In [1]:
```
%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.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import pickle
from tqdm import tqdm
import os
from chart_studio import plotly # use chart_studio instead of plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

In [2]:
```
import pandas

data = pandas.read_csv('preprocessed_data.csv',nrows=50000)
```

In [3]:
```
data.head(1)
```

Out[3]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projects | project_is_approved | clean_categories | clea |
|---|---|---|---|---|---|---|---|
| 0 | ca | mrs | grades_prek_2 | 53 | 1 | math_science | l |

In [4]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[4]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projects | clean_categories | clean_subcategories | es |
|---|---|---|---|---|---|---|---|
| 0 | ca | mrs | grades_prek_2 | 53 | math_science | appliedsciences health_lifescience | fortu enc use stem |

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

In [5]:
```
# 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)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

## set 1: BoW

## 1.3 Make Data Model Ready: encoding eassay

In [93]:
```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
print("="*100)

# initialize the model with 10000 feature due to low configuration centre
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)

# fit the essay feature value
vectorizer.fit(X_train['essay'].values)

# divide the data train,test and cv
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)

# print the fitting train test and cv data
print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
(22445, 8) (22445,)
(11055, 8) (11055,)
(16500, 8) (16500,)
====================================================================================================
After vectorizations
(22445, 10000) (22445,)
(11055, 10000) (11055,)
(16500, 10000) (16500,)
====================================================================================================
```

## 1.4 Make Data Model Ready: encoding numerical, categorical features

### 1.4.1 encoding categorical features: School State

In [94]:
```
##  here i apply same procedure just like essay feature ##

vectorizer_SchoolState = CountVectorizer()
vectorizer_SchoolState.fit(X_train['school_state'].values)


X_train_state_ohe = vectorizer_SchoolState.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer_SchoolState.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer_SchoolState.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer_SchoolState.get_feature_names()) #print the feature names
print("="*100)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'm
a', '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']
====================================================================================================
```

### 1.4.2 encoding categorical features: teacher_prefix

In [95]:
```
##  here i apply same procedure just like essay feature ##

vectorizer_teacherPrefix = CountVectorizer()

vectorizer_teacherPrefix.fit(X_train['teacher_prefix'].values)

X_train_teacher_ohe = vectorizer_teacherPrefix.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer_teacherPrefix.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer_teacherPrefix.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer_teacherPrefix.get_feature_names())   #print the feature names
print("="*100)
```

```
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

### 1.4.3 encoding categorical features: project_grade_category

In [96]:
```
##  here i apply same procedure just like essay feature ##

vectorizer_PGC = CountVectorizer()

vectorizer_PGC.fit(X_train['project_grade_category'].values)

X_train_grade_ohe = vectorizer_PGC.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer_PGC.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer_PGC.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer_PGC.get_feature_names())    #print the feature names
print("="*100)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

### 1.4.4 encoding categorical features: clean_category

In [97]:
```
##  here i apply same procedure just like essay feature ##

vectorizer_CleanCategory = CountVectorizer()

vectorizer_CleanCategory.fit(X_train['clean_categories'].values)

X_train_clean_category = vectorizer_CleanCategory.transform(X_train['clean_categories'].values)
X_cv_clean_category = vectorizer_CleanCategory.transform(X_cv['clean_categories'].values)
X_test_clean_category = vectorizer_CleanCategory.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_category.shape, y_train.shape)
print(X_cv_clean_category.shape, y_cv.shape)
print(X_test_clean_category.shape, y_test.shape)
print(vectorizer_CleanCategory.get_feature_names()) #print the feature names
print("="*100)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_art
s', 'specialneeds', 'warmth']
====================================================================================================
```

### 1.4.5 encoding categorical features: clean_subcategorie(CS)

In [98]:
```
##  here i apply same procedure just like essay feature ##

vectorizer_cleanSC = CountVectorizer()

vectorizer_cleanSC.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data

X_train_cs = vectorizer_cleanSC.transform(X_train['clean_subcategories'].values)
X_cv_cs = vectorizer_cleanSC.transform(X_cv['clean_subcategories'].values)
X_test_cs = vectorizer_cleanSC.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_cs.shape, y_train.shape)
print(X_cv_cs.shape, y_cv.shape)
print(X_test_cs.shape, y_test.shape)
print(vectorizer_cleanSC.get_feature_names()) #print the feature names
print("="*100)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservic
e', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignl
anguages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writi
ng', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsciences',
'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

### 1.4.5 encoding numerical features: price

In [99]:
```
from sklearn.preprocessing import Normalizer
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))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

### 1.4.5 encoding numerical features: teacher_number_of_previously_posted_projects(TNPP)

In [100]:
```
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_TNPP_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_cv_TNPP_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_TNPP_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_TNPP_norm.shape, y_train.shape)
print(X_cv_TNPP_norm.shape, y_cv.shape)
print(X_test_TNPP_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

### conecatenate and create sparse matrix for set1

In [101]:
```
from scipy.sparse import hstack #import library for concatenate features.

##here we cancatenating all features of training, test and cross validation
X_tr_set1 = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe,
                    X_train_grade_ohe, X_train_clean_category,X_train_cs,X_train_price_norm,X_train_TNPP_norm)).tocsr
()
X_cr_set1 = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
                    X_cv_clean_category,X_cv_cs,X_cv_price_norm,X_cv_TNPP_norm )).tocsr()
X_te_set1 = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe,
                    X_test_grade_ohe, X_test_clean_category,X_test_cs,X_test_price_norm,X_test_TNPP_norm)).tocsr()

print("Final Data matrix for set 1: ")
print(X_tr_set1.shape, y_train.shape)
print(X_cr_set1.shape, y_cv.shape)
print(X_te_set1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix for set 1:
(22445, 10101) (22445,)
(11055, 10101) (11055,)
(16500, 10101) (16500,)
====================================================================================================
```

## set2: TFIDF

### encoding essay feature:

In [102]:
```
# Tfidf vectorizer transform text into feature vector.

## initialize model with different parameters
vectorizer_essay_set2 = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)

# fitting the data into model
X_train_essay_tfidf=vectorizer_essay_set2.fit_transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer_essay_set2.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer_essay_set2.transform(X_test['essay'].values)

print("After using tfidf vectorization: ")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
After using tfidf vectorization:
(22445, 10000) (22445,)
(11055, 10000) (11055,)
(16500, 10000) (16500,)
====================================================================================================
```

### conecatenate and create sparse matrix for set2

In [103]:
```
X_tr_set2 = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe,
                    X_train_grade_ohe, X_train_clean_category,X_train_cs,X_train_price_norm,X_train_TNPP_norm)).tocsr
()
X_cr_set2 = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe,
                    X_cv_grade_ohe, X_cv_clean_category,X_cv_cs,X_cv_price_norm,X_cv_TNPP_norm )).tocsr()
X_te_set2 = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe,
                    X_test_grade_ohe, X_test_clean_category,X_test_cs,X_test_price_norm,X_test_TNPP_norm)).tocsr()

print("Final Data matrix for set 2: ")
print(X_tr_set2.shape, y_train.shape)
print(X_cr_set2.shape, y_cv.shape)
print(X_te_set2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix for set 2:
(22445, 10101) (22445,)
(11055, 10101) (11055,)
(16500, 10101) (16500,)
====================================================================================================
```

1. **Apply Multinomial NB on these feature sets**

- Set 1: categorical, numerical features + preprocessed_eassay (BOW)
- Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)

## Applying Multinomial Naive Bayes on Set 1:

In [104]:
```
## import multinomial naive bayes classiffier
from sklearn.naive_bayes import MultinomialNB

## importing confusion matrix
from sklearn.metrics import confusion_matrix
```

In [105]:
```
# creating model
Mul_Naive_bayes = MultinomialNB()

## fitting the data into model
Mul_Naive_bayes.fit(X_tr_set1,y_train)

## here i am doing prediction
set1_prediction= Mul_Naive_bayes.predict(X_te_set1)
```

In [106]:
```
### find confusion matrix ###

confusion_matrix(set1_prediction, y_test)
```

Out[106]:
```
array([[ 1282,  3151],
       [ 1360, 10707]], dtype=int64)
```

### Applying Multinomial Naive Bayes on Set 2:

In [107]:
```
Mul_Naive_bayes.fit(X_tr_set2,y_train)

set2_prediction= Mul_Naive_bayes.predict(X_te_set2)
```

In [108]:
```
### find confusion matrix ###

confusion_matrix(set2_prediction, y_test)
```

Out[108]:
```
array([[   55,   162],
       [ 2587, 13696]], dtype=int64)
```

## 2. The hyper paramter tuning(find best alpha:smoothing parameter)

### 2.1 Find the best hyper parameter which will give the maximum AUC value

### 2.1.1 using BOW

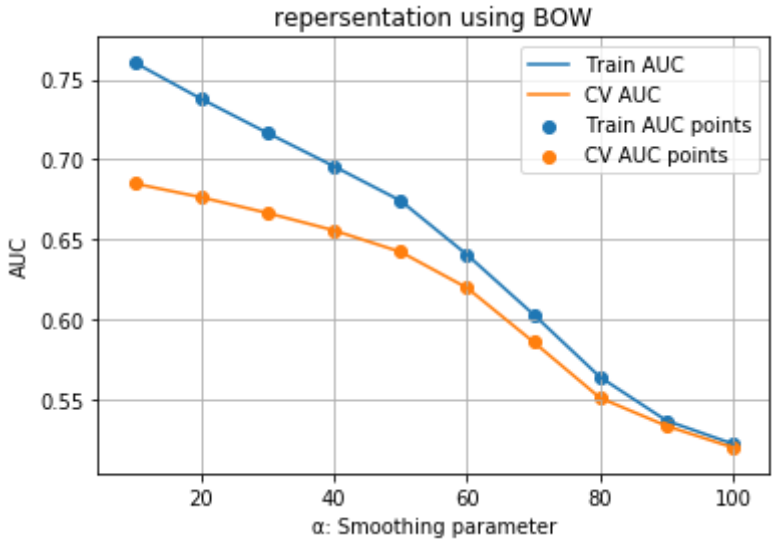
In [109]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [110]:

```
import matplotlib.pyplot as plt
#from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import MultinomialNB
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or non-thresholded measur
e of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
α = [10,20,30,40,50,60,70,80,90,100]  #values of alpha
for i in tqdm(α):
    mul_NB_model = MultinomialNB(alpha=i,class_prior=[0.5,0.5]) #checking ,model with all values of alpha
    mul_NB_model.fit(X_tr_set1, y_train)

    y_train_pred = batch_predict(mul_NB_model, X_tr_set1)
    y_cv_pred = batch_predict(mul_NB_model, X_cr_set1)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

plt.legend()
plt.xlabel("α: Smoothing parameter")
plt.ylabel("AUC")
plt.title("repersentation using BOW")
```

```
plt.grid()
plt.show()
```

```
100%|██████████| 10/10 [00:04<00:00,  2.44it/s]
```

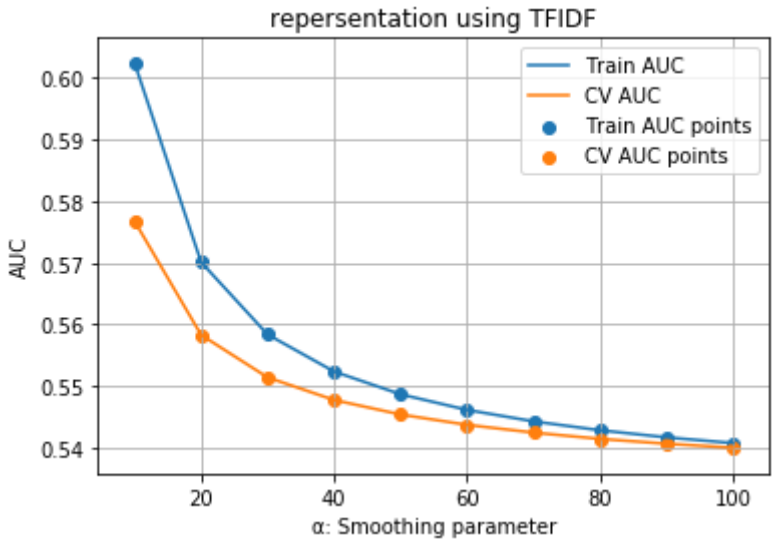



### 2.1.2 using TFIDF

In [111]:

```
train_auc = []
cv_auc = []
α = [10,20,30,40,50,60,70,80,90,100]
for i in tqdm(α):
    mul_NB_model = MultinomialNB(alpha=i,class_prior=[0.5,0.5])
    mul_NB_model.fit(X_tr_set2, y_train)

    y_train_pred = batch_predict(mul_NB_model, X_tr_set2)
    y_cv_pred = batch_predict(mul_NB_model, X_cr_set2)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

plt.legend()
plt.xlabel("α: Smoothing parameter")
plt.ylabel("AUC")
plt.title("repersentation using TFIDF")
plt.grid()
plt.show()
```

```
100%|██████████| 10/10 [00:02<00:00,  4.97it/
s]
```



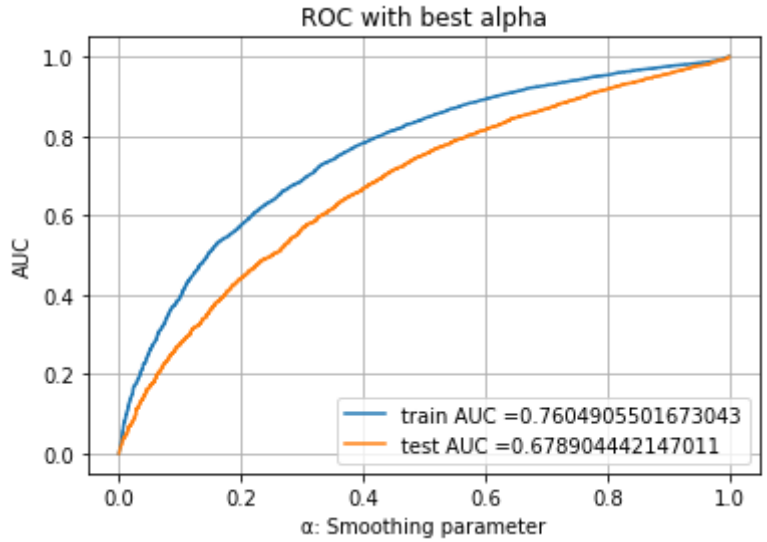

**observation: with the help of above curves the best smoothing parameter is α=10**

## plot roc curve with best smoothing parameter:

In [112]:
```
## function for calculating the ROC curve ###

def ROC_curve_with_best_alpha(y_train_pred,y_test_pred):

    train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
    test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("α: Smoothing parameter")
    plt.ylabel("AUC")
    plt.title("ROC with best alpha")
    plt.grid()
    plt.show()
```

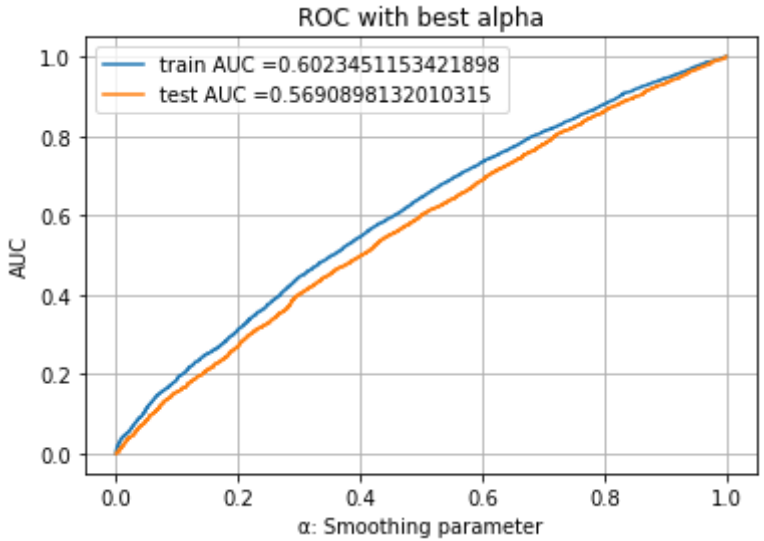
**using BOW:**

In [113]:
```
## here i am initializing the model and fit the best value of alpha ###
mul_NB_model = MultinomialNB(alpha=10,class_prior=[0.5,0.5])

## fit values in the model ##
mul_NB_model.fit(X_tr_set1, y_train)

## doing train prediction
y_train_pred = batch_predict(mul_NB_model, X_tr_set1)

## doing test prediction
y_test_pred = batch_predict(mul_NB_model, X_te_set1)

ROC_curve_with_best_alpha(y_train_pred,y_test_pred)
```




**using TFIDF:**

In [115]:
```
mul_NB_model = MultinomialNB(alpha=10,class_prior=[0.5,0.5])
mul_NB_model.fit(X_tr_set2, y_train)

y_train_pred = batch_predict(mul_NB_model, X_tr_set2)
y_test_pred = batch_predict(mul_NB_model, X_te_set2)
ROC_curve_with_best_alpha(y_train_pred,y_test_pred)
```




## Confusion matrix with best smoothing parameter:

### confusion matrix using BOW

In [116]:
```
## finding confusion matrix ###
confusion_matrix(y_test,mul_NB_model.predict(X_te_set1))
```

Out[116]:
```
array([[   29,  2613],
       [  106, 13752]], dtype=int64)
```

### confusion matrix using TFIDF:

In [117]:
```
## finding confusion matrix ###
confusion_matrix(y_test,mul_NB_model.predict(X_te_set2))
```

Out[117]:
```
array([[   14,  2628],
       [   68, 13790]], dtype=int64)
```

## 4. top 20 features from set2:

In [136]:
```
#  i have taken help from below link
#https://datascience.stackexchange.com/questions/65219/
#find-the-top-n-features-from-feature-set-using-absolute-values-of-feature-log-p

# initialize model with besr value of alpha
mul_NB_model = MultinomialNB(alpha=10,class_prior=[0.5,0.5])

# fit the data into model
mul_NB_model.fit(X_tr_set2, y_train)
```

Out[136]:
```
MultinomialNB(alpha=10, class_prior=[0.5, 0.5], fit_prior=True)
```

In [138]:
```
#we find feature indices sorted by log-probability of features

# argsort() will give the sorted feature

# For positive class
sorted_prob_class_positive_ind = mul_NB_model.feature_log_prob_[1, :].argsort()

# For negative class
sorted_prob_class_negative_ind = mul_NB_model.feature_log_prob_[0, :].argsort()
```

In [139]:
```
features_lst_tfidf = list(vectorizer_essay_set2.get_feature_names()+ vectorizer_SchoolState.get_feature_names()+
                          vectorizer_teacherPrefix.get_feature_names()+ vectorizer_PGC.get_feature_names()+
                         ['price']+vectorizer_CleanCategory.get_feature_names()+
                         ['teacher_number_of_previously_posted_projects']+vectorizer_cleanSC.get_feature_names())
```

In [141]:
```
Most_imp_features_positives = []
Most_imp_features_negatives = []

for index in sorted_prob_class_positive_ind[-20:-1]:
    Most_imp_features_positives.append(features_lst_tfidf[index])

for index in sorted_prob_class_negative_ind[-20:-1]:
    Most_imp_features_negatives.append(features_lst_tfidf[index])

print("20 most imp features for positive class:\n")
print(Most_imp_features_positives)

print("\n" + "-"*100)

print("\n20 most imp features for negative class:\n")
print(Most_imp_features_negatives)
```

```
20 most imp features for positive class:

['grades_9_12', 'price', 'students', 'performingarts', 'music_arts', 'gym_fitness', 'ca', 'grades_6_8', 'care_hunge
r', 'history_geography', 'literacy', 'health_wellness', 'grades_3_5', 'ms', 'literacy_language', 'grades_prek_2', 'hi
story_civics', 'mrs', 'warmth']

----------------------------------------------------------------------------------------------------

20 most imp features for negative class:

['students', 'warmth', 'music_arts', 'performingarts', 'price', 'gym_fitness', 'ca', 'grades_6_8', 'care_hunger', 'hi
story_geography', 'literacy', 'health_wellness', 'grades_3_5', 'ms', 'literacy_language', 'grades_prek_2', 'history_c
ivics', 'mrs', 'warmth']
```

## 5. Summarize result in table format:

In [150]:
```
#    i have taken help from below link:
#    https://stackoverflow.com/questions/39032720/formatting-lists-into-columns-of-a-table-output-python-3
#    importing libraries
import texttable as tt

# creating model
table = tt.Texttable()
headings = ['Vectorizer','Model','Hyper_parameter','AUC']
table.header(headings)
Vectorizer = ['BOW', 'TFIDF']
Model = ['MultinomialNB' , 'MultinomialNB']
Hyper_parameter = [10 , 10]
AUC = [0.678,0.569]

for row in zip(Vectorizer,Model,Hyper_parameter,AUC):
    table.add_row(row)

summarize_table = table.draw()
print (summarize_table)
```

```
+------------+---------------+-----------------+-------+
| Vectorizer |     Model     | Hyper_parameter |  AUC  |
+============+===============+=================+=======+
| BOW        | MultinomialNB | 10              | 0.678 |
+------------+---------------+-----------------+-------+
| TFIDF      | MultinomialNB | 10              | 0.569 |
+------------+---------------+-----------------+-------+
```

In [ ]: