Donors Choose Dataset

- ***Hyper-parameterization***
- ***Model Evaluation***
- ***Model Performance on CV and Test sets***

Notebook Contents

Importing_Packages

In [105…

```
import logging
logging.basicConfig(filename="Donors_Choose.log",
                    filemode='w',
                    level=logging.INFO,
                    format="%(asctime)s : %(levelname)s : %(message)s")

try :
    logging.info("#### Packages import ####")
    ## Some basic libaries
    import os
    import sys
    import re           # Tutorial about Python regular expressions: https://pymotw.c
    import string
    import shutil
    import warnings
    import pickle
    import sqlite3
    from tqdm import tqdm
    from collections import Counter
    from scipy import sparse

    ## Data Pre-processing Libraries
    import numpy as np
    import pandas as pd

    ## Visualization Libraries
    import matplotlib.pyplot as plt
    from matplotlib.colors import ListedColormap
```

```
    ### Visualization :: Seaborn
    import seaborn as sns

    ### Visualization :: Plotly
    from chart_studio import plotly
    import plotly.offline as offline
    import plotly.graph_objs as go
    offline.init_notebook_mode()

    ## NLP
    import nltk

    ### NLP :: Stopwords
    from nltk.corpus import stopwords

    ### NLP :: Stemmer and Lemmatizer
    from nltk.stem.porter import PorterStemmer
    from nltk.stem import PorterStemmer
    from nltk.stem.wordnet import WordNetLemmatizer

    ### NLP :: Word2Vec
    from gensim.models import Word2Vec
    from gensim.models import KeyedVectors

    ### NLP :: Text Featurization Libraries
    from sklearn.feature_extraction.text import TfidfTransformer
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.feature_extraction.text import CountVectorizer

    ## Features Scalers/Standardizers/Normalizers
    from sklearn.preprocessing import StandardScaler, MinMaxScaler, Normalizer

    ## Cross-Validation and Data Splitting
    from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import train_test_split

    ## ML Algorithms
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.model_selection import GridSearchCV
    from sklearn.naive_bayes import MultinomialNB

    ## Performace Metrics
    from sklearn import metrics
    from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, precisio
    from sklearn.metrics import plot_roc_curve
except ImportError as ie:
    # Output expected ImportErrors
    logging.error(msg=ie.__class__.__name__  + " :: Missing Package --> " + ie.name)
except Exception as exception:
    # Output unexpected Exceptions
    logging.info("#### Exceptions other than ModuleImportError ####")
    logging.log(msg=(exception, False))
    logging.log(msg=exception.__class__.__name__ + " :: " + exception.name)
%matplotlib inline
```

Importing_Datasets

In [2]:

```
X_tr_bow = sparse.load_npz("X_tr_bow.npz")
X_cv_bow = sparse.load_npz("X_cv_bow.npz")
X_te_bow = sparse.load_npz("X_te_bow.npz")
X_tr_tfidf = sparse.load_npz("X_tr_tfidf.npz")
```

```
X_cv_tfidf = sparse.load_npz("X_cv_tfidf.npz")
X_te_tfidf = sparse.load_npz("X_te_tfidf.npz")
```

In [3]:
```
y_train = np.loadtxt('y_train.csv')
y_cv = np.loadtxt('y_cv.csv')
y_test = np.loadtxt('y_test.csv')
```

In [4]:
```
print(y_train.shape, y_cv.shape, y_test.shape)
```

```
(40500,) (4500,) (15000,)
```

Hyper-parameterization

- **Grid-SearchCV(TF-IDF)**

In [5]:
```
clf = MultinomialNB(class_prior=[0.5,0.5])

param_dist = {"alpha": [0.00001,0.0005,0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,5

grid_srch = GridSearchCV(estimator=clf,param_grid=param_dist,cv=10,scoring='roc_auc'
grid_srch.fit(X_tr_tfidf,y_train)
```

Out[5]:
```
GridSearchCV(cv=10, estimator=MultinomialNB(class_prior=[0.5, 0.5]),
             param_grid={'alpha': [1e-05, 0.0005, 0.0001, 0.005, 0.001, 0.05,
                                   0.01, 0.1, 0.5, 1, 5, 10, 50, 100]},
             return_train_score=True, scoring='roc_auc')
```

In [6]:
```
results = pd.DataFrame(grid_srch.cv_results_)
```

In [7]:
```
results['log_param_alpha'] = results['param_alpha'].apply(lambda val: np.round(np.lo
```

In [36]:
```
results.head(9)
```

Out[36]:

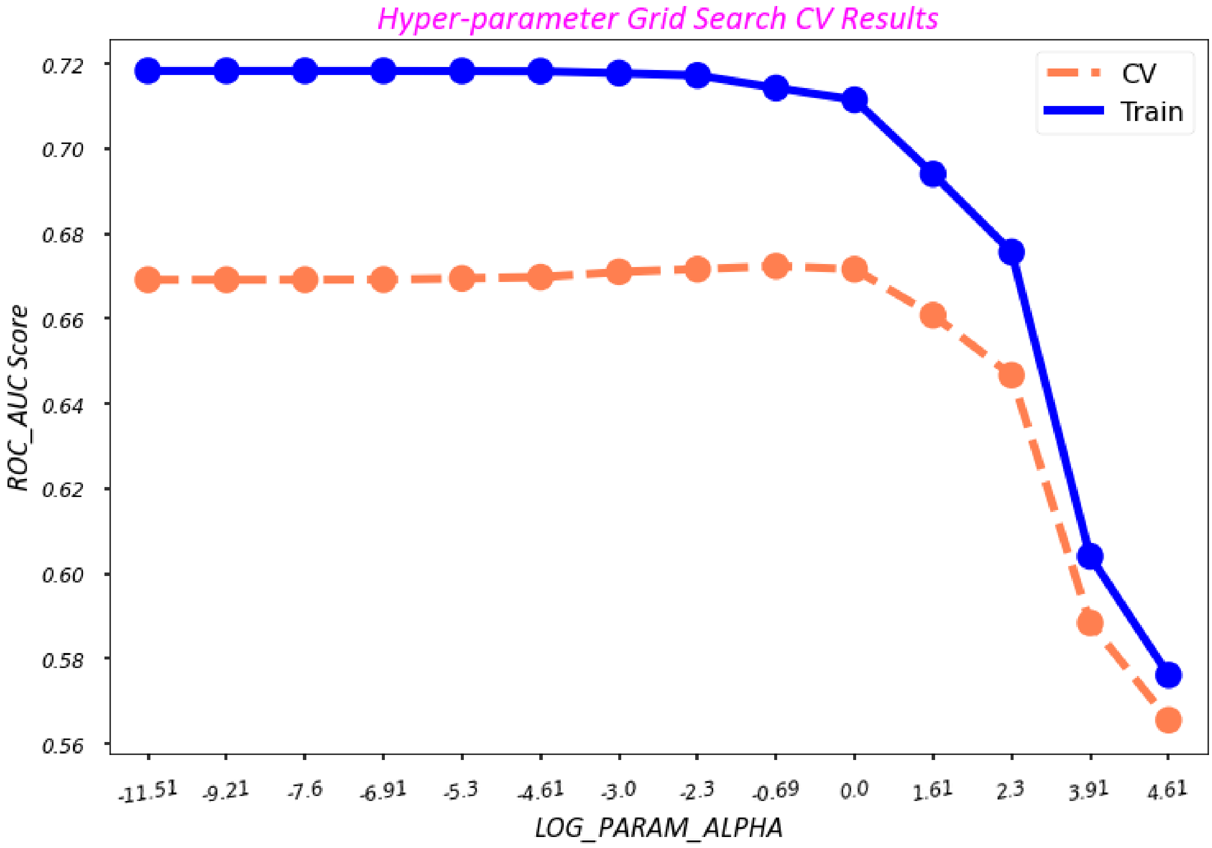
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_sc
0	0.127617	0.020068	0.011647	0.002209	1e-05	{'alpha': 1e-05}	0.700!
1	0.134299	0.023169	0.013303	0.002328	0.0005	{'alpha': 0.0005}	0.700!
2	0.126070	0.007432	0.014242	0.005230	0.0001	{'alpha': 0.0001}	0.700!
3	0.135696	0.020231	0.013402	0.002057	0.005	{'alpha': 0.005}	0.701(
4	0.127655	0.008282	0.014001	0.001733	0.001	{'alpha': 0.001}	0.700!
5	0.134524	0.009429	0.015358	0.004763	0.05	{'alpha': 0.05}	0.702(
6	0.130366	0.016529	0.013404	0.002578	0.01	{'alpha': 0.01}	0.701'
7	0.114000	0.008393	0.011202	0.001471	0.1	{'alpha': 0.1}	0.702(
8	0.163845	0.043132	0.017503	0.004129	0.5	{'alpha': 0.5}	0.703(

9 rows × 32 columns

In [8]:
```
## Global Variables
lbl_dict = {'family':'Calibri','size':18,'style':'oblique','color':'k'}
ttl_dict = {'family':'Calibri','size':21,'style':'oblique','color':'magenta'}
wdg_dict = {'linewidth': 1, 'edgecolor': 'black'}
```

In [34]:
```
def plot_cv_results(cv_results,col_x, metric_test='mean_test_score',metric_train='me
    """
    Description: This function is created for plotting the Train and CV ROC_AUC comp

    Inputs: It accepts below parameters:
        `cv_results` - Pandas DataFrame
            cv_results_ attribute of a GridSearchCV instance (or similar)
        `col_x` - str
            name of grid search parameter to plot on x axis
    """
    with plt.style.context('seaborn-poster'):
        plt.figure(figsize=(12,8))

        line1 = sns.pointplot(x=col_x, y=metric_test, data=cv_results, ci=99, n_boot
        line1.lines[0].set_linestyle("--")
        line1.lines[0].set_label('CV')
        line1.lines[0].set_color('coral')

        line2 = sns.pointplot(x=col_x, y=metric_train, data=cv_results, ci=99, n_boo
        line2.lines[1].set_linestyle("-")
        line2.lines[1].set_label('Train')
        line2.lines[1].set_color('b')

        plt.xlabel(str(col_x).upper(),fontdict=lbl_dict)
        plt.ylabel('ROC_AUC Score',fontdict=lbl_dict)
        plt.title("Hyper-parameter Grid Search CV Results",fontdict=ttl_dict)
        plt.xticks(size=12,style='oblique',rotation=10)
        plt.yticks(size=12,style='oblique')
        plt.legend()
        plt.show()
```

In [35]:
```
plot_cv_results(cv_results=results,col_x='log_param_alpha')
```



The best value alpha looks like 0.5 as it is the point where we have the highest CV score value and minimum gap or difference b/w Train and CV.

- **Grid-SearchCV(BOW)**

In [143…
```
clf = MultinomialNB(class_prior=[0.5,0.5])

param_dist = {"alpha": [0.00001,0.0005,0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,5

grid_srch = GridSearchCV(estimator=clf,param_grid=param_dist,cv=10,scoring='roc_auc'
grid_srch.fit(X_tr_bow,y_train)
```

Out[143…
```
GridSearchCV(cv=10, estimator=MultinomialNB(class_prior=[0.5, 0.5]),
             param_grid={'alpha': [1e-05, 0.0005, 0.0001, 0.005, 0.001, 0.05,
                                   0.01, 0.1, 0.5, 1, 5, 10, 50, 100]},
             return_train_score=True, scoring='roc_auc')
```

In [144…
```
results = pd.DataFrame(grid_srch.cv_results_)
```

In [145…
```
results['log_param_alpha'] = results['param_alpha'].apply(lambda val: np.round(np.lo
```

In [150…
```
results.tail(10)
```

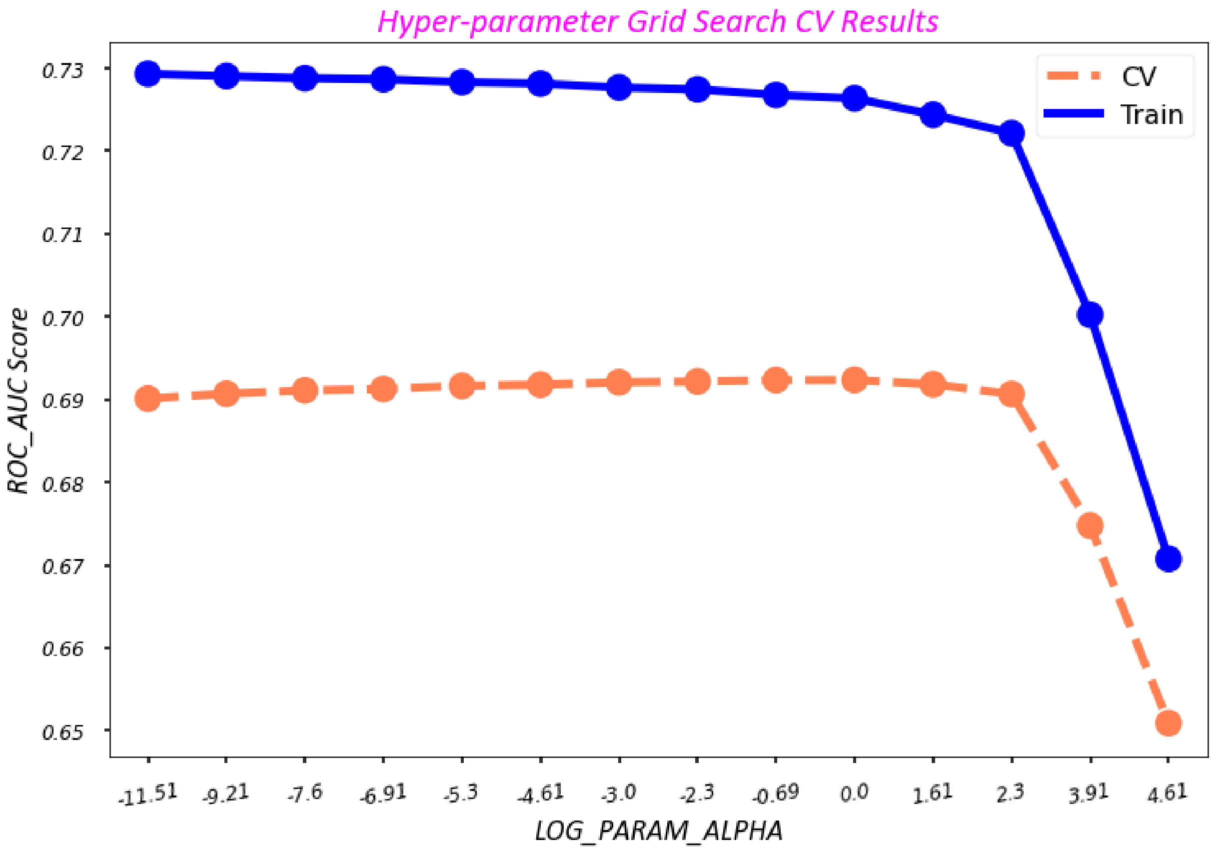
Out[150…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_s
4	0.145028	0.035384	0.015752	0.003721	0.001	{'alpha': 0.001}	0.71
5	0.132290	0.013549	0.014486	0.002537	0.05	{'alpha': 0.05}	0.71
6	0.128559	0.029332	0.015000	0.005253	0.01	{'alpha': 0.01}	0.71
7	0.119962	0.010759	0.015481	0.006581	0.1	{'alpha': 0.1}	0.71

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_s
8	0.143357	0.030854	0.017207	0.004688	0.5	{'alpha': 0.5}	0.71
9	0.128000	0.016388	0.015601	0.004177	1	{'alpha': 1}	0.71
10	0.157729	0.043314	0.016351	0.006744	5	{'alpha': 5}	0.71
11	0.134529	0.017570	0.013846	0.003566	10	{'alpha': 10}	0.71
12	0.128060	0.014818	0.017401	0.004387	50	{'alpha': 50}	0.70
13	0.122511	0.020997	0.021200	0.013044	100	{'alpha': 100}	0.67

10 rows × 32 columns

In [147…:
```
plot_cv_results(cv_results=results,col_x='log_param_alpha')
```



The best value alpha looks like 5 as it is the point where we have the highest CV score value and minimum gap or difference b/w Train and CV.

Model_Training_TFIDF

- **Training model on Tf-IDF encoded(Essays, Titles and Summaries) + Categorical + Numerical features**

In [49]:
```
mnb_model = MultinomialNB(alpha=0.5,class_prior=[0.5,0.5])
```

In [50]:
```
mnb_model.fit(X_tr_tfidf,y_train)
```

Out[50]:
```
MultinomialNB(alpha=0.5, class_prior=[0.5, 0.5])
```

Model_Testing_TFIDF

CV_Dataset_TFIDF

- **Testing on CV data**

In [51]:
```
y_pred = mnb_model.predict(X_cv_tfidf)
```

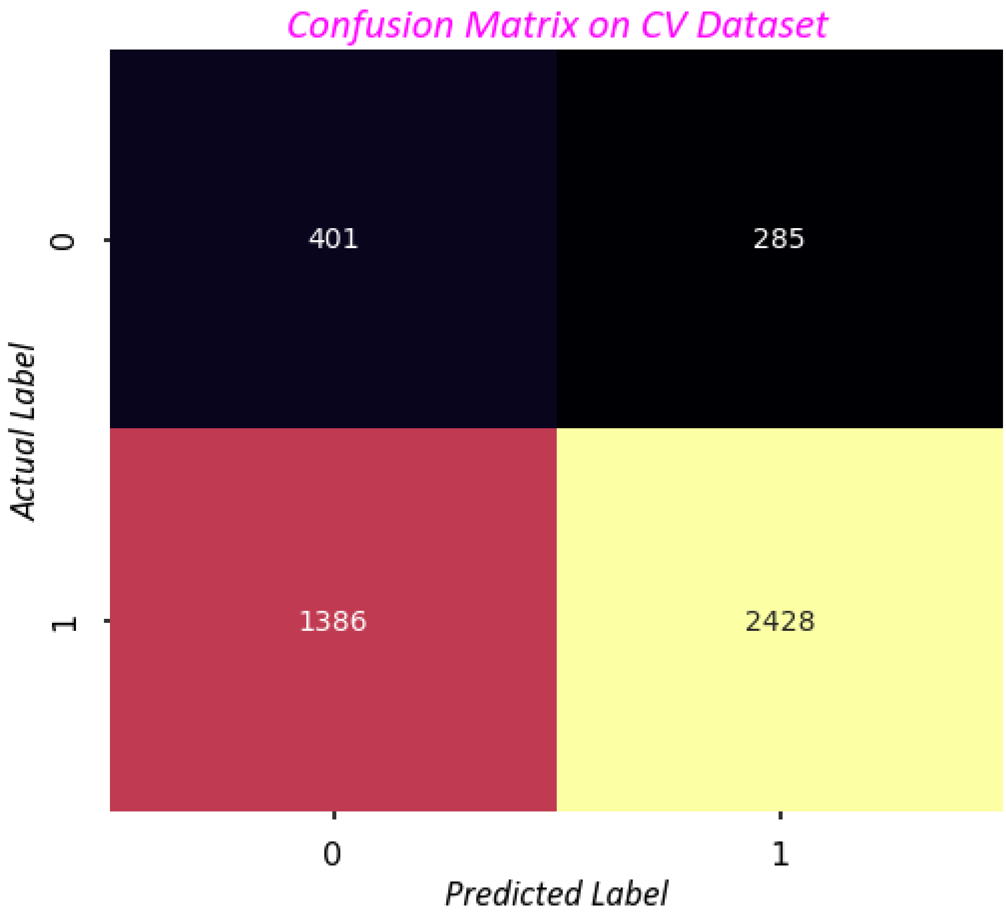
In [52]:
```
precision_score(y_cv,y_pred)
```

Out[52]:
```
0.8949502395871729
```

In [53]:
```
recall_score(y_cv,y_pred)
```

Out[53]:
```
0.6366019926586262
```

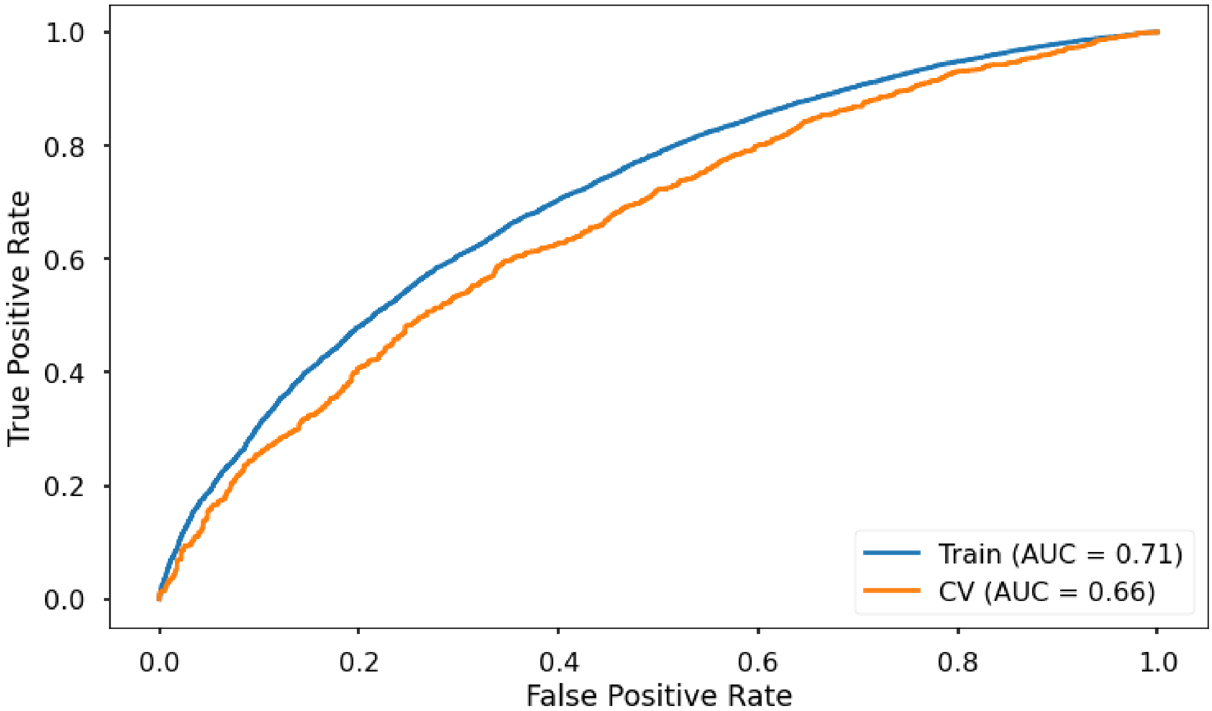
In [77]:
```
## Confusion Matrix on CV dataset
with plt.style.context('seaborn-poster'):
    plt.figure(figsize=(8,7))
    sns.heatmap(data=confusion_matrix(y_cv,y_pred),cmap='inferno',annot=True,annot_k
    plt.xlabel("Predicted Label",fontdict=lbl_dict)
    plt.ylabel("Actual Label",fontdict=lbl_dict)
    plt.title("Confusion Matrix on CV Dataset",fontdict=ttl_dict)
```



In [55]:
```
## ROC_AUC Score on CV dataset
roc_auc_score(y_cv,y_pred)
```

Out[55]:
```
0.6105750488074471
```

In [95]:
```
with plt.style.context('seaborn-poster'):
    fig, ax = plt.subplots(ncols=1,nrows=1,figsize=(12,7))
    train_roc_auc = plot_roc_curve(estimator=mnb_model,X=X_tr_tfidf,y=y_train,ax=ax,
    cv_roc_auc = plot_roc_curve(estimator=mnb_model,X=X_cv_tfidf,y=y_cv,ax=ax,name='
    plt.show()
```



Test_Dataset_TFIDF

- **Testing Model on Test data**

In [96]:
```
y_pred_te = mnb_model.predict(X_te_tfidf)
```

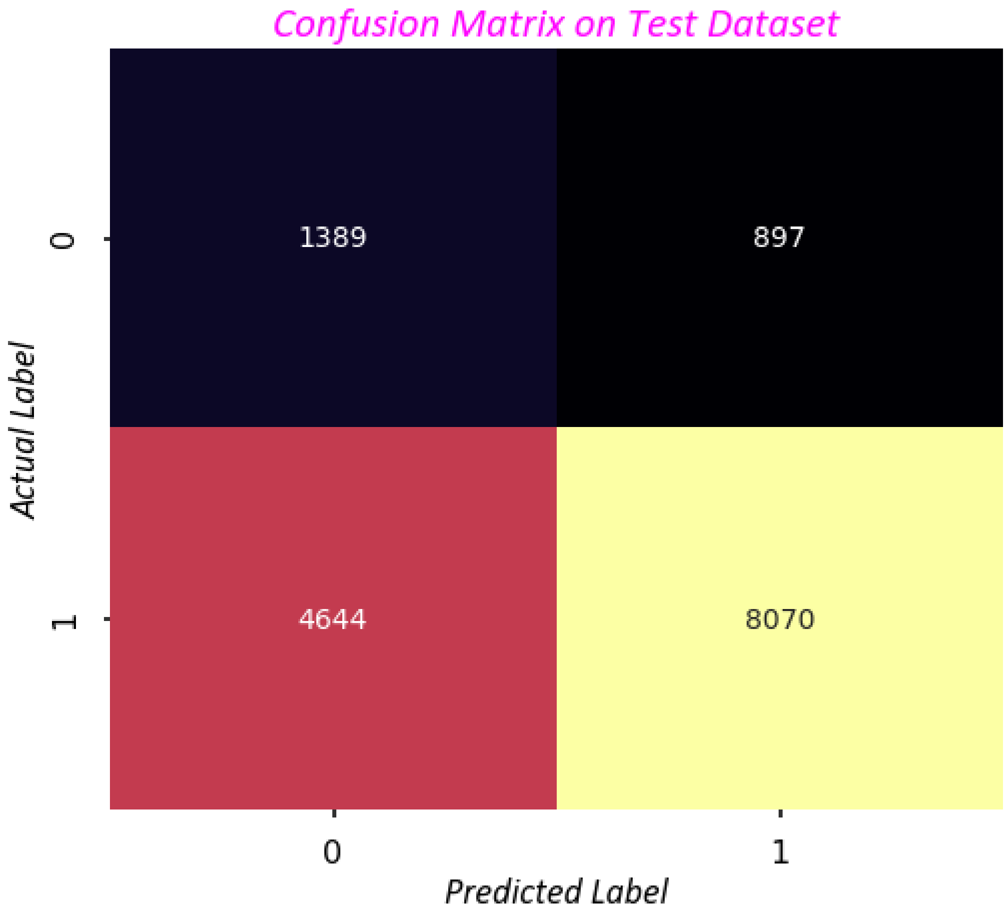
In [98]:
```
precision_score(y_test,y_pred_te)
```

Out[98]: 0.899966543994647

In [99]:
```
recall_score(y_test,y_pred_te)
```

Out[99]: 0.6347333647947145

In [100…
```
## Confusion Matrix on CV dataset
with plt.style.context('seaborn-poster'):
    plt.figure(figsize=(8,7))
    sns.heatmap(data=confusion_matrix(y_test,y_pred_te),cmap='inferno',annot=True,an
    plt.xlabel("Predicted Label",fontdict=lbl_dict)
    plt.ylabel("Actual Label",fontdict=lbl_dict)
    plt.title("Confusion Matrix on Test Dataset",fontdict=ttl_dict)
```



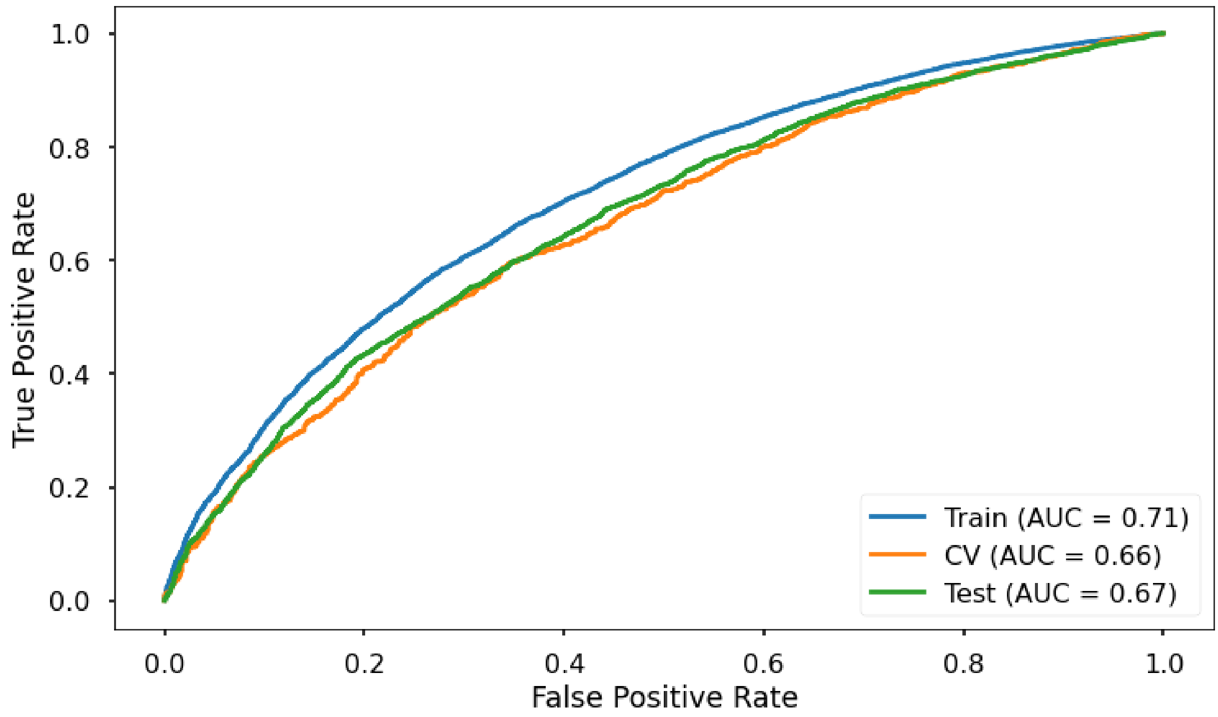
In [101…

```
## ROC_AUC Score on CV dataset
roc_auc_score(y_test,y_pred_te)
```

Out[101… 0.6211724566755725

In [104…

```
with plt.style.context('seaborn-poster'):
    fig, ax = plt.subplots(ncols=1,nrows=1,figsize=(12,7))
    train_roc_auc = plot_roc_curve(estimator=mnb_model,X=X_tr_tfidf,y=y_train,ax=ax,
    cv_roc_auc = plot_roc_curve(estimator=mnb_model,X=X_cv_tfidf,y=y_cv,ax=ax,name='
    test_roc_auc = plot_roc_curve(estimator=mnb_model,X=X_te_tfidf,y=y_test,ax=ax,na
    plt.show()
```



Model_Training_BOW

- **Training model on BOW encoded(Essays, Titles and Summaries) + Categorical + Numerical features**

In [151…
```
mnb_model_bow = MultinomialNB(alpha=5,class_prior=[0.5,0.5])
```

In [152…
```
mnb_model_bow.fit(X_tr_bow,y_train)
```

Out[152…
```
MultinomialNB(alpha=5, class_prior=[0.5, 0.5])
```

Model_Testing_BOW

CV_Dataset_BOW

- **Testing on CV data**

In [153…
```
y_pred_cv_bow = mnb_model_bow.predict(X_cv_bow)
```

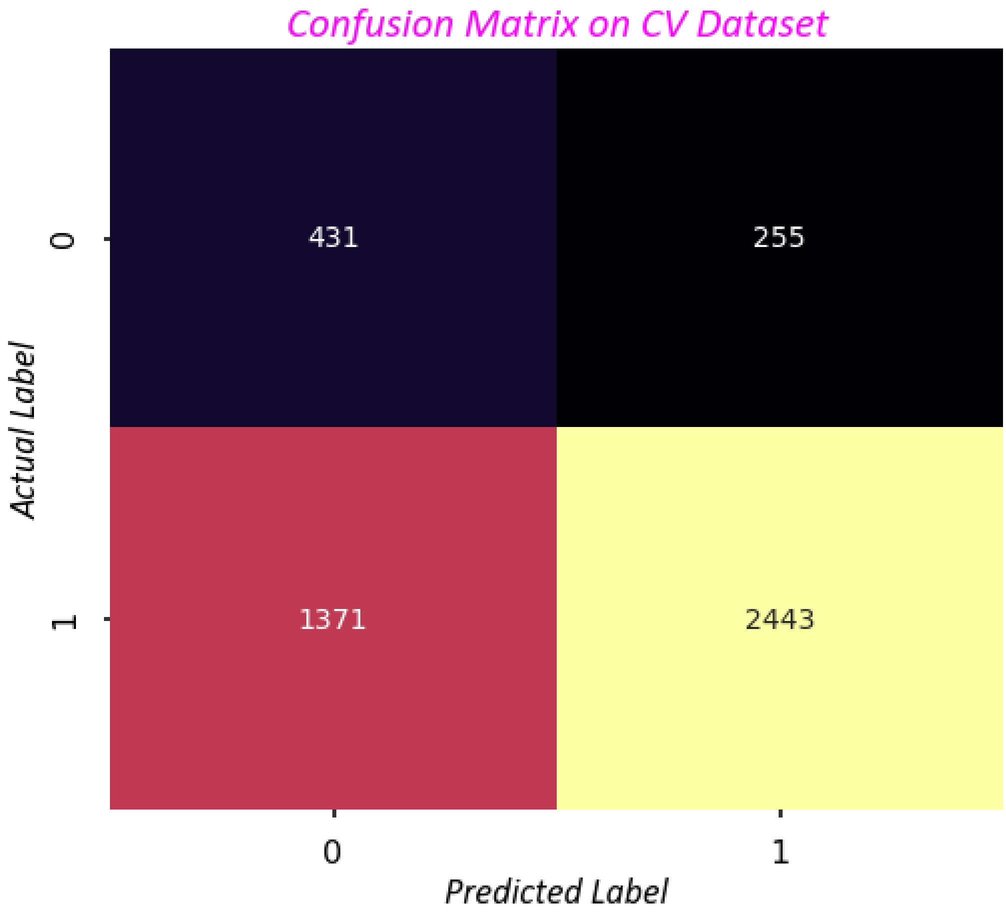
In [154…
```
precision_score(y_cv,y_pred_cv_bow)
```

Out[154…
```
0.9054855448480356
```

In [155…
```
recall_score(y_cv,y_pred_cv_bow)
```

Out[155…
```
0.640534871525957
```

In [156…
```
## Confusion Matrix on CV dataset
with plt.style.context('seaborn-poster'):
    plt.figure(figsize=(8,7))
    sns.heatmap(data=confusion_matrix(y_cv,y_pred_cv_bow),cmap='inferno',annot=True,
    plt.xlabel("Predicted Label",fontdict=lbl_dict)
    plt.ylabel("Actual Label",fontdict=lbl_dict)
    plt.title("Confusion Matrix on CV Dataset",fontdict=ttl_dict)
```



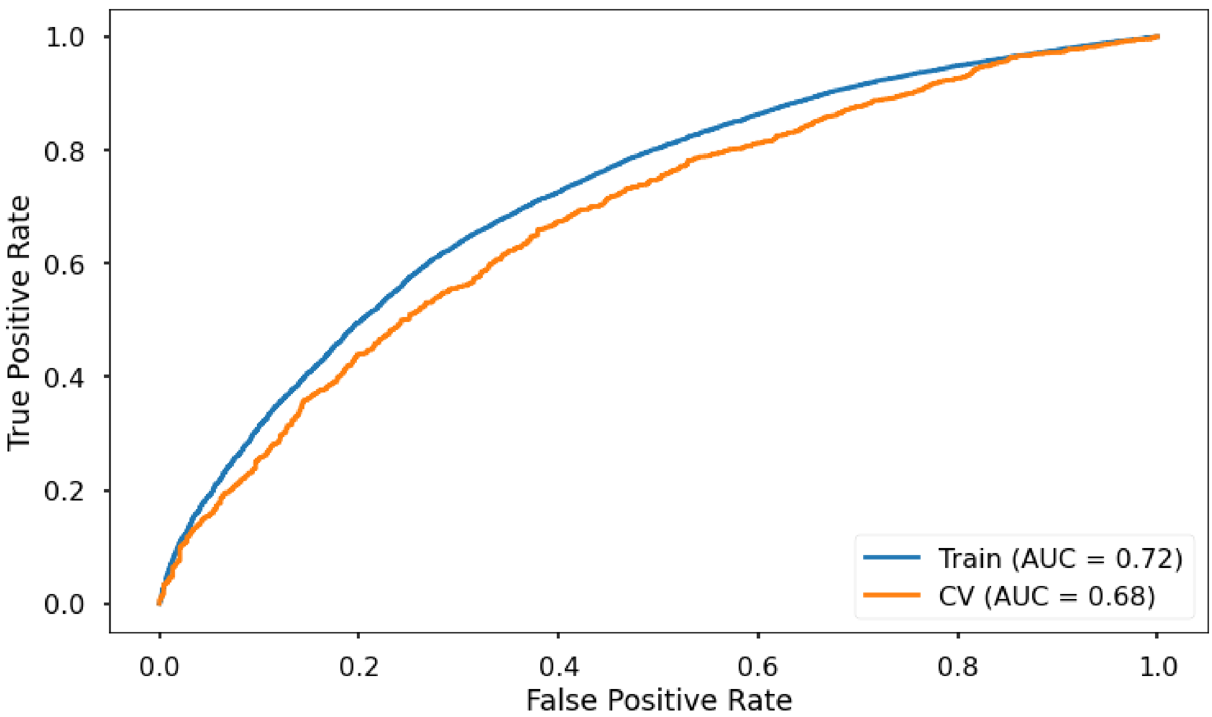
In [157…

```
## ROC_AUC Score on CV dataset
roc_auc_score(y_cv,y_pred_cv_bow)
```

Out[157… 0.6344073774539406

In [158…

```
with plt.style.context('seaborn-poster'):
    fig, ax = plt.subplots(ncols=1,nrows=1,figsize=(12,7))
    train_roc_auc = plot_roc_curve(estimator=mnb_model_bow,X=X_tr_bow,y=y_train,ax=a
    cv_roc_auc = plot_roc_curve(estimator=mnb_model_bow,X=X_cv_bow,y=y_cv,ax=ax,name
    plt.show()
```



Test_Dataset_BOW

- **Testing Model on Test data**

In [159...
```
y_pred_te_bow = mnb_model_bow.predict(X_te_bow)
```

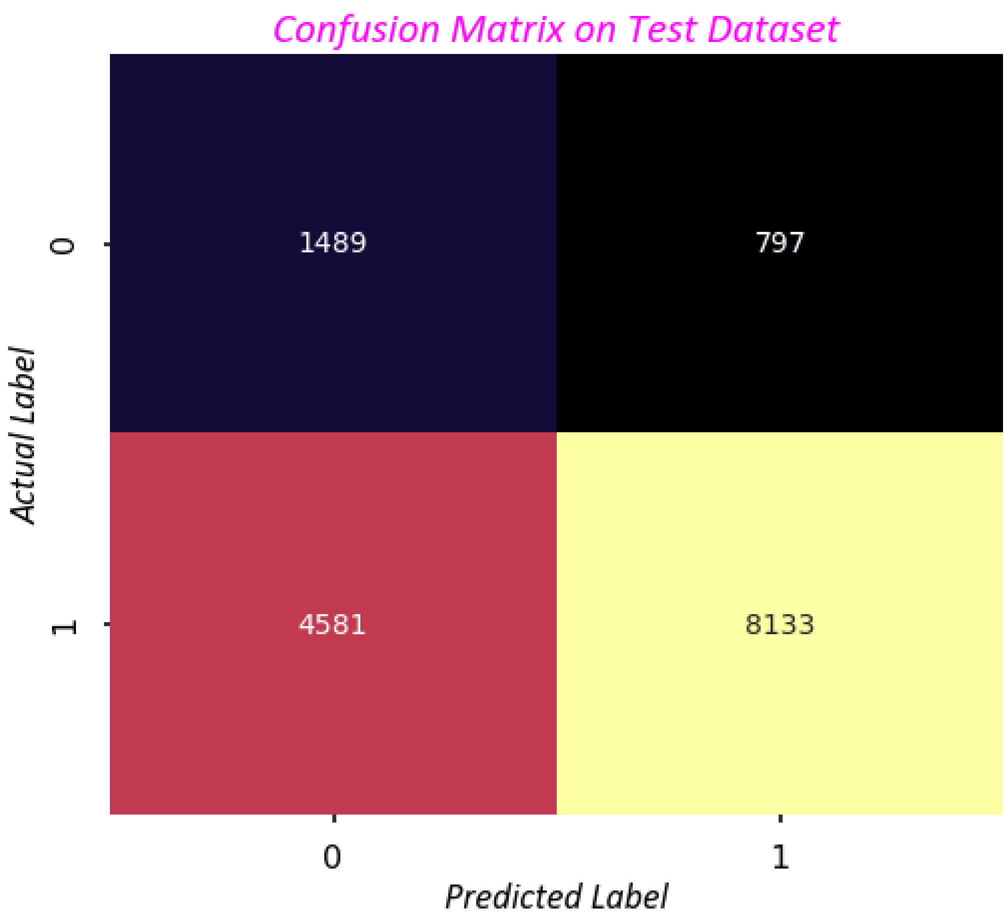
In [160...
```
precision_score(y_test,y_pred_te_bow)
```

Out[160... 0.9107502799552072

In [161...
```
recall_score(y_test,y_pred_te_bow)
```

Out[161... 0.6396885323265691

In [162...
```
## Confusion Matrix on CV dataset
with plt.style.context('seaborn-poster'):
    plt.figure(figsize=(8,7))
    sns.heatmap(data=confusion_matrix(y_test,y_pred_te_bow),cmap='inferno',annot=Tru
    plt.xlabel("Predicted Label",fontdict=lbl_dict)
    plt.ylabel("Actual Label",fontdict=lbl_dict)
    plt.title("Confusion Matrix on Test Dataset",fontdict=ttl_dict)
```

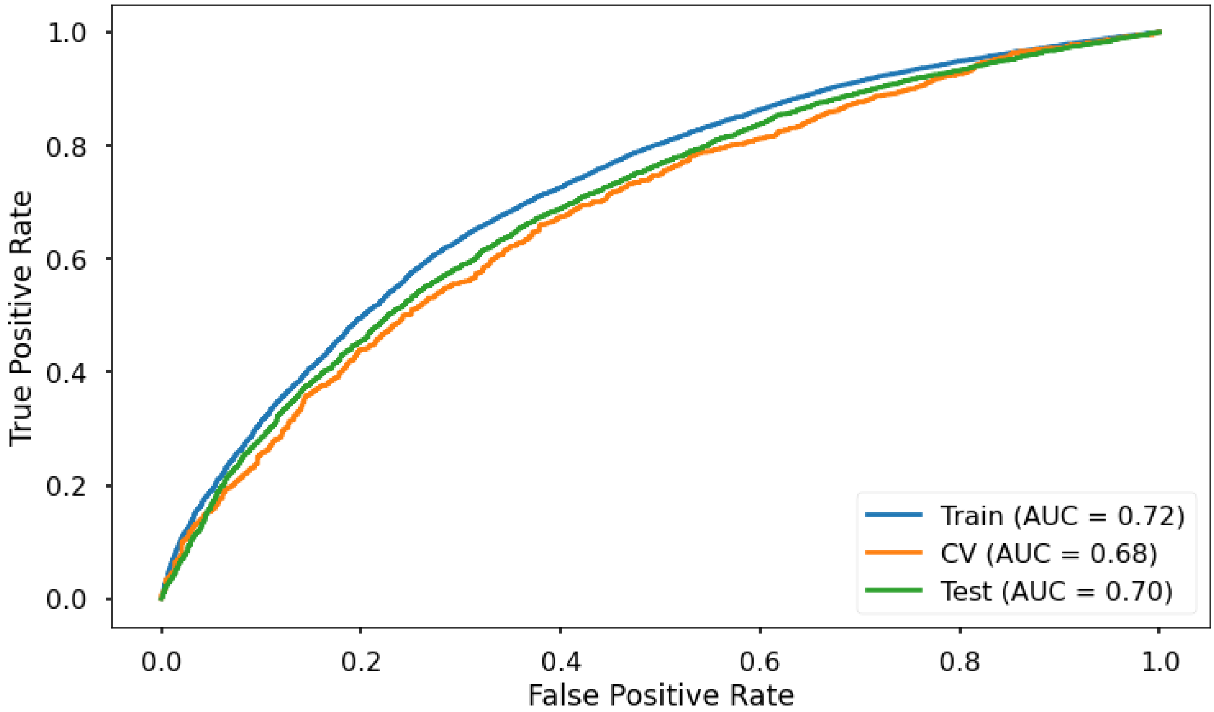


In [163...
```
## ROC_AUC Score on CV dataset
roc_auc_score(y_test,y_pred_te_bow)
```

Out[163... 0.6455223064082539

In [164...
```
with plt.style.context('seaborn-poster'):
    fig, ax = plt.subplots(ncols=1,nrows=1,figsize=(12,7))
    train_roc_auc = plot_roc_curve(estimator=mnb_model_bow,X=X_tr_bow,y=y_train,ax=a
    cv_roc_auc = plot_roc_curve(estimator=mnb_model_bow,X=X_cv_bow,y=y_cv,ax=ax,name
```

```
    test_roc_auc = plot_roc_curve(estimator=mnb_model_bow,X=X_te_bow,y=y_test,ax=ax,
    plt.show()
```



Top_20_Features

In [119…
```
with open('tfidf_feature_names', 'rb') as fp:
    tfidf_feature_names = pickle.load(fp)
```

In [122…
```
total_features = len(tfidf_feature_names)
total_features
```

Out[122… 4484

Negative Class Probabilities

In [114…
```
np.array(mnb_model.feature_log_prob_[0])
```

Out[114…
```
array([-11.3510112 ,  -5.29110949,  -3.71072778, ...,  -7.50889818,
        -8.73220893, -10.05231047])
```

Positive Class Probabilities

In [108…
```
mnb_model.feature_log_prob_[1]
```

Out[108…
```
array([-13.05818892,  -5.35578338,  -3.66266997, ...,  -8.05003694,
        -8.53496045,  -9.52786785])
```

Top 20 Positive Class Probabilities

In [140…
```
pos_class_tfidf_probs = mnb_model.feature_log_prob_[1, :].argsort()[::-1][:total_fea

top_20_pos_class_features = []

for pos_prob in pos_class_tfidf_probs[0:20]:
    top_20_pos_class_features.append(tfidf_feature_names[pos_prob])

print("*** Top 20 positive class features ***")
top_20_pos_class_features
```

```
*** Top 20 positive class features ***
```

Out[140…
```
['young students',
 'younger',
 'mrs',
 'grades_prek_2',
 'ms',
 'grades_3_5',
 'literacy__language',
 'grades_6_8',
 'math__science',
 'ca',
 'literacy__language__math__science',
 'students able',
 'grades_9_12',
 'mr',
 'health__sports',
 'history__geography__performing__arts',
 'history__geography__special__needs',
 'strengthen',
 'need',
 'student']
```

Top 20 Negative Class Probabilities

In [142…
```python
neg_class_tfidf_probs = mnb_model.feature_log_prob_[0, :].argsort()[::-1][:total_fea

top_20_neg_class_features = []

for neg_prob in neg_class_tfidf_probs[0:20]:
    top_20_neg_class_features.append(tfidf_feature_names[neg_prob])

print("*** Top 20 negative class features ***")
top_20_neg_class_features
```

```
*** Top 20 negative class features ***
```

Out[142…
```
['younger',
 'young students',
 'mrs',
 'grades_prek_2',
 'ms',
 'grades_3_5',
 'literacy__language',
 'math__science',
 'grades_6_8',
 'students able',
 'ca',
 'literacy__language__math__science',
 'grades_9_12',
 'mr',
 'health__sports',
 'tx',
 'strengthen',
 'need',
 'student',
 'history__geography__performing__arts']
```

Majority of the features are same, but their ordering are different.

Results_Summary

In [167…
```python
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Alpha:Hyper Parameter", "Train AUC", "CV AU

x.add_row(["BOW", "Multinomial NB", 5, 0.72, 0.68, '0.70'])
```

```
x.add_row(["TF-IDF", "Multinomial NB", 0.5, 0.71, 0.66, 0.67])

print(x)
```

```
+------------+----------------+----------------------+-----------+--------+--------
--+
| Vectorizer |     Model      | Alpha:Hyper Parameter | Train AUC | CV AUC | Test AU
C |
+------------+----------------+----------------------+-----------+--------+--------
--+
|    BOW     | Multinomial NB |          5           |    0.72   |  0.68  |   0.70
|
|   TF-IDF   | Multinomial NB |         0.5          |    0.71   |  0.66  |   0.67
|
+------------+----------------+----------------------+-----------+--------+--------
--+
```

Better score is achieved by using BOW as compared to Tf-IDF vectorisers.