4_DonorsChoose_NB

March 15, 2020

1 Assignment 6: Apply NB

```
<li><strong>Apply Multinomial NB on these feature sets</strong>
    <ul>
        <li><font color='red'>Set 1</font>: categorical, numerical features + preprocessed_eass
        <li><font color='red'>Set 2</font>: categorical, numerical features + preprocessed_eass
    </ul>
</li>
<li><strong>The hyper paramter tuning(find best alpha:smoothing parameter)</strong>
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>find the best hyper paramter using k-fold cross validation(use GridsearchCV or Randomsearch
<li></li>
    </ul>
</li>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='https://i.imgur.com/hUv6aEy.jpg' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='https://i.imgur.com/wMQDTFe.jpg' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='https://i.imgur.com/IdN5Ctv.png' width=300px></li>
    </ul>
</li>
<li>
```

fine the top 20 features from either from feature Set 1 or feature Set 2 using absolute values of `feature_log_prob_` parameter of `MultinomialNB` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print their corresponding feature names

```
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='http://i.imgur.com/YVpIGGE.jpg' width=400px>
</li>
```

2. Naive Bayes

1.1 1.1 Loading Data

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

        import sqlite3
        import pandas as pd
        import numpy as np
        import nltk
        import string
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.feature_extraction.text import TfidfTransformer
        from sklearn.feature_extraction.text import TfidfVectorizer

        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.metrics import confusion_matrix
        from sklearn import metrics
        from sklearn.metrics import roc_curve, auc
        from nltk.stem.porter import PorterStemmer

        import re
        # Tutorial about Python regular expressions: https://pymotw.com/2/re/
        import string
        from nltk.corpus import stopwords
        from nltk.stem import PorterStemmer
        from nltk.stem.wordnet import WordNetLemmatizer

        from gensim.models import Word2Vec
        from gensim.models import KeyedVectors
        import pickle

        from tqdm import tqdm
        import os

        import plotly.offline as offline
        import plotly.graph_objs as go
        offline.init_notebook_mode()
        from collections import Counter
```

```
In [2]: from google.colab import drive
        drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-

Enter your authorization code:
ůůůůůůůůůů
```

```
Mounted at /content/drive
```

```
In [0]: import pandas
        #data = pandas.read_csv(r'C:\Users\ASUS\Downloads\Applied AI\Assignments - Applied AI\
        project_data = pd.read_csv('/content/drive/My Drive/Assignments_DonorsChoose_2018/trair
        resource_data = pd.read_csv('/content/drive/My Drive/Assignments_DonorsChoose_2018/reso
```

```
In [23]: # how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
         cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.co


         #sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/408
         project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
         project_data.drop('project_submitted_datetime', axis=1, inplace=True)
         project_data.sort_values(by=['Date'], inplace=True)


         # how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
         project_data = project_data[cols]


         project_data.head(2)
```

```
Out[23]:        Unnamed: 0  ... project_is_approved
         55660        8393  ...                   1
         76127       37728  ...                   1

         [2 rows x 17 columns]
```

```
In [24]: print("Number of data points in train data", resource_data.shape)
         print(resource_data.columns.values)
         resource_data.head(2)
```

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

```
Out[24]:         id                                       description  quantity   price
         0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1  149.00
         1  p069063        Bouncy Bands for Desks (Blue support pipes)         3   14.95
```

1.2 1.2 preprocessing of `project_subject_categories`

```
In [0]: catogories = list(project_data['project_subject_categories'].values)
        # remove special characters from list of strings python: https://stackoverflow.com/a/4

        # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
        # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-str
        # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyt
        cat_list = []
        for i in catogories:
            temp = ""
            # consider we have text like this "Math & Science, Warmth, Care & Hunger"
            for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmt
                if 'The' in j.split(): # this will split each of the catogory based on space "
                    j=j.replace('The','') # if we have the words "The" we are going to replace
                j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"
                temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing sp
                temp = temp.replace('&','_') # we are replacing the & value into
            cat_list.append(temp.strip())

        project_data['clean_categories'] = cat_list
        project_data.drop(['project_subject_categories'], axis=1, inplace=True)

        from collections import Counter
        my_counter = Counter()
        for word in project_data['clean_categories'].values:
            my_counter.update(word.split())

        cat_dict = dict(my_counter)
        sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

1.3 1.3 preprocessing of `project_subject_subcategories`

```
In [0]: sub_catogories = list(project_data['project_subject_subcategories'].values)
        # remove special characters from list of strings python: https://stackoverflow.com/a/4

        # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
        # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-str
        # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyt

        sub_cat_list = []
        for i in sub_catogories:
            temp = ""
            # consider we have text like this "Math & Science, Warmth, Care & Hunger"
            for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmt
                if 'The' in j.split(): # this will split each of the catogory based on space "
                    j=j.replace('The','') # if we have the words "The" we are going to replace
                j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"
                temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing sp
                temp = temp.replace('&','_')
            sub_cat_list.append(temp.strip())

        project_data['clean_subcategories'] = sub_cat_list
        project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
        # count of all the words in corpus python: https://stackoverflow.com/a/22898595/408403
        my_counter = Counter()
        for word in project_data['clean_subcategories'].values:
            my_counter.update(word.split())

        sub_cat_dict = dict(my_counter)
        sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

1.4 Preprocessing of project_grade_category

```
In [0]: project_grade = list(project_data['project_grade_category'].values)
        # remove special characters from list of strings python: https://stackoverflow.com/a/4

        # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
        # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-str
        # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyt

        grade_cat_list = []
        for i in project_grade:
            # consider we have text like this:
            for j in i.split(' '): #    # split by spae
                j=j.replace('Grades','')# clean grades from the row
            grade_cat_list.append(j.strip())

        project_data['grade_cat_list'] = grade_cat_list
        project_data.drop(['project_grade_category'], axis=1, inplace=True)

        my_counter = Counter()
        for word in project_data['grade_cat_list'].values:
            my_counter.update(word.split())

        sub_cat_dict = dict(my_counter)
        sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

1.5 Join train & Resource dataset

```
In [28]: price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset
         project_data = pd.merge(project_data, price_data, on='id', how='left')
         project_data.head(2)
```

```
Out[28]:    Unnamed: 0       id  ...   price quantity
         0        8393  p205479  ...  725.05        4
         1       37728  p043609  ...  213.03        8

         [2 rows x 19 columns]
```

1.6 Train Test split

```
In [42]: y = project_data['project_is_approved'].values
         X = project_data.drop(['project_is_approved'], axis=1)
         X.head(1)
```

```
Out[42]:    Unnamed: 0  ...                                          essay
         0        8393  ...  I have been fortunate enough to use the Fairy ...

         [1 rows x 19 columns]
```

```
In [0]: # train test split
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
```

1.7 1.3 Text preprocessing

```
In [0]: # merge two column text dataframe:
        project_data["essay"] = project_data["project_essay_1"].map(str) +\
                                project_data["project_essay_2"].map(str) + \
                                project_data["project_essay_3"].map(str) + \
                                project_data["project_essay_4"].map(str)
```

```
In [32]: project_data.head(2)
```

```
Out[32]:    Unnamed: 0  ...                                          essay
         0        8393  ...  I have been fortunate enough to use the Fairy ...
         1       37728  ...  Imagine being 8-9 years old. You're in your th...

         [2 rows x 20 columns]
```

```
In [33]: # printing some random reviews
         print(project_data['essay'].values[0])
         print("="*50)
         print(project_data['essay'].values[150])
         print("="*50)
         print(project_data['essay'].values[1000])
         print("="*50)
         print(project_data['essay'].values[20000])
         print("="*50)
         print(project_data['essay'].values[99999])
         print("="*50)
```

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as well as the STE
==================================================
I teach high school English to students with learning and behavioral disabilities. My students
==================================================
\"Life moves pretty fast. If you don't stop and look around once in awhile, you could miss it.\
==================================================
\"A person's a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the
==================================================
My classroom consists of twenty-two amazing sixth graders from different cultures and backgrou
==================================================
```

```
In [0]: # https://stackoverflow.com/a/47091490/4084039
        import re

        def decontracted(phrase):
            # 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
```

```
In [35]: sent = decontracted(project_data['essay'].values[20000])
         print(sent)
         print("="*50)
```

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with th
==================================================
```

```
In [36]: # \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-
         sent = sent.replace('\\r', ' ')
         sent = sent.replace('\\"', ' ')
         sent = sent.replace('\\n', ' ')
         print(sent)
```

```
 A person is a person, no matter how small.  (Dr.Seuss) I teach the smallest students with the
```

```
In [37]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
         print(sent)
```

```
 A person is a person no matter how small Dr Seuss I teach the smallest students with the bigge
```

```
In [0]: # https://gist.github.com/sebleier/554280
        # we are removing the words from the stop words list: 'no', 'nor', 'not'
        stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're
                    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
                    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', '
                    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
                    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
                    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
                    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
                    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any
                    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
                    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
                    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", '
                    'won', "won't", 'wouldn', "wouldn't"]
```

```
In [40]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_essays = []
         # tqdm is for printing the status bar
         for sentance in tqdm(project_data['essay'].values):
             sent = decontracted(sentance)
             sent = sent.replace('\\r', ' ')
             sent = sent.replace('\\"', ' ')
             sent = sent.replace('\\n', ' ')
             sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
             # https://gist.github.com/sebleier/554280
             sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
             preprocessed_essays.append(sent.lower().strip())
```

```
100%|| 109248/109248 [01:01<00:00, 1763.12it/s]
```

```
In [44]: # Combining all the above stundents
         from tqdm import tqdm
         train_preprocessed_essays = []
         # tqdm is for printing the status bar
         for sentance in tqdm(X_train['essay'].values):
             sent = decontracted(sentance)
             sent = sent.replace('\\r', ' ')
             sent = sent.replace('\\"', ' ')
             sent = sent.replace('\\n', ' ')
             sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
             # https://gist.github.com/sebleier/554280
             sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
             train_preprocessed_essays.append(sent.lower().strip())
```

```
100%|| 87398/87398 [00:49<00:00, 1778.79it/s]
```

```
In [45]: # Combining all the above stundents
         from tqdm import tqdm
         test_preprocessed_essays = []
         # tqdm is for printing the status bar
         for sentance in tqdm(X_test['essay'].values):
             sent = decontracted(sentance)
             sent = sent.replace('\\r', ' ')
             sent = sent.replace('\\"', ' ')
             sent = sent.replace('\\n', ' ')
             sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
             # https://gist.github.com/sebleier/554280
             sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
             test_preprocessed_essays.append(sent.lower().strip())
```

```
100%|| 21850/21850 [00:12<00:00, 1776.15it/s]
```

```
In [46]: # after preprocesing
         test_preprocessed_essays[20000]
```

```
Out[46]: 'students world potential hands journey begins opening pages book walk door every day
```

1.7.1 1.5.1 Vectorizing Categorical data

```
In [47]: # we use count vectorizer to convert the values into one
         from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False
         categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
         #print(vectorizer.get_feature_names())
         print("Shape of matrix after one hot encodig ",categories_one_hot.shape)

         cat_features = vectorizer.get_feature_names()
         print(cat_features)
         print(len(cat_features))
```

```
Shape of matrix after one hot encodig  (109248, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'I
9
```

```
In [48]: # we use count vectorizer to convert the values into one
         vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fa
         sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories']
         #print(vectorizer.get_feature_names())
         print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
         sub_cat_features=vectorizer.get_feature_names()
         print(sub_cat_features)
         print(len(sub_cat_features))
```

```
Shape of matrix after one hot encodig  (109248, 30)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
30
```

```
In [0]: # count of all the words in corpus python: https://stackoverflow.com/a/22898595/408403
        my_counter = Counter()
        for word in project_data['school_state'].values:
            my_counter.update(word.split())

        sub_cat_dict = dict(my_counter)
        sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

```
In [50]: # we use count vectorizer to convert the values into one
         from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fa
         categories_one_hot = vectorizer.fit_transform(project_data['school_state'].values)
         #print(vectorizer.get_feature_names())
         print("Shape of matrix after one hot encodig ",categories_one_hot.shape)

         state_features=vectorizer.get_feature_names()
         print(state_features)
         print(len(state_features))
```

```
Shape of matrix after one hot encodig  (109248, 51)
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME', 'HI', 'DC', 'NM', 'KS
51
```

```
In [0]: # count of all the words in corpus python: https://stackoverflow.com/a/22898595/408403
        project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(" ")
        my_counter = Counter()
        for word in project_data['teacher_prefix'].values.astype('str'):  #https://stackoverfl
            my_counter.update(word.split())

        sub_fix_dict = dict(my_counter)
        sorted_fix_dict = dict(sorted(sub_fix_dict.items(), key=lambda kv: kv[1]))
```

```
In [76]: # we use count vectorizer to convert the values into one
         from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer(vocabulary=list(sorted_fix_dict.keys()), lowercase=False
         prefix_one_hot = vectorizer.fit_transform(project_data['teacher_prefix'].values.astype
         #print(vectorizer.get_feature_names())
         print("Shape of matrix after one hot encodig ",prefix_one_hot.shape)
         prefix_features=vectorizer.get_feature_names()
         print(prefix_features)
         print(len(prefix_features))
```

```
Shape of matrix after one hot encodig  (109248, 5)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
5
```

```
In [0]: # count of all the words in corpus python: https://stackoverflow.com/a/22898595/408403
        my_counter = Counter()
        for word in project_data['grade_cat_list'].values:
            my_counter.update(word.split())

        sub_cat_dict = dict(my_counter)
        sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

```
In [67]: # we use count vectorizer to convert the values into one
         from sklearn.feature_extraction.text import CountVectorizer
         vectorizer1 = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=F
         grade_one_hot = vectorizer1.fit_transform(project_data['grade_cat_list'].values)
         print(vectorizer1.get_feature_names())
         print("Shape of matrix after one hot encodig ",grade_one_hot.shape)
```

```
['9-12', '6-8', '3-5', 'PreK-2']
Shape of matrix after one hot encodig  (109248, 4)
```

1.7.2 1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

```
In [55]: # We are considering only the words which appeared in at least 10 documents(rows or p
         vectorizer = CountVectorizer(min_df=10)
         text_bow = vectorizer.fit_transform(preprocessed_essays)
         print("Shape of matrix after one hot encodig ",text_bow.shape)

         essays_features=vectorizer.get_feature_names()
         #print(essays_features)
         print(len(essays_features))
```

```
Shape of matrix after one hot encodig  (109248, 16512)
16512
```

1.5.2.2 TFIDF vectorizer

```
In [99]: from sklearn.feature_extraction.text import TfidfVectorizer
         vectorizer3 = TfidfVectorizer(min_df=10)
         text_tfidf = vectorizer3.fit_transform(preprocessed_essays)
         print("Shape of matrix after one hot encodig ",text_tfidf.shape)

         essays_features_tf=vectorizer3.get_feature_names()
         #print(essays_features)
         print(len(essays_features_tf))
```

```
Shape of matrix after one hot encodig  (109248, 16512)
16512
```

1.7.3 1.5.3 Vectorizing Numerical features

```
In [0]: # check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
        # standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.p
        from sklearn.preprocessing import Normalizer

        # price_standardized = standardScalar.fit(project_data['price'].values)
        # this will rise the error
        # ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ...
        # Reshape your data either using array.reshape(-1, 1)

        price_scalar = Normalizer()
        price_scalar.fit(project_data['price'].values.reshape(-1,1)) # finding the mean and st


        # Now standardize the data with above maen and variance.
        price_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1
        tr_price_standardized = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
        te_price_standardized = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
```

```
In [0]: # check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
        # standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.p
        from sklearn.preprocessing import Normalizer

        # price_standardized = standardScalar.fit(project_data['price'].values)
        # this will rise the error
        # ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ...
        # Reshape your data either using array.reshape(-1, 1)

        quantity_scalar = Normalizer()
        quantity_scalar.fit(project_data['quantity'].values.reshape(-1,1)) # finding the mean 

        # Now standardize the data with above maen and variance.
        quantity_standardized = quantity_scalar.transform(project_data['quantity'].values.resha
        tr_quantity_standardized = quantity_scalar.transform(X_train['quantity'].values.reshape
        te_quantity_standardized = quantity_scalar.transform(X_test['quantity'].values.reshape
```

1.3 Make Data Model Ready: encoding eassay, and project_title

2 Encoding - Categorical

In [59]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train

# we use the fitted CountVectorizer to convert the text to vector
X_train_cl_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_test_cl_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cl_categories_ohe.shape, y_train.shape)
print(X_test_cl_categories_ohe.shape, y_test.shape)
print("="*100)


# print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
# vectorizer = CountVectorizer()
# x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
# x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
# x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

# print(x_train_bow.shape, y_train.shape)
# print(x_cv_bow.shape, y_cv.shape)
# print(x_test_bow.shape, y_test.shape)

print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(87398, 19) (87398,)
(21850, 19) (21850,)
===============================================================================================
After vectorizations
(87398, 9) (87398,)
(21850, 9) (21850,)
===============================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

In [60]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)
```

```
         vectorizer = CountVectorizer()
         vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on tra

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_cl_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].val
         X_test_cl_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].valu

         print("After vectorizations")
         print(X_train_cl_subcategories_ohe.shape, y_train.shape)
         print(X_test_cl_subcategories_ohe.shape, y_test.shape)
         print("="*100)

         print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(87398, 19) (87398,)
(21850, 19) (21850,)
===============================================================================================
After vectorizations
(87398, 30) (87398,)
(21850, 30) (21850,)
===============================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

```
In [61]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer = CountVectorizer()
         vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_school_state_ohe = vectorizer.transform(X_train['school_state'].values)
         X_test_school_state_ohe = vectorizer.transform(X_test['school_state'].values)

         print("After vectorizations")
         print(X_train_school_state_ohe.shape, y_train.shape)
         print(X_test_school_state_ohe.shape, y_test.shape)
         print("="*100)


         # print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
         # vectorizer = CountVectorizer()
         # x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
         # x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
         # x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

         # print(x_train_bow.shape, y_train.shape)
         # print(x_cv_bow.shape, y_cv.shape)
         # print(x_test_bow.shape, y_test.shape)

         print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(87398, 19) (87398,)
(21850, 19) (21850,)
==============================================================================================
After vectorizations
(87398, 51) (87398,)
(21850, 51) (21850,)
==============================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

```
In [77]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer = CountVectorizer(vocabulary=list(sorted_fix_dict.keys()), lowercase=False
         vectorizer.fit(X_train['teacher_prefix'].values.astype('U')) # fit has to happen only
         #https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueer
         # we use the fitted CountVectorizer to convert the text to vector
         X_train_teacher_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].values.ast
         X_test_teacher_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].values.astyp

         print("After vectorizations")
         print(X_train_teacher_prefix_ohe.shape, y_train.shape)
         print(X_test_teacher_prefix_ohe.shape, y_test.shape)
         print("="*100)


         # print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
         # vectorizer = CountVectorizer()
         # x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
         # x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
         # x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

         # print(x_train_bow.shape, y_train.shape)
         # print(x_cv_bow.shape, y_cv.shape)
         # print(x_test_bow.shape, y_test.shape)

         print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(87398, 19) (87398,)
(21850, 19) (21850,)
===============================================================================================
After vectorizations
(87398, 5) (87398,)
(21850, 5) (21850,)
===============================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

```
In [69]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer1 = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=
         vectorizer1.fit(X_train['grade_cat_list'].values) # fit has to happen only on train d

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_grade_ohe = vectorizer1.transform(X_train['grade_cat_list'].values)
         X_test_grade_ohe = vectorizer1.transform(X_test['grade_cat_list'].values)

         print("After vectorizations")
         print(X_train_grade_ohe.shape, y_train.shape)
         print(X_test_grade_ohe.shape, y_test.shape)
         print("="*100)


         # print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
         # vectorizer = CountVectorizer()
         # x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
         # x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
         # x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

         # print(x_train_bow.shape, y_train.shape)
         # print(x_cv_bow.shape, y_cv.shape)
         # print(x_test_bow.shape, y_test.shape)

         print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(87398, 19) (87398,)
(21850, 19) (21850,)
===============================================================================================
After vectorizations
(87398, 4) (87398,)
(21850, 4) (21850,)
===============================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

#
2.3 Make Data Model Ready: encoding eassay, and project_title

3 Text - BOW

```
In [70]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer = CountVectorizer(min_df=10, max_features=5000)
         vectorizer.fit(train_preprocessed_essays) # fit has to happen only on train data

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_essay_bow = vectorizer.transform(train_preprocessed_essays)
         X_test_essay_bow = vectorizer.transform(test_preprocessed_essays)

         print("After vectorizations")
         print(X_train_essay_bow.shape, y_train.shape)
         print(X_test_essay_bow.shape, y_test.shape)
         print("="*100)


         # print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
         # vectorizer = CountVectorizer()
         # x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
         # x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
         # x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

         # print(x_train_bow.shape, y_train.shape)
         # print(x_cv_bow.shape, y_cv.shape)
         # print(x_test_bow.shape, y_test.shape)

         print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(87398, 19) (87398,)
(21850, 19) (21850,)
===============================================================================================
After vectorizations
(87398, 5000) (87398,)
(21850, 5000) (21850,)
===============================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

4 Text - TfIdf

```
In [71]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer = TfidfVectorizer(min_df=10, max_features=5000)
         vectorizer.fit(train_preprocessed_essays) # fit has to happen only on train data

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_essay_tf = vectorizer.transform(train_preprocessed_essays)
         X_test_essay_tf = vectorizer.transform(test_preprocessed_essays)

         print("After vectorizations")
         print(X_train_essay_tf.shape, y_train.shape)
         print(X_test_essay_tf.shape, y_test.shape)
         print("="*100)


         # print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
         # vectorizer = CountVectorizer()
         # x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
         # x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
         # x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

         # print(x_train_bow.shape, y_train.shape)
         # print(x_cv_bow.shape, y_cv.shape)
         # print(x_test_bow.shape, y_test.shape)

         print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(87398, 19) (87398,)
(21850, 19) (21850,)
===============================================================================================
After vectorizations
(87398, 5000) (87398,)
(21850, 5000) (21850,)
===============================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

1.5 Appling NB on different kind of featurization as mentioned in the instructions

Apply NB on different kind of featurization as mentioned in the instructions For Every model that you work on make sure you do the step 2 and step 3 of instrucations

5 Concatinating all the features

```
In [81]: from scipy.sparse import hstack
         # with the same hstack function we are concatinating a sparse matrix and a dense mati
         X_Bow_train = hstack((X_train_essay_bow, tr_price_standardized, tr_quantity_standardiz
                              X_train_cl_subcategories_ohe, X_train_school_state_ohe, X_train_

         X_Bow_train=X_Bow_train.tocsr()
         print(X_Bow_train.shape, y_train.shape)
```

```
(87398, 5101) (87398,)
```

```
In [82]: X_Bow_test = hstack((X_test_essay_bow, te_price_standardized, te_quantity_standardized
                              X_test_cl_subcategories_ohe, X_test_school_state_ohe, X_test_tea

         X_Bow_test = X_Bow_test.tocsr()
         print(X_Bow_test.shape, y_test.shape)
```

```
(21850, 5101) (21850,)
```

```
In [0]: #TfIdf
```

```
In [83]: from scipy.sparse import hstack
         # with the same hstack function we are concatinating a sparse matrix and a dense mati
         X_tf_train = hstack((X_train_essay_tf, tr_price_standardized, tr_quantity_standardized
                              X_train_cl_subcategories_ohe, X_train_school_state_ohe, X_train_


         X_tf_train = X_tf_train.tocsr()
         print(X_tf_train.shape, y_train.shape)
```

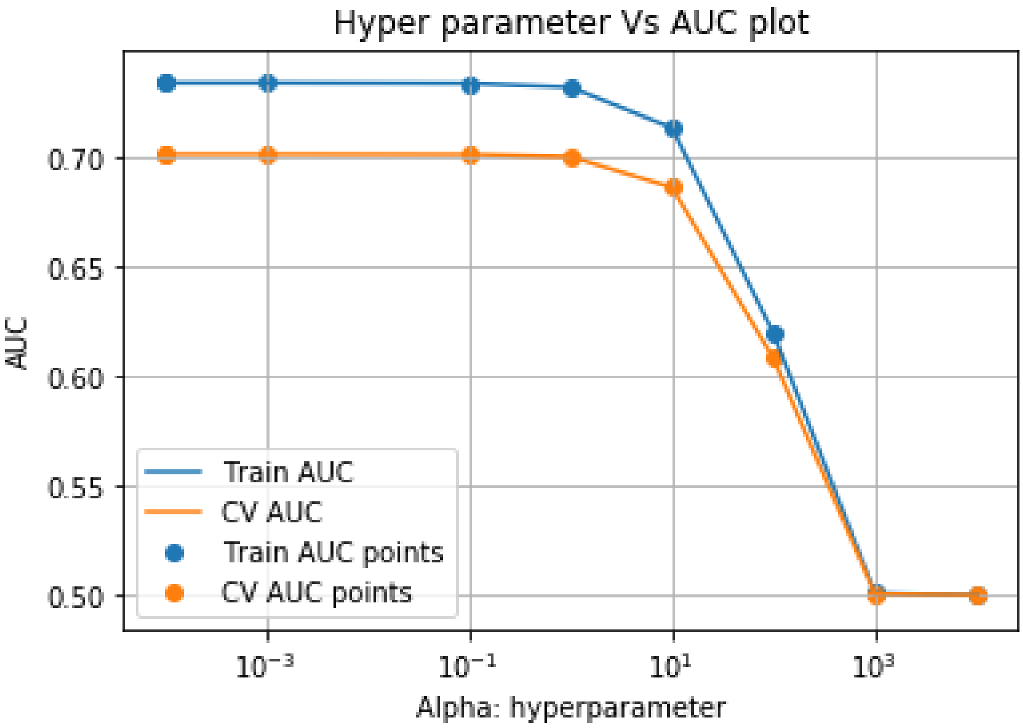
```
(87398, 5101) (87398,)
```

```
In [84]: X_tf_test = hstack((X_test_essay_tf, te_quantity_standardized, te_price_standardized,
                              X_test_cl_subcategories_ohe, X_test_school_state_ohe, X_test_tea

         X_tf_test = X_tf_test.tocsr()
         print(X_tf_test.shape, y_test.shape)
```

```
(21850, 5101) (21850,)
```

6 Set : 1

```
In [85]: # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearch
         from sklearn.model_selection import GridSearchCV
         from scipy.stats import randint as sp_randint
         from sklearn.model_selection import RandomizedSearchCV
         from sklearn.naive_bayes import MultinomialNB

         alpha=[0.0001,0.0001,0.001,0.1,1,10,100,1000,10000]
         MNB = MultinomialNB()
         parameters = dict(alpha=[0.0001,0.0001,0.001,0.1,1,10,100,1000,10000])
         clf = GridSearchCV(MNB, parameters, scoring='roc_auc', return_train_score=True)
         clf.fit(X_Bow_train, y_train)

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

         train_auc= results['mean_train_score']
         train_auc_std= results['std_train_score']
         cv_auc = results['mean_test_score']
         cv_auc_std= results['std_test_score']
         #K =  results['param_alpha']

         plt.plot(alpha, train_auc, label='Train AUC')
         # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
         # plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha

         plt.plot(alpha, cv_auc, label='CV AUC')
         # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
         # plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='

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

         plt.xscale('log')# we take the log in the x axis
         plt.legend()
         plt.xlabel("Alpha: hyperparameter")
         plt.ylabel("AUC")
         plt.title("Hyper parameter Vs AUC plot")
         plt.grid()
         plt.show()

         results.head()
```

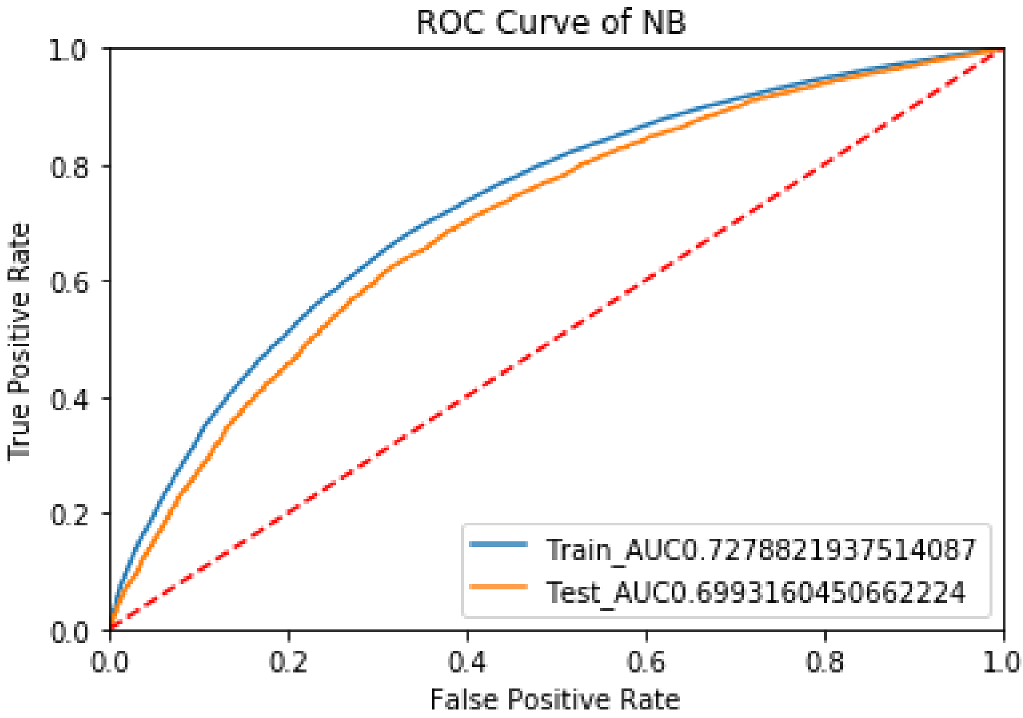


```
Out[85]:    mean_fit_time  std_fit_time  ...  mean_train_score  std_train_score
         0       0.079387      0.001326  ...          0.733684         0.000650
         1       0.079662      0.000581  ...          0.733684         0.000650
         2       0.079192      0.001937  ...          0.733678         0.000653
         3       0.078312      0.000674  ...          0.733472         0.000670
         4       0.078341      0.000758  ...          0.731736         0.000704

         [5 rows x 21 columns]
```

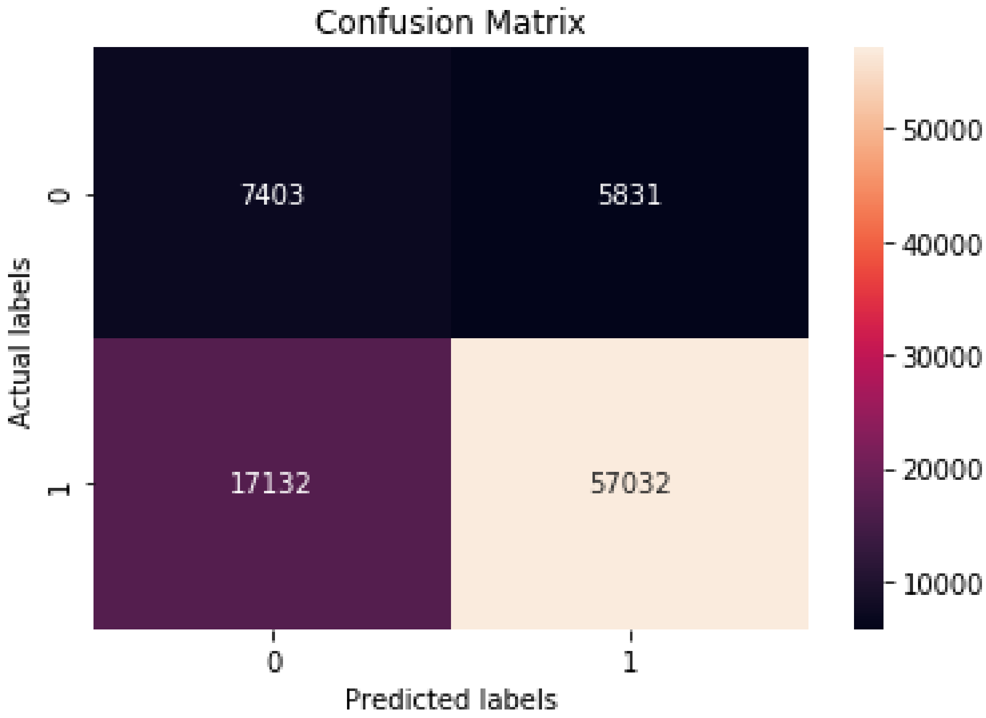
```
In [86]: #https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearch
         print(clf.best_estimator_)
         print(clf.best_index_)
         a=clf.best_params_["alpha"]
         print("Best Parameter Found:- ",a)
         print(clf.best_score_)
```

```
MultinomialNB(alpha=0.001, class_prior=None, fit_prior=True)
2
Best Parameter Found:-  0.001
0.7013408748453728
```

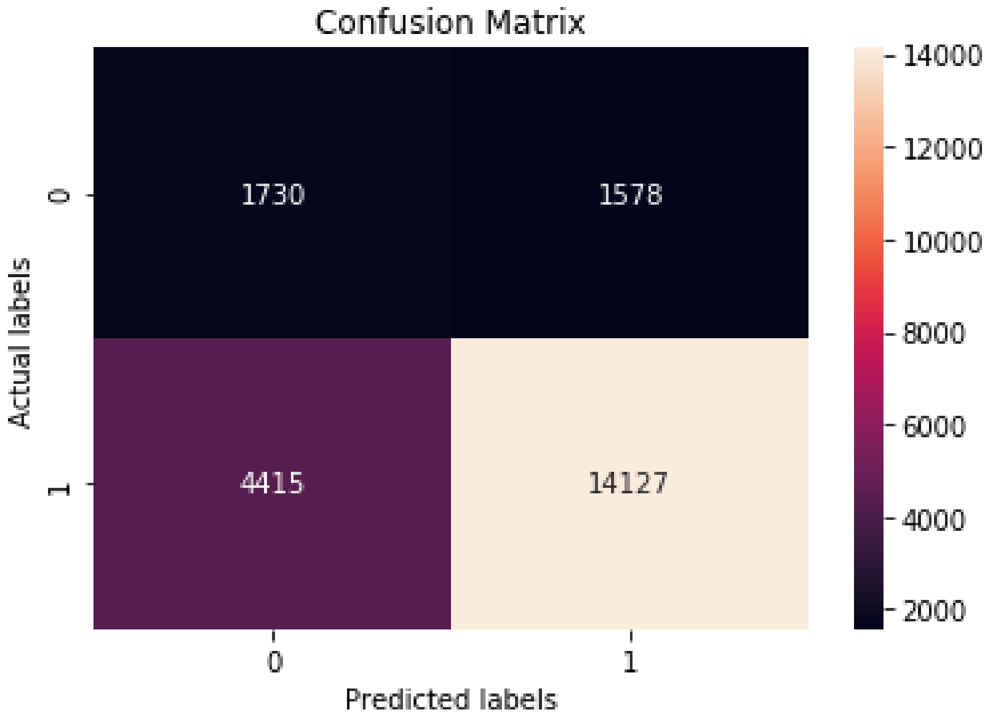
```
In [87]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sk
         from sklearn.metrics import roc_curve, auc
         MNB = MultinomialNB(alpha=a) # n_jobs=-1 means parallel operations
         MNB.fit(X_Bow_train, y_train)

Out[87]: MultinomialNB(alpha=0.001, class_prior=None, fit_prior=True)

In [88]: from  sklearn.metrics import roc_curve
         from sklearn.metrics import auc
         import matplotlib.pyplot as plt


         score_roc_train = MNB.predict_proba(X_Bow_train)
         fpr_train, tpr_train, threshold_train = roc_curve(y_train, score_roc_train[:,1])
         roc_auc_train = auc(fpr_train, tpr_train)

         score_roc_test = MNB.predict_proba(X_Bow_test)
         fpr_test, tpr_test, threshold_test = roc_curve(y_test, score_roc_test[:,1])
         roc_auc_test = auc(fpr_test, tpr_test)


         plt.plot(fpr_train, tpr_train, label = "Train_AUC"+str(auc(fpr_train, tpr_train)))
         plt.plot(fpr_test, tpr_test, label = "Test_AUC"+str(auc(fpr_test, tpr_test)))
         plt.legend(loc = 'lower right')

         plt.plot([0, 1], [0, 1],'r--')
         plt.xlim([0, 1])
         plt.ylim([0, 1])

         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.title('ROC Curve of NB ')
         plt.show()
```



```
In [89]: #https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
         import seaborn as sns
         import matplotlib.pyplot as plt
         ax= plt.subplot()
         sns.heatmap(confusion_matrix(y_train, MNB.predict(X_Bow_train)), annot=True, ax = ax,

         # labels, title and ticks
         ax.set_xlabel('Predicted labels');
         ax.set_ylabel('Actual labels');
         ax.set_title('Confusion Matrix');
```



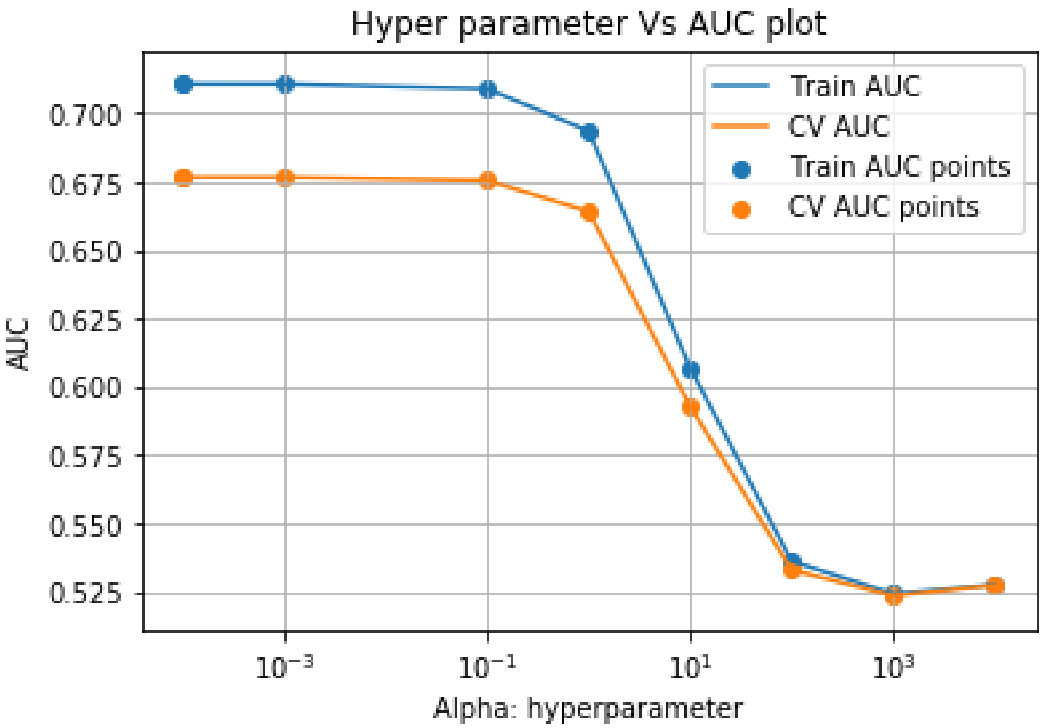
```
In [90]: #https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
         #Test Confusion Matrix
         import seaborn as sns
         import matplotlib.pyplot as plt
         ax= plt.subplot()
         sns.heatmap(confusion_matrix(y_test, MNB.predict(X_Bow_test)), annot=True, ax = ax,fm

         # labels, title and ticks
         ax.set_xlabel('Predicted labels');
         ax.set_ylabel('Actual labels');
         ax.set_title('Confusion Matrix');
```



7 Set : 2

```
In [91]: # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearc
         from sklearn.model_selection import GridSearchCV
         from scipy.stats import randint as sp_randint
         from sklearn.model_selection import RandomizedSearchCV

         alpha=[0.0001,0.0001,0.001,0.1,1,10,100,1000,10000]
         MNB1 = MultinomialNB()
         parameters = dict(alpha=[0.0001,0.0001,0.001,0.1,1,10,100,1000,10000])
         clf1 = GridSearchCV(MNB1, parameters, scoring='roc_auc',return_train_score=True)
         clf1.fit(X_tf_train, y_train)

         results = pd.DataFrame.from_dict(clf1.cv_results_)
         results = results.sort_values(['param_alpha'])

         train_auc= results['mean_train_score']
         train_auc_std= results['std_train_score']
         cv_auc = results['mean_test_score']
         cv_auc_std= results['std_test_score']
         #K =  results['param_alpha']

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

```
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=

plt.plot(alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='

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

plt.xscale('log')# we take the log in the x axis
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



```
Out[91]:    mean_fit_time  std_fit_time  ...  mean_train_score  std_train_score
         0       0.080035      0.005750  ...          0.710610         0.000673
         1       0.075708      0.000932  ...          0.710610         0.000673
```

```
2       0.075102      0.000728  ...          0.710589         0.000675
3       0.075453      0.000860  ...          0.708872         0.000703
4       0.074701      0.000257  ...          0.693549         0.000760

[5 rows x 21 columns]
```

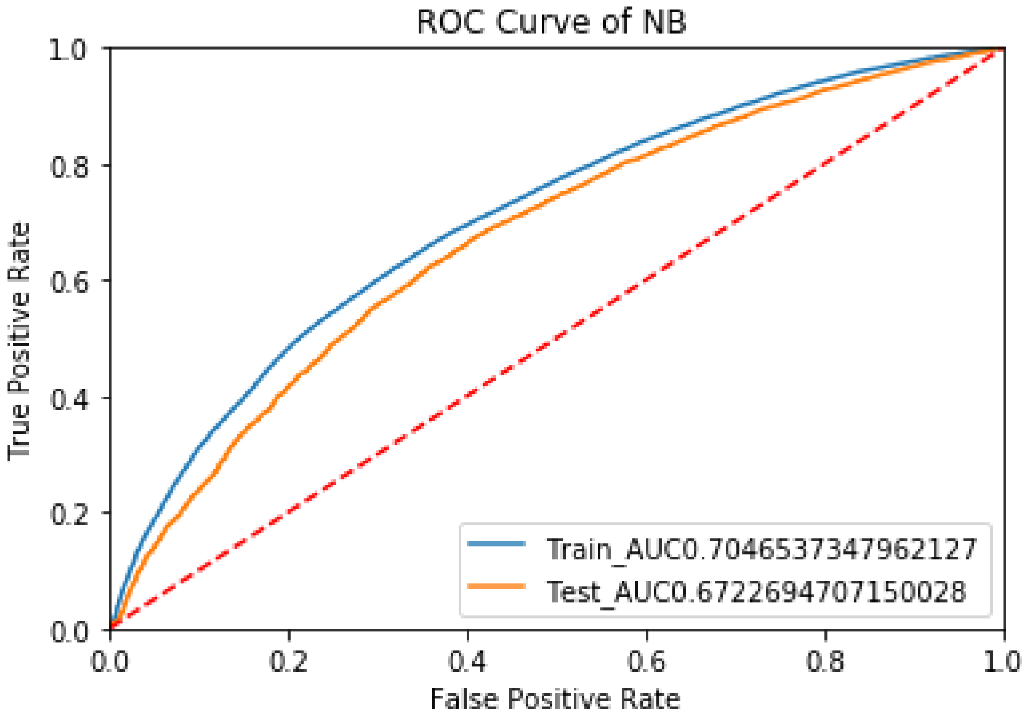
```
In [92]: #https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC
         print(clf.best_estimator_)
         print(clf.best_index_)
         a1=clf.best_params_["alpha"]
         print("Best Parameter Found:- ",clf.best_params_)
         print(clf.best_score_)
```

```
MultinomialNB(alpha=0.001, class_prior=None, fit_prior=True)
2
Best Parameter Found:-  {'alpha': 0.001}
0.7013408748453728
```

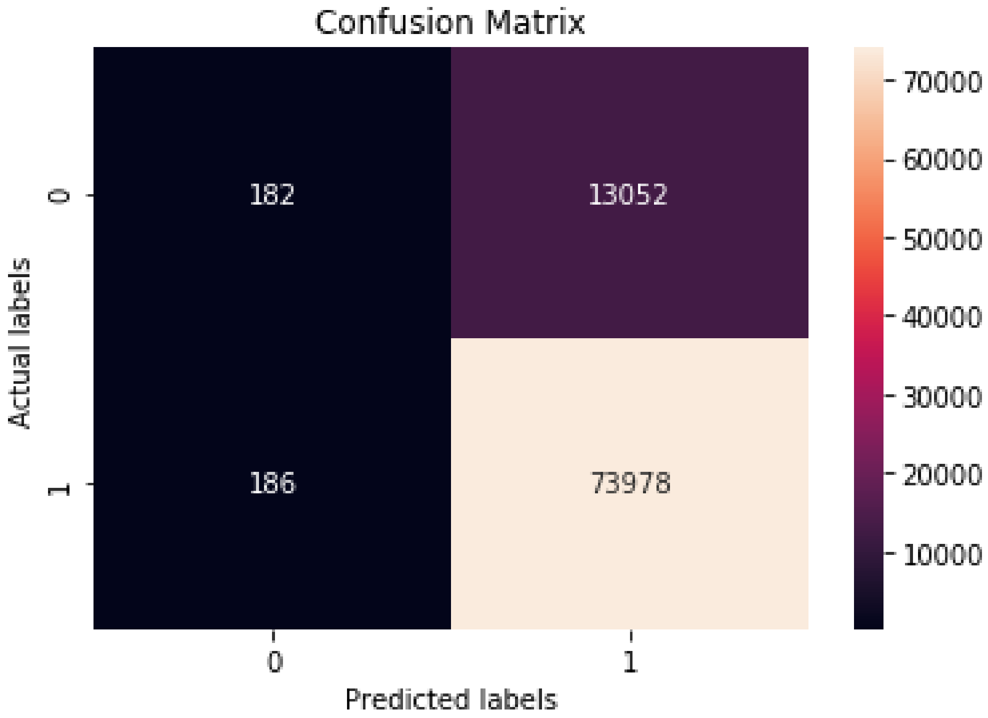
```
In [93]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sk
         from sklearn.metrics import roc_curve, auc
         MNB1 = MultinomialNB(alpha=a1) # n_jobs=-1 means parallel operations
         MNB1.fit(X_tf_train, y_train)
```

```
Out[93]: MultinomialNB(alpha=0.001, class_prior=None, fit_prior=True)
```

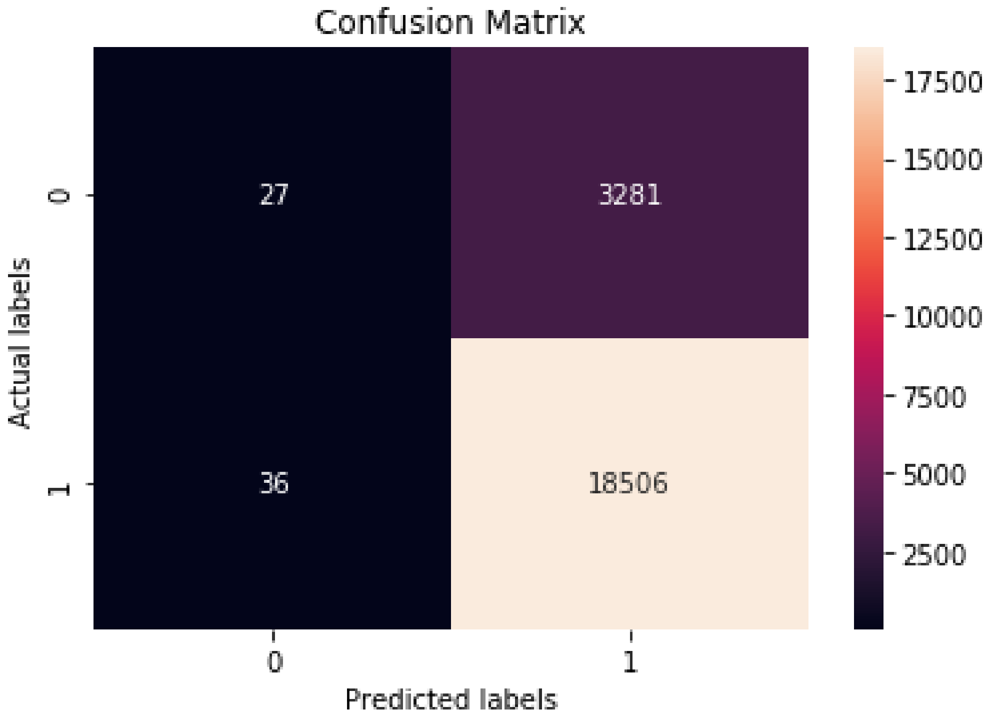
```
In [94]: from  sklearn.metrics import roc_curve
         from sklearn.metrics import auc
         import matplotlib.pyplot as plt


         score_roc_train = MNB1.predict_proba(X_tf_train)
         fpr_train, tpr_train, threshold_train = roc_curve(y_train, score_roc_train[:,1])
         roc_auc_train = auc(fpr_train, tpr_train)

         score_roc_test = MNB1.predict_proba(X_tf_test)
         fpr_test, tpr_test, threshold_test = roc_curve(y_test, score_roc_test[:,1])
         roc_auc_test = auc(fpr_test, tpr_test)


         plt.plot(fpr_train, tpr_train, label = "Train_AUC"+str(auc(fpr_train, tpr_train)))
         plt.plot(fpr_test, tpr_test, label = "Test_AUC"+str(auc(fpr_test, tpr_test)))
         plt.legend(loc = 'lower right')

         plt.plot([0, 1], [0, 1],'r--')
         plt.xlim([0, 1])
         plt.ylim([0, 1])

         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.title('ROC Curve of NB ')
         plt.show()
```



```
In [95]: #https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
         import seaborn as sns
         import matplotlib.pyplot as plt
         ax= plt.subplot()
         sns.heatmap(confusion_matrix(y_train, MNB.predict(X_tf_train)), annot=True, ax = ax,f

         # labels, title and ticks
         ax.set_xlabel('Predicted labels');
         ax.set_ylabel('Actual labels');
         ax.set_title('Confusion Matrix');
```



```
In [96]: #https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
         import seaborn as sns
         import matplotlib.pyplot as plt
         ax= plt.subplot()
         sns.heatmap(confusion_matrix(y_test, MNB.predict(X_tf_test)), annot=True, ax = ax,fmt=

         # labels, title and ticks
         ax.set_xlabel('Predicted labels');
         ax.set_ylabel('Actual labels');
         ax.set_title('Confusion Matrix');
```



8 Top 10 features (negatives and positives)

9 Applying top 10 features on BoW

```
In [0]: #https://stackoverflow.com/questions/54988116/why-am-i-getting-almost-same-top-10-feat
```

```
In [0]: essays_features_tf=vectorizer3.get_feature_names()
```

```
In [100]: sorted_idx = np.argsort(MNB.feature_log_prob_[1] )[-10:]

          for i in sorted_idx:
              print(vectorizer3.get_feature_names()[i])
```

```
cliche
embracing
embraces
busy
chew
complexity
alway
chicagoland
defensive
discouraging
```

```
In [101]: sorted_idx = np.argsort(-1 * MNB.feature_log_prob_[0] )[0:11]

          for i in sorted_idx:
              print(vectorizer3.get_feature_names()[i])
```

```
discouraging
defensive
chicagoland
alway
complexity
chew
busy
embracing
embraces
commonalities
cliche
```

10 Conclusion

```
In [102]: # Please compare all your models using Prettytable library
          #how to use pretty table http://zetcode.com/python/prettytable/
          from prettytable import PrettyTable
          tb = PrettyTable()
          tb.field_names= ("Vectorizer", "HyperParameter", "AUC")
          tb.add_row(["BOW", a, 0.699])
          tb.add_row(["Tf-Idf",a1 , 0.672])
          print(tb.get_string(titles = "Naive Bayes - Observations"))
          #print(tb)
```

```
+------------+----------------+-------+
| Vectorizer | HyperParameter |  AUC  |
+------------+----------------+-------+
|    BOW     |     0.001      | 0.699 |
|   Tf-Idf   |     0.001      | 0.672 |
+------------+----------------+-------+
```