_data = data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')

# Droping Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[7]:
```
(364173, 10)
```

In [8]:
```
def decontracted(phrase):  # this function expands english language contraction such as (that's) to ('that is')
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase


# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [9]:
```
## Preprocessing 'Text' column
# The Below code removes url's , html tags, words with numbers, special character, stopwords and decontracts words in the Text for each review

preprocessed_reviews = []
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████████████████████████████████████████████| 364173/364173 [02:30<00:00, 2417.41it/s]
```

In [10]:
```
# Preprocessing Summary column
# The Below code removes url's , html tags, words with numbers, special character, stopwords and decontracts words in the Summary for each rev

preprocessed_Summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].values):
    sentance = re.sub(r"http\S+", "", str(sentance))
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_Summary.append(sentance.strip())
```

```
100%|██████████████████████████████████████████████████████████████████████| 364173/364173 [01:31<00:00, 3988.02it/s]
```

In [11]:
```
# let's replace the 'Summary' and 'Text' column with the preprocessed data.
final["Summary"] = preprocessed_Summary
final['Text'] = preprocessed_reviews
final.drop(['Id', 'ProductId', 'UserId', 'ProfileName'], axis = 1, inplace=True)  # not considering these columns for classification.
```

In [12]:
```
final.reset_index(inplace=True)
```

In [13]:
```
final.drop(['index'], axis=1, inplace=True)
final.head()
```

Out[13]:

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	0	0	1	939340800	every book educational	witty little book makes son laugh loud recite ...
1	1	1	1	1194739200	love book miss hard cover version	grew reading sendak books watching really rosi...
2	1	1	1	1191456000	chicken soup rice months	fun way children learn months year learn poems...
3	1	1	1	1076025600	good swingy rhythm reading aloud	great little book read aloud nice rhythm well ...
4	3	4	1	1018396800	great way learn months	book poetry months year goes month cute little...

In [14]:
```
y = final['Score'].values
X = final.drop(['Score'], axis=1)

# splitting the data and class labels in to train set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

In [15]:
```
print('train data shape', X_train.shape)
print('train data labels shape ', y_train.shape)
print('test data shape', X_test.shape)
print('test data labels shape', y_test.shape)
```

```
train data shape (243995, 5)
train data labels shape  (243995,)
test data shape (120178, 5)
test data labels shape (120178,)
```

In [21]:
```
### Text feature
# calling the CountVectorizer class with three parameters
vectorizer = CountVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['Text'].values) # fitting the model with train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_text = vectorizer.transform(X_train['Text'].values)
X_test_text = vectorizer.transform(X_test['Text'].values)

# getting the features
features_text = vectorizer.get_feature_names() # we will be using this afterwords

print("After vectorizations using BOW")
print(X_train_text.shape, y_train.shape)
print(X_test_text.shape, y_test.shape)
```

```
After vectorizations using BOW
(243995, 5000) (243995,)
(120178, 5000) (120178,)
```

In [22]:
```
################# Summary feature ##################
# calling the CountVectorizer class with three parameters
vectorizer = CountVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['Summary'].values) # fitting the model with train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_summary = vectorizer.transform(X_train['Summary'].values)
X_test_summary = vectorizer.transform(X_test['Summary'].values)

# getting the features of bow vectorization
features_summary = vectorizer.get_feature_names() # we will be using this afterwords

print("After vectorizations using BOW")
print(X_train_summary.shape, y_train.shape)
print(X_test_summary.shape, y_test.shape)
```

```
After vectorizations using BOW
(243995, 5000) (243995,)
(120178, 5000) (120178,)
```

In [ ]:
```
X_train_price_norm = normalizer.fit_transform(X_train['price'].values.reshape(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))
```

In [29]:
```
# "HelpfulnessNumerator" feature
normalizer = Normalizer() # normalizing numerical features such that all numerical features are in same range.

normalizer.fit_transform(X_train['HelpfulnessNumerator'].values.reshape(-1,1))

X_train_help_num_norm = normalizer.transform(X_train['HelpfulnessNumerator'].values.reshape(-1,1))
X_test_help_num_norm = normalizer.transform(X_test['HelpfulnessNumerator'].values.reshape(-1,1))


print("After normalization of price feature")
print(X_train_help_num_norm.shape, y_train.shape)
print(X_test_help_num_norm.shape, y_test.shape)
print("="*100)
```

```
After normalization of price feature
(243995, 1) (243995,)
(120178, 1) (120178,)
====================================================================================================
```

In [30]:
```
# "HelpfulnessDenominator" feature
normalizer = Normalizer()

normalizer.fit(X_train['HelpfulnessDenominator'].values.reshape(-1,1))

X_train_help_den_norm = normalizer.transform(X_train['HelpfulnessDenominator'].values.reshape(-1,1))
X_test_help_den_norm = normalizer.transform(X_test['HelpfulnessDenominator'].values.reshape(-1,1))


print("After normalization of price feature")
print(X_train_help_den_norm.shape, y_train.shape)
print(X_test_help_den_norm.shape, y_test.shape)
print("="*100)
```

```
After normalization of price feature
(243995, 1) (243995,)
(120178, 1) (120178,)
====================================================================================================
```

In [31]:
```
# "Time" feature
normalizer = Normalizer()

normalizer.fit(X_train['Time'].values.reshape(-1,1))

X_train_time_norm = normalizer.transform(X_train['Time'].values.reshape(-1,1))
X_test_time_norm = normalizer.transform(X_test['Time'].values.reshape(-1,1))


print("After normalization of price feature")
print(X_train_time_norm.shape, y_train.shape)
print(X_test_time_norm.shape, y_test.shape)
print("="*100)
```

```
After normalization of price feature
(243995, 1) (243995,)
(120178, 1) (120178,)
====================================================================================================
```

In [32]:
```
from scipy.sparse import hstack

X_tr = hstack((X_train_text, X_train_summary, X_train_help_num_norm,  # train data after BOW representation for 'Text' and 'Summary' feature.
                  X_train_help_den_norm, X_train_time_norm)).tocsr()

X_te = hstack((X_test_text, X_test_summary, X_test_help_num_norm,  # test data after BOW representation for 'Text' and 'Summary' feature.
                  X_test_help_den_norm, X_test_time_norm)).tocsr()


print("Final Data matrix with BOW representation for essay")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix with BOW representation for essay
(243995, 10003) (243995,)
(120178, 10003) (120178,)
====================================================================================================
```

Choosing the Algorithm for the project

- I have choosen Naive Bayes's classifier because it handles large data very well.
- It doesn't require as much training data.
- It handles both continuous and discrete data.
- It is highly scalable with the number of predictors and data points. It is fast and can be used to make real-time predictions.

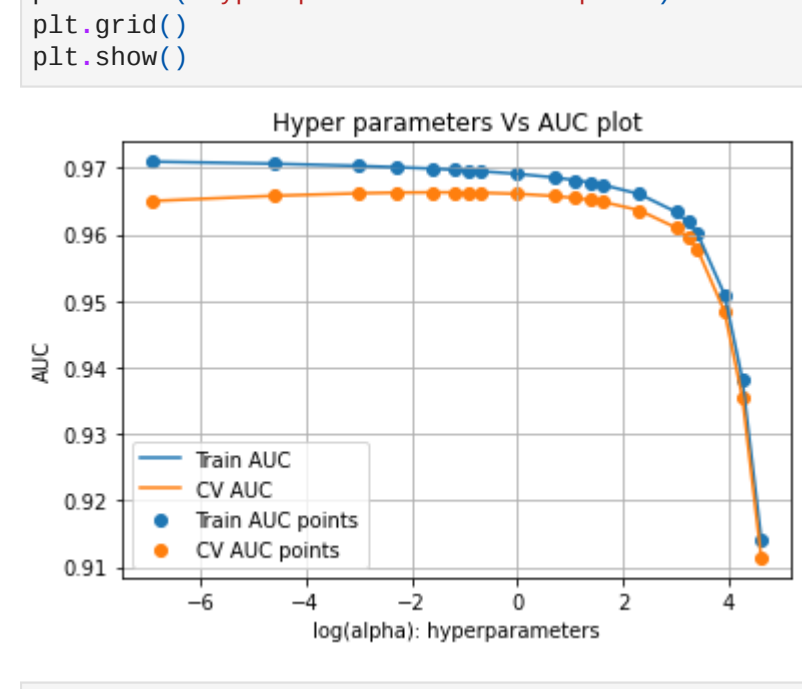
In [33]:
```
from sklearn.model_selection import GridSearchCV

NB_classifier = MultinomialNB(class_prior=[0.5, 0.5])

parameters = {'alpha': [0.001, 0.05, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 3, 4, 5, 10, 20, 25, 30, 50, 70, 100]}  # various values of alhap's
clf = GridSearchCV(NB_classifier, parameters, cv=5, scoring='roc_auc', return_train_score=True)  # gridsearchCV with 5 fold cross validation .
clf.fit(X_tr, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])
```

Assumptions

- From above figure(misclassification error vs optimal alpha) It is showing that classification error for each value of alpha, when alpha is increaseing the error is also increasing.
- As I tested our model on unseen data(test data) the accuracy is 86% when alpha = 9.0
- In confusion matrix, It is clear that out of 30k unseen data-points classifier predict 12853 +ve and 12871 -ve class label but in real 15049 were +ve and 14951 were -ve.
- In a nutshell we can say the generalization error is low means t

Model Evaluation & Techniques

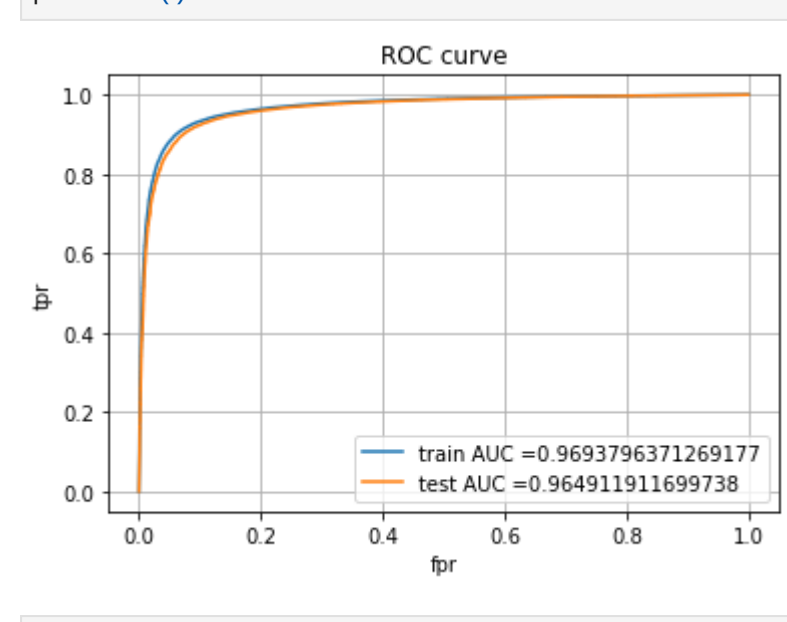
In [34]:
```
cv_auc = results['mean_test_score']      # mean test scores for every 'alpha'
train_auc = results['mean_train_score']  # mean train scores for every 'alpha

alpha =  list(results['param_alpha'])
alpha=np.log(alpha)   # taking log of alphas so to make the plot more readable

plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')

plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("log(alpha): hyperparameters")
plt.ylabel("AUC")
plt.title("Hyper parameters Vs AUC plot")
plt.grid()
plt.show()
```



In [35]:
```
clf.best_estimator_ # using this estimator lets predict the labels of test dataset
```

Out[35]:
```
MultinomialNB(alpha=0.2, class_prior=[0.5, 0.5])
```

In [36]:
```
NBclassifier = MultinomialNB(alpha=0.2, class_prior=[0.5, 0.5], fit_prior=True)
NBclassifier.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = NBclassifier.predict_proba(X_tr)[:,1]     # predicted probabilities of train datapoints belonging to positive class
y_test_pred = NBclassifier.predict_proba(X_te)[:,1]      # predicted probabilities of test datapoints belonging to positive class

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)   # fpr and tpr for train data
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)       # fpr and tpr for test data

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curve")
plt.grid()
plt.show()
```



In [37]:
```
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

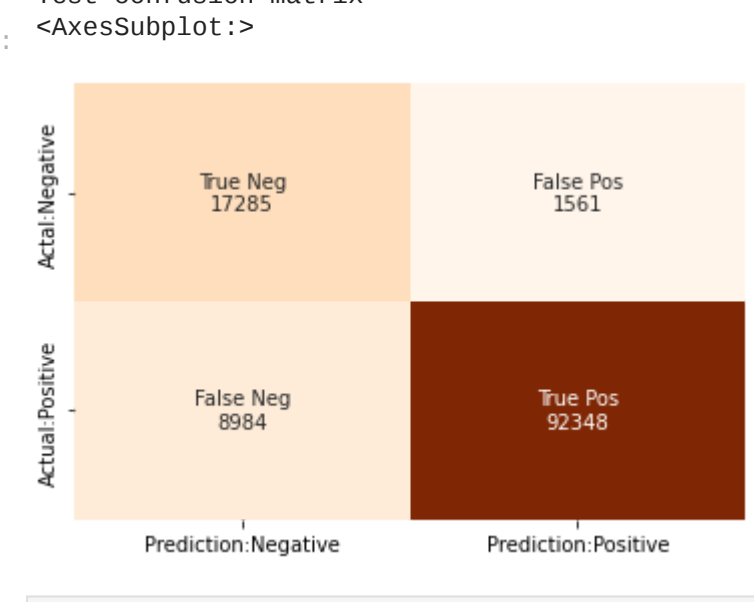
In [38]:
```
from sklearn.metrics import confusion_matrix

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)  # getting the best threshold for separating the positive classes form negat

test_confusion_matrix = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))  # calculates the confusion matrix
```

```
the maximum value of tpr*(1-fpr) 0.8472203643930851 for threshold 0.53
```

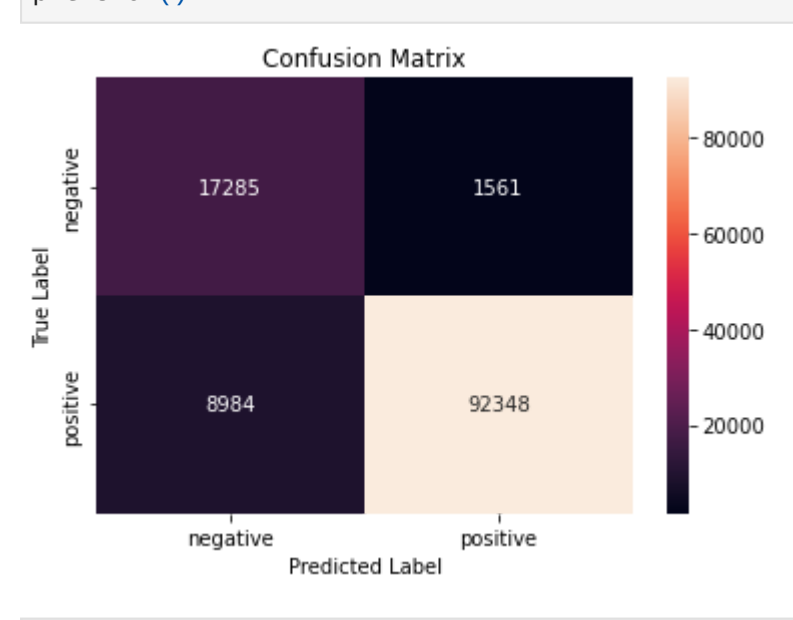
In [39]:
```
group_names = ['True Neg','False Pos','False Neg','True Pos']

group_counts = ['{0:0.0f}'.format(value) for value in
                test_confusion_matrix.flatten()]

labels = [f'{v1}\n{v2}' for v1, v2 in
          zip(group_names,group_counts)]
labels = np.asarray(labels).reshape(2,2)

print("Test confusion matrix")
sns.heatmap(test_confusion_matrix, annot=labels, fmt='', cmap='Oranges',cbar=False, xticklabels=['Prediction:Negative', 'Prediction:Positive']
```

```
Test confusion matrix
```

Out[39]:
```
<AxesSubplot:>
```



In [40]:
```
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(test_confusion_matrix, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



In [41]:
```
list_of_features = features_text + features_summary + ['HelpfulnessNumerator', 'HelpfulnessDenominator', 'Time']
```

In [42]:
```
# Top 20 features form negative class
features = np.argsort(NBclassifier.feature_log_prob_[0]) # sorting the features log probability for negtive class
                                                          #  form low probability to high probability and getting its indice

features = features[::-1]  # reversing it form high probability to low probability indice

for i in features[:20]:  # printing top 20 features from negative calss
    print(list_of_features[i])
```

```
not
Time
HelpfulnessDenominator
HelpfulnessNumerator
like
product
would
taste
one
good
no
flavor
coffee
not
get
even
food
tea
buy
amazon
```

In [43]:
```
# Top 20 features form negative class
features = np.argsort(NBclassifier.feature_log_prob_[1]) # sorting the features log probability for negtive class
                                                          #  form low probability to high probability and getting its indice

features = features[::-1]  # reversing it form high probability to low probability indice

for i in features[:20]:  # printing top 20 features from negative calss
    print(list_of_features[i])
```

```
Time
not
HelpfulnessDenominator
HelpfulnessNumerator
like
good
great
one
taste
product
love
flavor
tea
coffee
would
get
no
food
really
use
```

Inferences & Future Scope

- Naive bayes are good at text classification task like spam filtering, sentimental analysis, RS etc.
- As we know when a model performs good on training data but poor performence on unseen data(test data)i.e. its dependent on training data only, tends to overfits and when a model perform poor performence on training data and good performence on test data i.e. it fails to learn relationship in training data tends to underfit. We need to balance between both i.e. reduce training error and balance err
- Another concept bias vs variance is also related with underfitting and overfitting. when a model has high bias and low variance tend to underfitting and its reverse- high variance and low bias called overfitting and we balanced using cross-validataion. As it is shown in below table where both models have low trainig error and test error.
- Overall, both of the models are performing well on unseen data.
- As we are not applying naive bayes on word2vec representation because it sometimes gives -ve value(i.e. if two word have 0 cosine similarity the word is completly orthogonal i.e. they are not related with each other. and 1 represents perfect relationship between word vector. whereas -ve similarity means they are perfect opposite relationship between word) and we know naive bayes assume that presence of a particular feature in a class is unrelated to presence of any other feature, which is most unlikely in real word. Although, it works well.
- And from point # 5, features are dependent or there are relationship between features. So applying naive bayes on dependent feature does not make any sense.