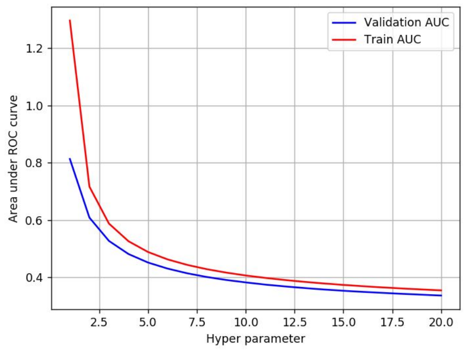
Assignment 6: Apply NB

1. Minimum data points need to be considered for people having 4GB RAM is **50k** and for 8GB RAM is **100k**
2. When you are using ramdomsearchcv or gridsearchcv you need not split the data into X_train,X_cv,X_test. As the above methods use kfold. The model will learn better if train data is more so splitting to X_train,X_test will suffice.
3. If you are writing for loops to tune your model then you need split the data into X_train,X_cv,X_test.
4. While splitting the data explore stratify parameter.
5. **Apply Multinomial NB on these feature sets**

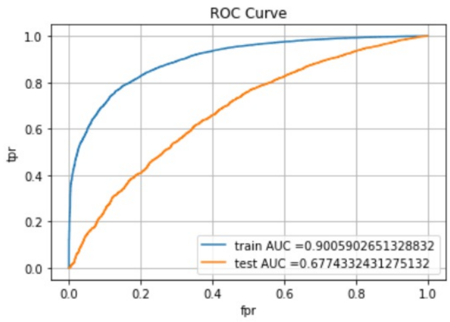
   - Features that need to be considered
     essay
     while encoding essay, try to experiment with the max_features and n_grams parameter of vectorizers and see if it increases AUC score.
     categorical features
     - teacher_prefix
     - project_grade_category
     - school_state
     - clean_categories
     - clean_subcategories
     numerical features
     - price
     - teacher_number_of_previously_posted_projects
     while encoding the numerical features check this and this

   - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
   - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
6. **The hyper paramter tuning(find best alpha:smoothing parameter)**

   - Consider alpha values in range: 10^-5 to 10^2 like [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]
   - Explore class_prior = [0.5, 0.5] parameter which can be present in MultinomialNB function(go through this) then check how results might change.
   - Find the best hyper parameter which will give the maximum AUC value
   - For hyper parameter tuning using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



     -while plotting take log(alpha) on your X-axis so that it will be more readable
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

-plot the confusion matrix in heatmaps, while plotting the confusion matrix go through the link

7. find the top 20 features from either from feature Set 1 or feature Set 2 using values of `feature_log_prob_ ` parameter of `MultinomialNB` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print **BOTH** positive as well as negative corresponding feature names.
   - go through the link
8. You need to summarize the results at the end of the notebook, summarize it in the table format

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

In [1]:

```
# importing necessary libraries
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

import pickle
from tqdm import tqdm,tnrange,tqdm_notebook
import os
import plotly as ply
import plotly.graph_objs as go
from plotly import offline
offline.init_notebook_mode()
from collections import Counter
```

2. Naive Bayes

Loading Data

In [2]:

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/dri
ve", force_remount=True).
```

In [3]:

```
dataPath = '/content/drive/MyDrive/8_Apply Naive Bayes on Donors Choose dataset/preprocessed_data.csv'
```

In [4]:

```
import pandas
data = pandas.read_csv(dataPath,nrows=100000)
```

In [5]:

```
data.shape
```

Out[5]:

```
(100000, 9)
```

In [6]:

```
data['project_is_approved'].value_counts()
```

Out[6]:

```
1    84817
0    15183
Name: project_is_approved, dtype: int64
```

Splitting data into Train and cross validation(or test): Stratified Sampling

In [7]:

```
# make data into as X and Y
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[7]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcat
0	ca	mrs	grades_prek_2	53	math_science	applieds health_life

In [8]:

```
# 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=0.33, stratify=y )
```

In [9]:

```
X_train.columns
```

Out[9]:

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

encoding categorical features (One hot encoding)

1. school_state
2. teacher_prefix
3. project_grade_category
4. clean_categories
5. clean_subcategories

In [10]:

```
# empty list to save all feature_names (for all features) order wise
# for getting top 20 features in both pos and neg from set-1:
l = []
```

In [11]:

```
# 1. school_state encoding
vectorizer = OneHotEncoder(sparse=False,handle_unknown='ignore')                # initialize One Hot En
coder.
vectorizer.fit(X_train['school_state'].values.reshape(-1,1))                    # fit has to happen onl
y on train data.

# use the vectorizer to convert string categories of school_state to numerical vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values.reshape(-1,1))
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.categories_)
print("="*100)
# saving features names of all features ( categorical/text/numerical)
l.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(67000, 51) (67000,)
(33000, 51) (33000,)
[array(['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga',
       'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me',
       'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
       'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx',
       'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy'], dtype=object)]
====================================================================================================
```

In [12]:

```
# 2. teacher_prefix
vectorizer = OneHotEncoder(sparse=False,handle_unknown='ignore')                # initialize One Hot En
coder.
vectorizer.fit(X_train['teacher_prefix'].values.reshape(-1,1))                   # fit has to happen on
ly on train data.

# use the vectorizer to convert categories of teacher_prefix to vector
X_train_teacher_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].values.reshape(-1,1))
X_test_teacher_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].values.reshape(-1,1))

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

```
# saving features names of all features ( categorical/text/numerical)
l.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(67000, 5) (67000,)
(33000, 5) (33000,)
[array(['dr', 'mr', 'mrs', 'ms', 'teacher'], dtype=object)]
====================================================================================================
```

In [13]:

```
# 3. project_grade_category
vectorizer = OneHotEncoder(sparse=False,handle_unknown='ignore')                # initialize One Hot En
coder.
vectorizer.fit(X_train['project_grade_category'].values.reshape(-1,1))                  # fit has to h
appen only on train data.

# use the vectorizer to convert categories of project_grade_category to vector
X_train_project_grade_category_ohe = vectorizer.transform(X_train['project_grade_category'].values.resh
ape(-1,1))
X_test_project_grade_category_ohe = vectorizer.transform(X_test['project_grade_category'].values.reshap
e(-1,1))

print("After vectorizations")
print(X_train_project_grade_category_ohe.shape, y_train.shape)
print(X_test_project_grade_category_ohe.shape, y_test.shape)
print(vectorizer.categories_)
print("="*100)
# saving features names of all features ( categorical/text/numerical)
l.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(67000, 4) (67000,)
(33000, 4) (33000,)
[array(['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2'],
      dtype=object)]
====================================================================================================
```

In [14]:

```
# 4. clean_categories
vectorizer = OneHotEncoder(sparse=False,handle_unknown='ignore')                # initialize One Hot En
coder.
vectorizer.fit(X_train['clean_categories'].values.reshape(-1,1))                  # fit has to happen
only on train data.

# use the vectorizer to convert categories of clean_categories to vector
X_train_clean_categories_ohe = vectorizer.transform(X_train['clean_categories'].values.reshape(-1,1))
X_test_clean_categories_ohe = vectorizer.transform(X_test['clean_categories'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_clean_categories_ohe.shape, y_train.shape)
print(X_test_clean_categories_ohe.shape, y_test.shape)
print("="*100)
# saving features names of all features ( categorical/text/numerical)
l.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(67000, 51) (67000,)
(33000, 51) (33000,)
====================================================================================================
```

In [15]:

```
# 5.clean_subcategories
vectorizer = OneHotEncoder(sparse=False,handle_unknown='ignore')                # initialize One Hot En
coder.
vectorizer.fit(X_train['clean_subcategories'].values.reshape(-1,1))                  # fit has to happ
en only on train data.
```

```
# use the vectorizer to convert categories of clean_subcategories to vector
X_train_clean_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].values.reshape(-1
,1))
X_test_clean_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].values.reshape(-1,1
))

print("After vectorizations")
print(X_train_clean_subcategories_ohe.shape, y_train.shape)
print(X_test_clean_subcategories_ohe.shape, y_test.shape)
print("="*100)
# saving features names of all features ( categorical/text/numerical)
l.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(67000, 387) (67000,)
(33000, 387) (33000,)
====================================================================================================
```

Encoding Numerical Features:

1. price
2. teacher_number_of_previously_posted_projects

In [16]:

```
# 6. price
# https://imgur.com/ldZA1zg
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(1,-1))
X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1)).reshape(-1,1)
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1)).reshape(-1,1)

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
# saving features names of all features ( categorical/text/numerical)
l.append("Price")
```

```
After vectorizations
(67000, 1) (67000,)
(33000, 1) (33000,)
====================================================================================================
```

In [17]:

```
# 7. teacher_number_of_previously_posted_projects
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_train_teacher_noOf_previous_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_
projects'].values.reshape(1,-1)).reshape(-1,1)
# X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1)).reshape(-1,1)
X_test_teacher_noOf_previous_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_pr
ojects'].values.reshape(1,-1)).reshape(-1,1)

print("After vectorizations")
print(X_train_teacher_noOf_previous_norm.shape, y_train.shape)
# print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_teacher_noOf_previous_norm.shape, y_test.shape)
print("="*100)
# saving features names of all features ( categorical/text/numerical)
l.append("teacher_number_of_previously_posted_projects")
```

```
After vectorizations
(67000, 1) (67000,)
(33000, 1) (33000,)
====================================================================================================
```

Encoding Text features:--> essay (BOW)

In [18]:

```
# 8.essay
vectorizer = CountVectorizer(ngram_range=(1,4),min_df=10,max_features=6000)
vectorizer.fit(X_train['essay'].values)

# using fitting countVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
# saving features names of all features ( categorical/text/numerical)
l.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(67000, 6000) (67000,)
(33000, 6000) (33000,)
====================================================================================================
```

Encoding Text feature --> essay (TFidf)

In [19]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
vectorizer.fit(X_train['essay'].values)                                          # fit has to happen
only on train data

# we use the fitted TfidfVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(67000, 5000) (67000,)
(33000, 5000) (33000,)
====================================================================================================
```

concatenate features : set-1 and set-2

In [20]:

```
# set-1
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_bow = hstack((X_train_state_ohe,X_train_teacher_prefix_ohe,X_train_project_grade_category_ohe,
               X_train_clean_categories_ohe,X_train_clean_subcategories_ohe,X_train_price_norm,X_train_
teacher_noOf_previous_norm,
               X_train_essay_bow)).tocsr()
X_te_bow = hstack((X_test_state_ohe,X_test_teacher_prefix_ohe,X_test_project_grade_category_ohe,
               X_test_clean_categories_ohe,X_test_clean_subcategories_ohe,X_test_price_norm,X_test_teac
her_noOf_previous_norm,
               X_test_essay_bow)).tocsr()
```

```
print("Final Data matrix")
print(X_tr_bow.shape, y_train.shape)
print(X_te_bow.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(67000, 6500) (67000,)
(33000, 6500) (33000,)
====================================================================================================
```

In [21]:

```
# length of list - 'l' containing features_names of all features comparing with set-1 dimensionality
print(len(l))
```

```
6500
```

In [22]:

```
# set-2
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_state_ohe,X_train_teacher_prefix_ohe,X_train_project_grade_category_ohe,
               X_train_clean_categories_ohe,X_train_clean_subcategories_ohe,X_train_price_norm,X_train_
teacher_noOf_previous_norm,
               X_train_essay_tfidf)).tocsr()
X_te_tfidf = hstack((X_test_state_ohe,X_test_teacher_prefix_ohe,X_test_project_grade_category_ohe,
               X_test_clean_categories_ohe,X_test_clean_subcategories_ohe,X_test_price_norm,X_test_teac
her_noOf_previous_norm,
               X_test_essay_tfidf)).tocsr()

print("Final Data matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(67000, 5500) (67000,)
(33000, 5500) (33000,)
====================================================================================================
```

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

Model-1 :

using set-1

In [23]:

```
alpha = [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]
parameters  = {'alpha': alpha}
```

In [24]:

```
# initializing MultinomialNB classifier
naive = MultinomialNB(class_prior=[0.5,0.5])
# using GridSearch with given parameters and "roc_auc" as a metric - 10 fold cross validation.
clf = GridSearchCV(naive,parameters,scoring='roc_auc',n_jobs=-1,cv=10,return_train_score=True)
```

In [25]:

```
# fit the train data of set-1
clf.fit(X_tr_bow,y_train)
results = pd.DataFrame.from_dict(clf.cv_results_)
results.head(2)
```

Out[25]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_te
0	0.152206	0.042386	0.015209	0.001654	1e-05	{'alpha': 1e-05}	0.691759	0.693484	
1	0.140035	0.012463	0.014635	0.000455	0.0005	{'alpha': 0.0005}	0.692013	0.694442	

In [26]:

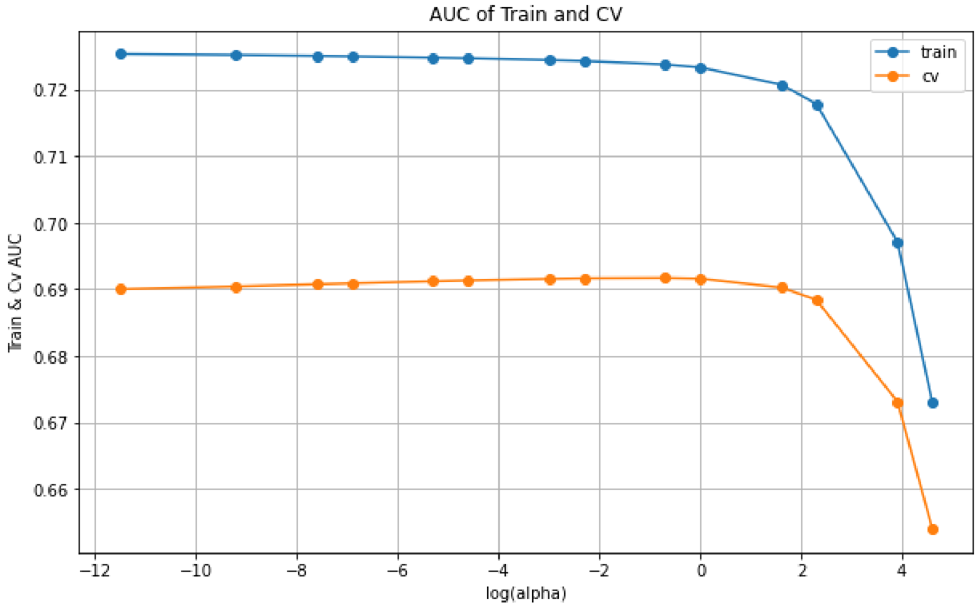
```
# sort results using alpha
results = results.sort_values(['param_alpha'])
```

In [27]:

```
# converting alpha in log scale for better understanding
alphas       = np.log(list(results['param_alpha']))
train_auc    = results['mean_train_score']
cv_auc       = results['mean_test_score']
# alphas =  results['param_alpha']
```

In [28]:

```
# finding best alpha through plots
plt.figure(figsize=(10,6))
plt.plot(alphas,train_auc,label='train',marker='o')
plt.plot(alphas,cv_auc,label='cv',marker='o')
plt.xlabel('log(alpha)')
plt.ylabel('Train & Cv AUC')
plt.title("AUC of Train and CV")
plt.legend()
plt.grid()
```



Observations:

1. alpha is a smoothing parameter, introduced to avoid zero multiplication problem and parameter which controls model complexity.
2. as alpha goes from 0.00001 to 1, cross validation AUC increases and after then its value start decreasing.
3. train AUC decreases slowly as alpha increases from 0.00001 to 1 and then decreases in higher rate.
4. alpha that gives best AUC for both Train and CV is `alpha=1`

In [29]:

```
# best alpha
alpha1 = 1
```

In [30]:

```
# trianing model using best alpha on set-1 (X_tr_bow (train),X_te_bow (test))
naive = MultinomialNB(alpha=alpha1,class_prior=[0.5,0.5])
naive.fit(X_tr_bow,y_train)
```

Out[30]:

```
MultinomialNB(alpha=1, class_prior=[0.5, 0.5], fit_prior=True)
```

In [31]:

```
# classes order
naive.classes_
```

Out[31]:

```
array([0, 1])
```

In [32]:

```
# predicted probability scores of train data
proba      = naive.predict_proba(X_tr_bow)
# proba contains both classes probabilities, here we need to pick class-1 proba scores
prob_train = proba[:,1]

# predicted probability scores of test data
proba      = naive.predict_proba(X_te_bow)
prob_test = proba[:,1]
```

In [33]:

```
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score
from sklearn.metrics import roc_auc_score
auc_train = roc_auc_score(y_train,prob_train)
auc_test = roc_auc_score(y_test,prob_test)
auc_test_model1 = auc_test
print(" Train auc = " + str(auc_train),'\n',"Test auc  = "+ str(auc_test))
```

```
 Train auc = 0.7205102998330201
 Test auc  = 0.6867406951014012
```

In [34]:

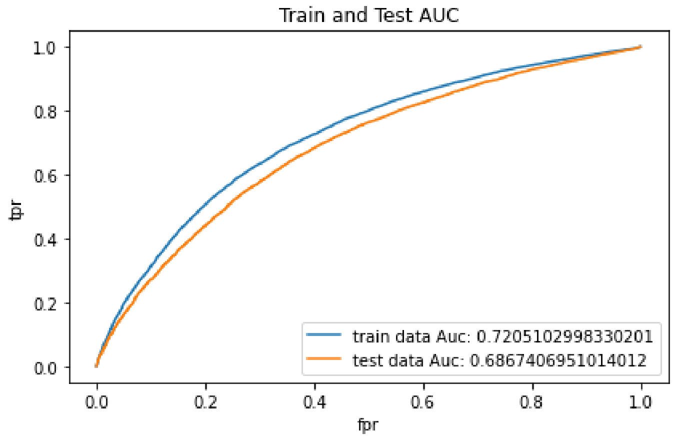
```
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn import metrics
fpr_tr, tpr_tr, thresholds_tr = metrics.roc_curve(y_train,prob_train, pos_label=1)
fpr_te, tpr_te, thresholds_te = metrics.roc_curve(y_test,prob_test, pos_label=1)
```

AUC

In [35]:

In [35]:

```
plt.plot(fpr_tr,tpr_tr,label="train data Auc: "+str(auc_train))
plt.plot(fpr_te,tpr_te,label="test data Auc: "+str(auc_test))
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("Train and Test AUC")
plt.legend()
plt.tight_layout()
plt.show()
```



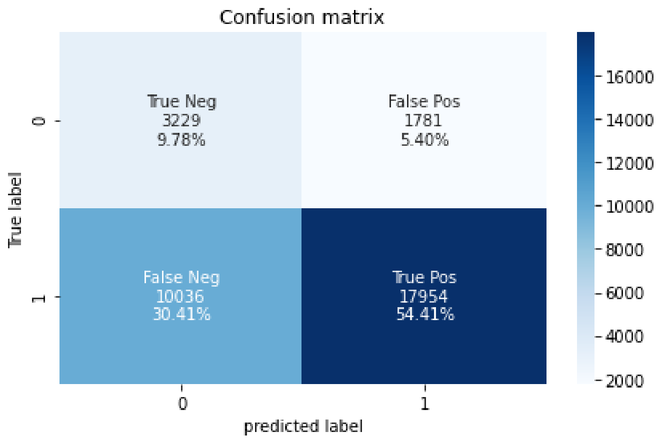
Confusion matrix

In [36]:

```
# predicted
y_predicted = naive.predict(X_te_bow)
mat = confusion_matrix(y_test,y_predicted)
```

In [37]:

```
#  https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                mat.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     mat.flatten()/np.sum(mat)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(mat, annot=labels, fmt='', cmap='Blues').set_title("Confusion matrix")
plt.ylabel("True label")
plt.xlabel("predicted label")
plt.tight_layout()
```



Top 20 Pos and neg features

In [38]:

```
# https://imgur.com/mWvE7gj
negClass = naive.feature_log_prob_[0]
posClass = naive.feature_log_prob_[1]

# getting neg and pos probability indices from highest to lowest
neg = np.argsort(negClass)[::-1]
pos = np.argsort(posClass)[::-1]

top20_neg = neg[:20]
top20_pos = pos[:20]


# getting top features from l using indeces:
neg_features = [l[i] for i in top20_neg]
pos_features = [l[i] for i in top20_pos]
```

In [39]:

```
print(neg_features)
print(pos_features)
```

```
['students', 'school', 'learning', 'my', 'classroom', 'not', 'learn', 'they', 'help', 'the', 'my studen
ts', 'nannan', 'many', 'we', 'need', 'work', 'come', 'love', 'materials', 'reading']
['students', 'school', 'my', 'learning', 'classroom', 'the', 'not', 'they', 'my students', 'learn', 'he
lp', 'many', 'nannan', 'we', 'reading', 'need', 'work', 'use', 'love', 'able']
```

Model-2 :

using set-2

In [40]:

```
alpha = [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]
parameters  = {'alpha': alpha}
```

In [41]:

```
# initializing MultinomialNB classifier
naive = MultinomialNB(class_prior=[0.5,0.5])
# using GridSearch with given parameters and "roc_auc" as a metric - 10 fold cross validation.
clf = GridSearchCV(naive,parameters,scoring='roc_auc',n_jobs=-1,cv=10,return_train_score=True)
```

In [42]:

```
clf.fit(X_tr_tfidf,y_train)
results = pd.DataFrame.from_dict(clf.cv_results_)
results.head(5)
```

Out[42]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_te
0	0.146920	0.006998	0.015211	0.001913	1e-05	{'alpha': 1e-05}	0.654994	0.651068	
1	0.145252	0.006250	0.014555	0.000483	0.0005	{'alpha': 0.0005}	0.654992	0.651068	
2	0.132707	0.004670	0.014287	0.000250	0.0001	{'alpha': 0.0001}	0.654995	0.651067	
3	0.129321	0.005374	0.014426	0.000314	0.005	{'alpha': 0.005}	0.654963	0.651056	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_te
4	0.131501	0.003552	0.014650	0.000500	0.001	{'alpha': 0.001}	0.654986	0.651067	

In [43]:

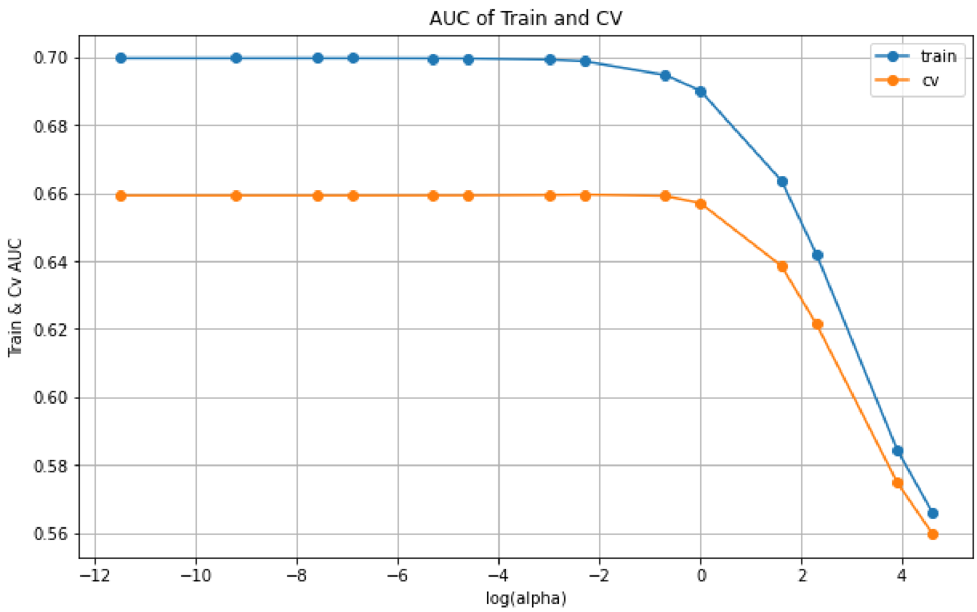
```
# sort results using alpha
results = results.sort_values(['param_alpha'])
```

In [44]:

```
alphas      = np.log(list(results['param_alpha']))
train_auc   = results['mean_train_score']
cv_auc      = results['mean_test_score']
```

In [45]:

```
# finding best alpha through plots
plt.figure(figsize=(10,6))
plt.plot(alphas,train_auc,label='train',marker='o')
plt.plot(alphas,cv_auc,label='cv',marker='o')
plt.xlabel('log(alpha)')
plt.ylabel('Train & Cv AUC')
plt.title("AUC of Train and CV")
plt.legend()
plt.grid()
```



Observations:

1. as alpha goes from 0.00001 to 0.1, cross validation AUC increases very slightly and after then its value start decreasing.
2. train AUC decreases slowly as alpha increases from 0.00001 to 0.1 and then decreases in higher rate.
3. alpha that gives best AUC for both Train and CV is `alpha=0.1`

In [46]:

```
# best alpha
alpha2 = 0.1
```

In [47]:

```
# trianing model using best alpha on set-2 (X_tr_tfidf (train),X_te_tfidf (test))
naive = MultinomialNB(alpha=alpha2,class_prior=[0.5,0.5])
naive.fit(X_tr_tfidf,y_train)
```

Out[47]:

```
MultinomialNB(alpha=0.1, class_prior=[0.5, 0.5], fit_prior=True)
```

In [48]:

```
# predicted probability scores of test data
proba      = naive.predict_proba(X_tr_tfidf)
# proba contains both classes probabilities, hence we need to pick class-1 proba scores
prob_train = proba[:,1]

# predicted probability scores of train data
proba      = naive.predict_proba(X_te_tfidf)
prob_test = proba[:,1]
```

In [49]:

```
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics
.roc_auc_score
from sklearn.metrics import roc_auc_score
auc_train = roc_auc_score(y_train,prob_train)
auc_test = roc_auc_score(y_test,prob_test)
auc_test_model2 = auc_test
print(" Train auc = " + str(auc_train),'\n',"Test auc = "+ str(auc_test))
```

```
 Train auc = 0.6960904774037342
 Test auc = 0.6478085058892575
```

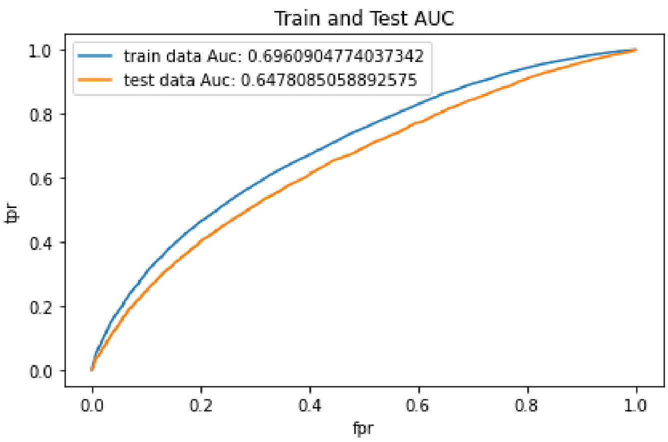
In [50]:

```
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn import metrics
fpr_tr, tpr_tr, thresholds_tr = metrics.roc_curve(y_train,prob_train, pos_label=1)
fpr_te, tpr_te, thresholds_te = metrics.roc_curve(y_test,prob_test, pos_label=1)
```

AUC

In [51]:

```
plt.plot(fpr_tr,tpr_tr,label="train data Auc: "+str(auc_train))
plt.plot(fpr_te,tpr_te,label="test data Auc: "+str(auc_test))
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("Train and Test AUC")
plt.legend()
plt.tight_layout()
plt.show()
```



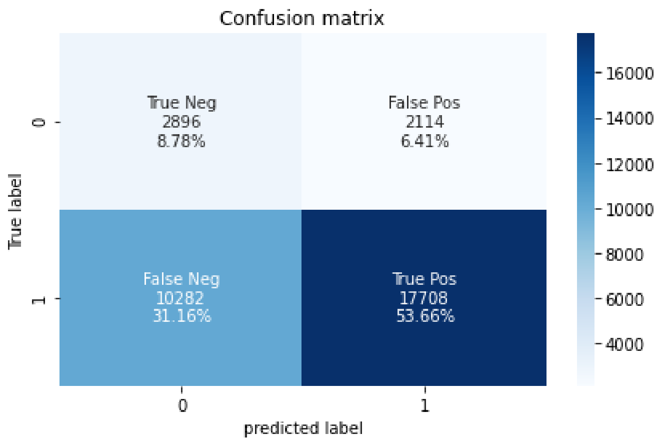
Confusion matrix

In [52]:

```
# predicted
y_predicted = naive.predict(X_te_tfidf)
mat = confusion_matrix(y_test,y_predicted)
```

In [53]:

```
#  https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                mat.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     mat.flatten()/np.sum(mat)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(mat, annot=labels, fmt='', cmap='Blues').set_title("Confusion matrix")
plt.ylabel("True label")
plt.xlabel("predicted label")
plt.tight_layout()
```



Summary

In [54]:

```
from tabulate import tabulate
table = [["BOW","Mutli_naive",alpha1,auc_test_model1],["TFIDF","Mutli_naive",alpha2,auc_test_model2]]
headers = ["Vectorizer","Model","alpha","Depth","AUC"]
print(tabulate(table,headers,tablefmt="grid"))
```

```
+--------------+-------------+---------+----------+
| Vectorizer   | Model       |   alpha |    Depth |
+==============+=============+=========+==========+
| BOW          | Mutli_naive |     1   | 0.686741 |
+--------------+-------------+---------+----------+
| TFIDF        | Mutli_naive |     0.1 | 0.647809 |
+--------------+-------------+---------+----------+
```

In [54]: