```
In [1]:
import pandas as pd
import numpy as np
import cufflinks as cf
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected = True) # to get the connection
cf.go_offline()#offline mode
from sklearn.model_selection import train_test_split
from scipy.sparse import hstack,csr_matrix
import warnings
warnings.filterwarnings("ignore")
from scipy.sparse import hstack,vstack,csr_matrix
from sklearn.metrics import roc_auc_score,confusion_matrix,roc_curve, auc,accuracy_score
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, chi2
from sklearn import naive_bayes
from tqdm import tqdm
import seaborn as sns
%matplotlib inline
```

```
In [2]:
data=pd.read_csv('/home/sathish/Desktop/appliedai/Donors_choose_preprocessed_data.csv',na_filter=False)
```

```
In [3]:
y=data['project_is_approved'].values
x=data.drop(['project_is_approved'],axis=1)
```

1. Splitting Data

```
In [4]:
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,stratify=y)
x_train,x_val,y_train,y_val=train_test_split(x_train,y_train,test_size=0.33,stratify=y_train)
len_train=x_train.shape[0];len_val=x_val.shape[0];len_test=x_test.shape[0]
```

2. Creating Data Matrix

2.1 converting text to vectors

2.1.1 Bag of words

```
In [5]:
from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer()
bow_features=[]
x_train_essay_bow=vectorizer.fit_transform(x_train['essay'].values)
x_val_essay_bow=vectorizer.transform(x_val['essay'].values)
x_test_essay_bow=vectorizer.transform(x_test['essay'].values)

bow_features.extend(vectorizer.get_feature_names())

x_train_summary_bow=vectorizer.fit_transform(x_train['project_resource_summary'].values)
x_val_summary_bow=vectorizer.transform(x_val['project_resource_summary'].values)
x_test_summary_bow=vectorizer.transform(x_test['project_resource_summary'].values)

bow_features.extend(vectorizer.get_feature_names())


x_train_title_bow=vectorizer.fit_transform(x_train['project_title'].values)
x_val_title_bow=vectorizer.transform(x_val['project_title'].values)
x_test_title_bow=vectorizer.transform(x_test['project_title'].values)

bow_features.extend(vectorizer.get_feature_names())
```

```
In [6]:
len(bow_features)
```

```
Out[6]:
68536
```

2.1.2 TFIDF

```
In [7]:
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer=TfidfVectorizer()

tfidf_features=[]
x_train_essay_tfidf=vectorizer.fit_transform(x_train['essay'].values)
x_val_essay_tfidf=vectorizer.transform(x_val['essay'].values)
x_test_essay_tfidf=vectorizer.transform(x_test['essay'].values)

tfidf_features.extend(vectorizer.get_feature_names())

x_train_summary_tfidf=vectorizer.fit_transform(x_train['project_resource_summary'].values)
x_val_summary_tfidf=vectorizer.transform(x_val['project_resource_summary'].values)
x_test_summary_tfidf=vectorizer.transform(x_test['project_resource_summary'].values)

tfidf_features.extend(vectorizer.get_feature_names())


x_train_title_tfidf=vectorizer.fit_transform(x_train['project_title'].values)
x_val_title_tfidf=vectorizer.transform(x_val['project_title'].values)
x_test_title_tfidf=vectorizer.transform(x_test['project_title'].values)
tfidf_features.extend(vectorizer.get_feature_names())
```

2.2 one hot encoding the categorical features

```
In [8]:
cat_features=[]
vectorizer=CountVectorizer()
x_train_state_onehot=vectorizer.fit_transform(x_train['school_state'].values)
x_val_state_onehot=vectorizer.transform(x_val['school_state'].values)
x_test_state_onehot=vectorizer.transform(x_test['school_state'].values)

cat_features.extend(vectorizer.get_feature_names())

x_train_tprefix_onehot=vectorizer.fit_transform(x_train['teacher_prefix'].values)
x_val_tprefix_onehot=vectorizer.transform(x_val['teacher_prefix'].values)
x_test_tprefix_onehot=vectorizer.transform(x_test['teacher_prefix'].values)
cat_features.extend(vectorizer.get_feature_names())


x_train_grade_onehot=vectorizer.fit_transform(x_train['project_grade_category'].values)
x_val_grade_onehot=vectorizer.transform(x_val['project_grade_category'].values)
x_test_grade_onehot=vectorizer.transform(x_test['project_grade_category'].values)
cat_features.extend(vectorizer.get_feature_names())

x_train_cat_onehot=vectorizer.fit_transform(x_train['project_subject_categories'].values)
x_val_cat_onehot=vectorizer.transform(x_val['project_subject_categories'].values)
x_test_cat_onehot=vectorizer.transform(x_test['project_subject_categories'].values)
cat_features.extend(vectorizer.get_feature_names())

x_train_subcat_onehot=vectorizer.fit_transform(x_train['project_subject_subcategories'].values)
x_val_subcat_onehot=vectorizer.transform(x_val['project_subject_subcategories'].values)
x_test_subcat_onehot=vectorizer.transform(x_test['project_subject_subcategories'].values)
cat_features.extend(vectorizer.get_feature_names())
```

2.3 Normalizing the numerical features

```
In [9]:
from sklearn.preprocessing import Normalizer
import scipy
normalizer=Normalizer()
x_train_price_norm=normalizer.fit_transform(x_train['price'].values.reshape(-1,1))
x_val_price_norm=normalizer.transform(x_val['price'].values.reshape(-1,1))
x_test_price_norm=normalizer.transform(x_test['price'].values.reshape(-1,1))

x_train_quantity_norm=normalizer.fit_transform(x_train['quantity'].values.reshape(-1,1))
x_val_quantity_norm=normalizer.transform(x_val['quantity'].values.reshape(-1,1))
x_test_quantity_norm=normalizer.transform(x_test['quantity'].values.reshape(-1,1))
```

Combining all the feature names

```
In [10]:
bow_features.extend(cat_features)
bow_features.extend(['price','quantity'])

tfidf_features.extend(cat_features)
tfidf_features.extend(['price','quantity'])
```

2.4 Building data matrix

2.4.1 Set 1: Categorical, numerical, essay(BOW), summary(BOW), title(BOW)

```
In [11]:
#text features
bow_train=hstack((x_train_essay_bow,x_train_summary_bow,x_train_title_bow))
bow_valid=hstack((x_val_essay_bow,x_val_summary_bow,x_val_title_bow))
bow_test=hstack((x_test_essay_bow,x_test_summary_bow,x_test_title_bow))

#all categrocial features
x_train_cat=hstack((x_train_state_onehot,x_train_tprefix_onehot,x_train_grade_onehot,x_train_cat_onehot,x_train_subcat_onehot))
x_val_cat=hstack((x_val_state_onehot,x_val_tprefix_onehot,x_val_grade_onehot,x_val_cat_onehot,x_val_subcat_onehot))
x_test_cat=hstack((x_test_state_onehot,x_test_tprefix_onehot,x_test_grade_onehot,x_test_cat_onehot,x_test_subcat_onehot))

#numerical features
x_train_num=hstack((csr_matrix(x_train_price_norm),x_train_quantity_norm))
x_val_num=hstack((csr_matrix(x_val_price_norm),x_val_quantity_norm))
x_test_num=hstack((csr_matrix(x_test_price_norm),x_test_quantity_norm))

#final matrices
final_bow_train=hstack((bow_train,x_train_cat,x_train_num)).tocsr()
final_bow_val=hstack((bow_valid,x_val_cat,x_val_num)).tocsr()
final_bow_test=hstack((bow_test,x_test_cat,x_test_num)).tocsr()
```

2.4.2 Set 2:Categorical, numerical, essay(TFIDF), summary(TFIDF), title(TFIDF)

```
In [12]:
#text features
tfidf_train=hstack((x_train_essay_tfidf,x_train_summary_tfidf,x_train_title_tfidf))
tfidf_valid=hstack((x_val_essay_tfidf,x_val_summary_tfidf,x_val_title_tfidf))
tfidf_test=hstack((x_test_essay_tfidf,x_test_summary_tfidf,x_test_title_tfidf))

#all categrocial features
x_train_cat=hstack((x_train_state_onehot,x_train_tprefix_onehot,x_train_grade_onehot,x_train_cat_onehot,x_train_subcat_onehot))
x_val_cat=hstack((x_val_state_onehot,x_val_tprefix_onehot,x_val_grade_onehot,x_val_cat_onehot,x_val_subcat_onehot))
x_test_cat=hstack((x_test_state_onehot,x_test_tprefix_onehot,x_test_grade_onehot,x_test_cat_onehot,x_test_subcat_onehot))

#numerical features
x_train_num=hstack((csr_matrix(x_train_price_norm),x_train_quantity_norm))
x_val_num=hstack((csr_matrix(x_val_price_norm),x_val_quantity_norm))
x_test_num=hstack((csr_matrix(x_test_price_norm),x_test_quantity_norm))

#final matrices
final_tfidf_train=hstack((tfidf_train,x_train_cat,x_train_num)).tocsr()
final_tfidf_val=hstack((tfidf_valid,x_val_cat,x_val_num)).tocsr()
final_tfidf_test=hstack((tfidf_test,x_test_cat,x_test_num)).tocsr()
```

```
In [13]:
#making sure that evertything is ok
print(final_bow_train.shape,final_bow_val.shape,final_bow_test.shape)
print(final_tfidf_train.shape,final_tfidf_val.shape,final_tfidf_test.shape)
```

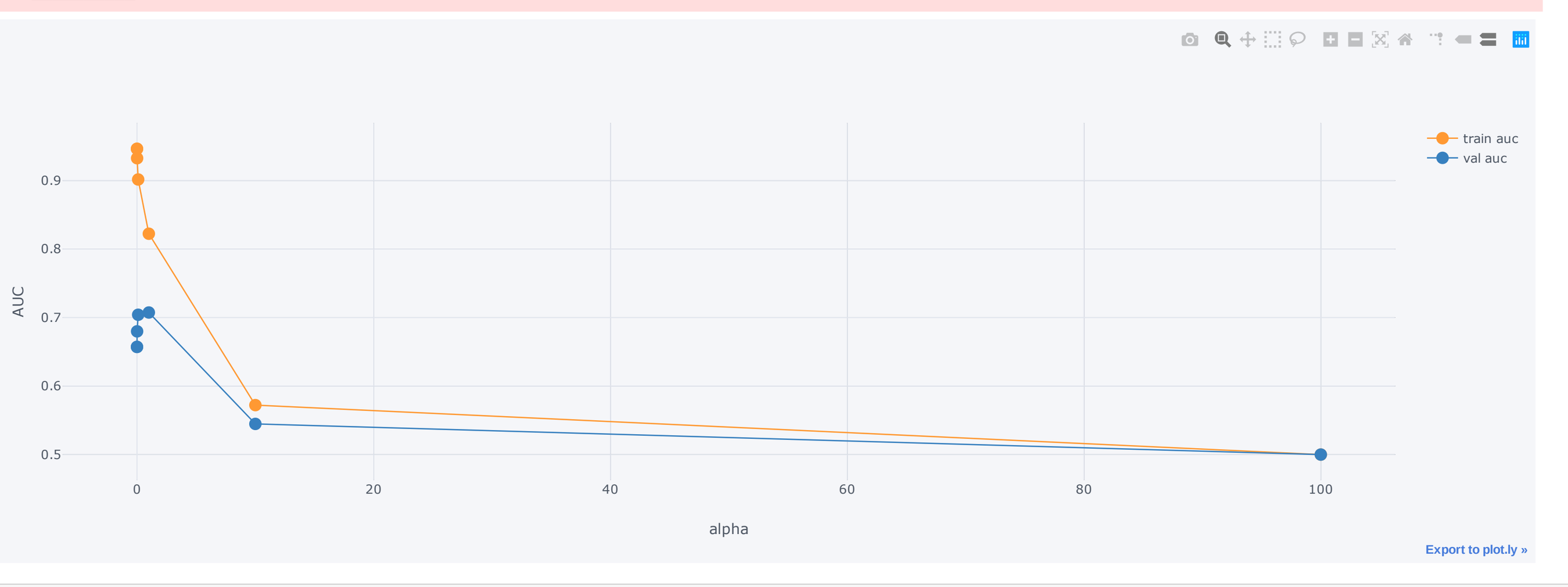
```
(49041, 68664) (24155, 68664) (36052, 68664)
(49041, 68664) (24155, 68664) (36052, 68664)
```

2.5 Naive Bayes

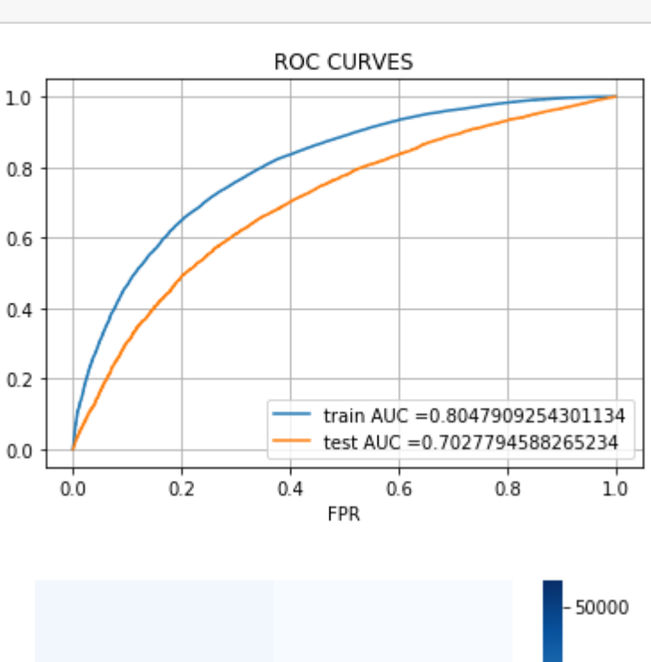
2.5.1 Applying Naive Bayes -Set 1

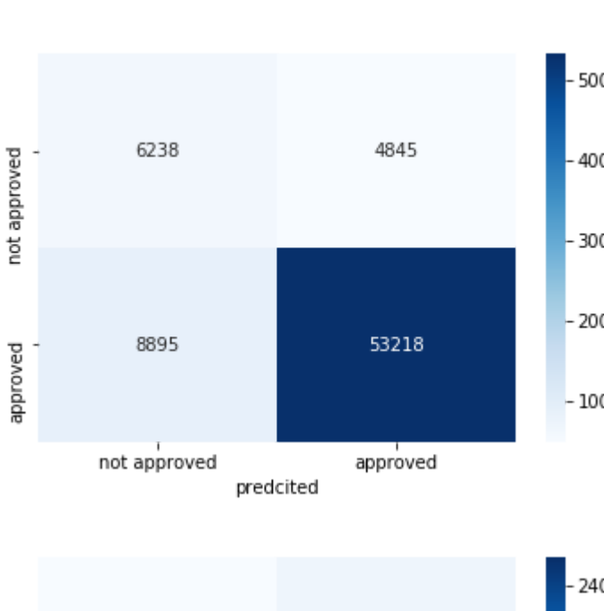
```
In [14]:
#alpha is smoothing parameter

#considering alpha value between 10**-3 and 10**-3

alpha=[0.001,0.001,0.01,0.1,1,10,100]
train_auc=[]
val_auc=[]
for k in tqdm(alpha):
    bayes=naive_bayes.MultinomialNB(alpha=k,class_prior=[0.5,0.5])
    bayes.fit(final_bow_train,y_train)
    train_pred = bayes.predict_proba(final_bow_train)[:,1]
    val_pred=bayes.predict_proba(final_bow_val)[:,1]
    train_auc.append(roc_auc_score(y_train,train_pred))
    val_auc.append(roc_auc_score(y_val,val_pred))

df=pd.DataFrame({'train auc':train_auc,'val auc':val_auc},index=alpha)

df.iplot(kind='scatter',xTitle='alpha',yTitle='AUC',mode='markers+lines')
```

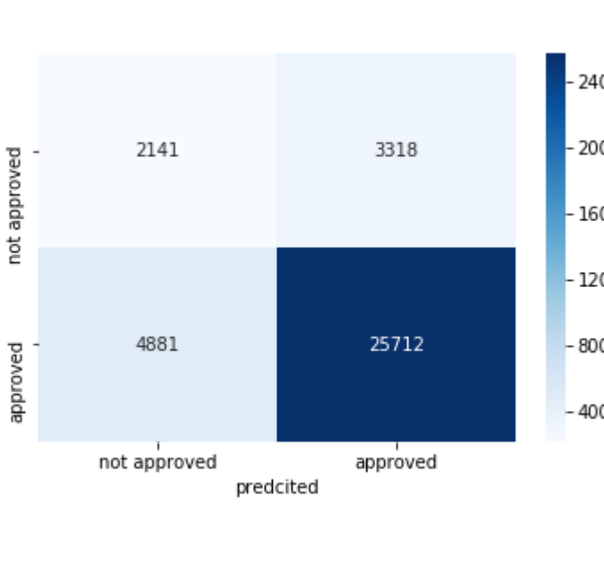
```
100%|██████████| 7/7 [00:01<00:00,  4.12it/s]
```




```
In [15]:
# we will choose α in such way that CV AUC is high.

# From the graph we can observe that we are going to underfit the model as we increase α value.

# So we will choose alpha value as 1

#explored by setting class priors also.But we are able better distinguish without any explicit priors.



bow_alpha_best=1

train_set=vstack((final_bow_train,final_bow_val))
test_set=final_bow_test
labels_train=np.concatenate((y_train,y_val))
labels_test=y_test


bayes=naive_bayes.MultinomialNB(alpha=bow_alpha_best)
bayes.fit(train_set,labels_train)
y_train_pred = bayes.predict_proba(train_set)[:,1]
y_test_pred = bayes.predict_proba(test_set)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(labels_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(labels_test, y_test_pred)
bow_auc=auc(test_fpr,test_tpr)

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(bow_auc))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC CURVES")
plt.grid()
plt.show()

train_predict=bayes.predict(train_set)
cm=confusion_matrix(labels_train,train_predict)
df=pd.DataFrame(cm,['not approved','approved'],['not approved','approved'])
plt.xlabel('predcited')
plt.ylabel('actual')
sns.heatmap(df,annot=True,cmap='Blues',fmt='g')
plt.xlabel('predcited')
plt.ylabel('actual')
plt.show()

test_predict=bayes.predict(test_set)
cm=confusion_matrix(labels_test,test_predict)
df=pd.DataFrame(cm,['not approved','approved'],['not approved','approved'])
plt.xlabel('predcited')
plt.ylabel('actual')
sns.heatmap(df,annot=True,cmap='Blues',fmt='g')
plt.xlabel('predcited')
plt.ylabel('actual')
plt.show()
```







2.5.1.1 Top 10 important features of positive class from SET 1

```
In [16]:
#https://stackoverflow.com/q/54988116

tfidf_features=np.array(tfidf_features)
bow_features=np.array(bow_features)
sorted_idx = np.argsort( bayes.feature_log_prob_[1] )[-10:]#gives top 10 indices which have highest likelihoodsfor  positive negative class

for p,q in zip(tfidf_features[ sorted_idx ], bayes.feature_log_prob_[1][ sorted_idx ]):
    print('{:30}:{}'.format(p,q))
```

```
help                          :-5.059294566015522
learn                         :-5.034198753860997
not                           :-4.990202885655069
they                          :-4.987025108629698
the                           :-4.955012469059964
classroom                     :-4.713791144093577
learning                      :-4.686351975684781
my                            :-4.6430613011488315
school                        :-4.329144968482373
students                      :-3.1858830722062894
```

2.5.1.2 Top 10 important features of negative class from SET 1

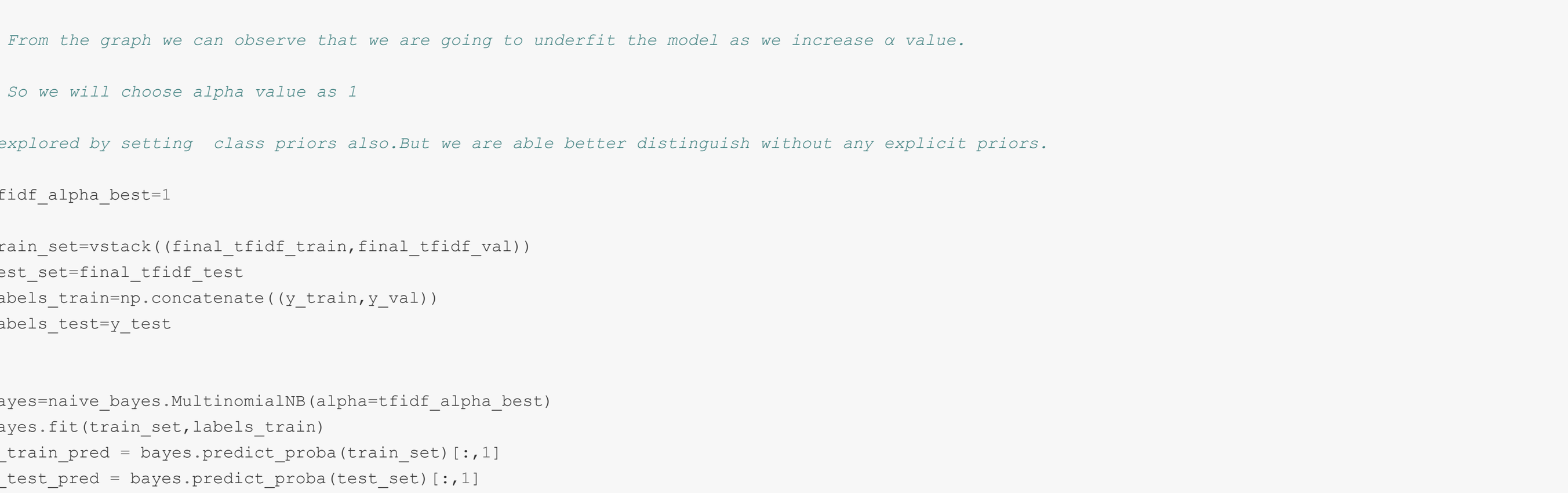
```
In [17]:
sorted_idx = np.argsort( bayes.feature_log_prob_[0] )[-10:]#gives top 10 indices which have highest likelihoods for negative class
for p,q in zip(tfidf_features[ sorted_idx ], bayes.feature_log_prob_[0][ sorted_idx ]):
    print('{:30}:{}'.format(p,q))
```

```
students                      :-5.051072127943659
help                          :-5.038560361222478
they                          :-5.021697091747162
learn                         :-5.006433204931639
not                           :-4.978851589818225
classroom                     :-4.812565112177973
my                            :-4.694007364778654
learning                      :-4.65291713296226
school                        :-4.330642333830269
students                      :-3.2395939176230595
```

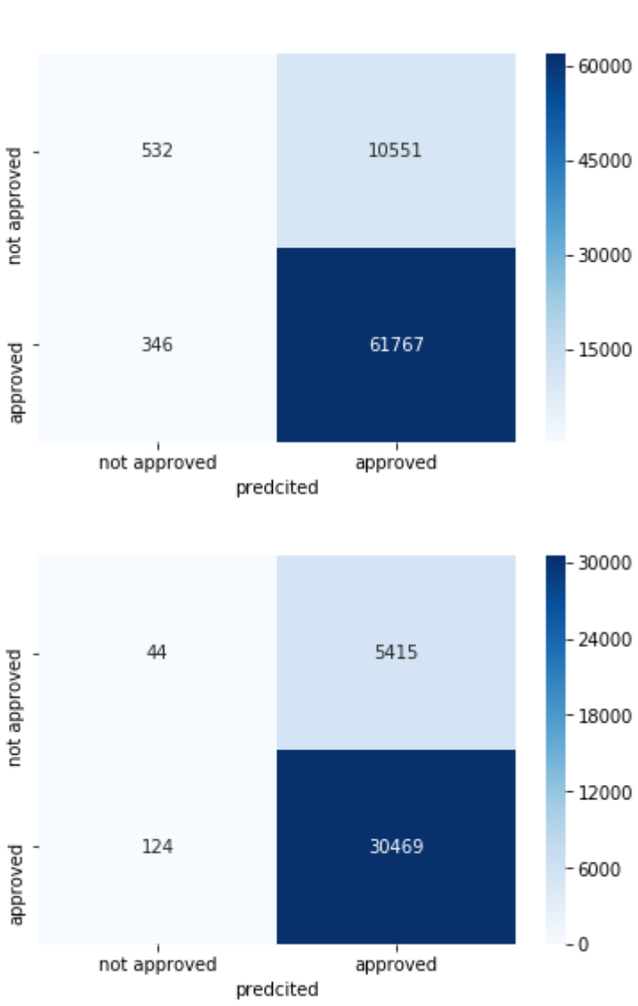
2.5.2 Applying KNN-set 2

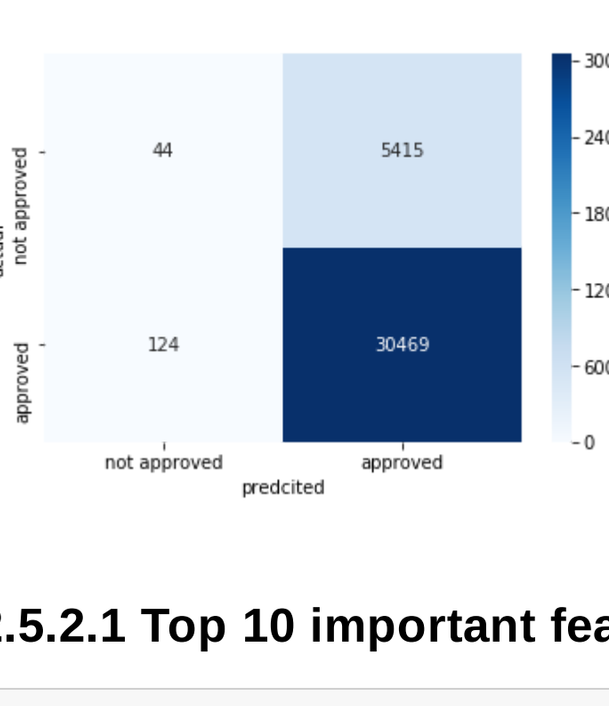
```
In [18]:
#alpha is smoothing parameter

#considering alpha value between 10**-3 and 10**-3



alpha=[0.001,0.001,0.01,0.1,1,10,100]
train_auc=[]
val_auc=[]
for k in tqdm(alpha):
    bayes=naive_bayes.MultinomialNB(alpha=k,class_prior=[0.5,0.5])
    bayes.fit(final_bow_train,y_train)
    train_pred = bayes.predict_proba(final_tfidf_train)[:,1]
    val_pred=bayes.predict_proba(final_tfidf_val)[:,1]
    train_auc.append(roc_auc_score(y_train,train_pred))
    val_auc.append(roc_auc_score(y_val,val_pred))

df=pd.DataFrame({'train auc':train_auc,'val auc':val_auc},index=alpha)

df.iplot(kind='scatter',xTitle='alpha',yTitle='AUC',mode='markers+lines')
```

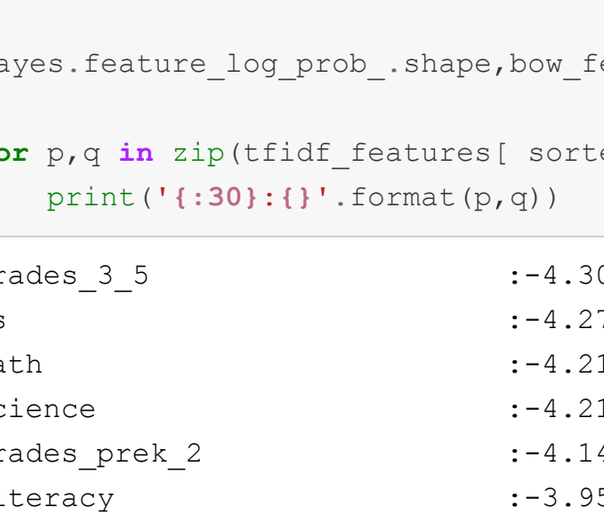
```
100%|██████████| 7/7 [00:01<00:00,  4.20it/s]
```




```
In [19]:
# we will choose α in such way that CV AUC is high .

# From the graph we can observe that we are going to underfit the model as we increase α value.

# So we will choose alpha value as 1

#explored by setting  class priors also.But we are able better distinguish without any explicit priors.

tfidf_alpha_best=1

train_set=vstack((final_tfidf_train,final_tfidf_val))
test_set=final_tfidf_test
labels_train=np.concatenate((y_train,y_val))
labels_test=y_test


bayes=naive_bayes.MultinomialNB(alpha=tfidf_alpha_best)
bayes.fit(train_set,labels_train)
y_train_pred = bayes.predict_proba(train_set)[:,1]
y_test_pred = bayes.predict_proba(test_set)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(labels_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(labels_test, y_test_pred)
tfidf_auc=auc(test_fpr,test_tpr)

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(tfidf_auc))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC CURVES")
plt.grid()
plt.show()

train_predict=bayes.predict(train_set)
cm=confusion_matrix(labels_train,train_predict)
df=pd.DataFrame(cm,['not approved','approved'],['not approved','approved'])
plt.xlabel('predcited')
plt.ylabel('actual')
sns.heatmap(df,annot=True,cmap='Blues',fmt='g')
plt.xlabel('predcited')
plt.ylabel('actual')
plt.show()

test_predict=bayes.predict(test_set)
cm=confusion_matrix(labels_test,test_predict)
df=pd.DataFrame(cm,['not approved','approved'],['not approved','approved'])
plt.xlabel('predcited')
plt.ylabel('actual')
sns.heatmap(df,annot=True,cmap='Blues',fmt='g')
plt.xlabel('predcited')
plt.ylabel('actual')
plt.show()
```







2.5.2.1 Top 10 important features of positive class from SET 2

```
In [20]:
sorted_idx = np.argsort( bayes.feature_log_prob_[1] )[-10:]#gives top 10 indices which have highest likelihoods for positive class

bayes.feature_log_prob_.shape,bow_features.shape

for p,q in zip(tfidf_features[ sorted_idx ], bayes.feature_log_prob_[1][ sorted_idx ]):
    print('{:30}:{}'.format(p,q))
```

```
grades_3_5                    :-4.305987418245687
ms                            :-4.2730371806442164
math                          :-4.218917396076968
science                       :-4.218917396076968
grades_prek_2                 :-4.140732252549185
literacy                      :-3.9533131472044207
language                      :-3.9533131472044207
mrs                           :-3.87373471859377
price                         :-3.2363801456792967
quantity                      :-3.2363801456792967
```

2.5.2.2 Top 10 important features of negative class from SET 2

```
In [21]:
sorted_idx = np.argsort( bayes.feature_log_prob_[0] )[-10:]#gives top 10 indices which have highest likelihoods for negative class
for p,q in zip(tfidf_features[ sorted_idx ], bayes.feature_log_prob_[0][ sorted_idx ]):
    print('{:30}:{}'.format(p,q))
```

```
grades_3_5                    :-4.525566257966526
ms                            :-4.4101578695124175
science                       :-4.335022801078235
math                          :-4.335022801078235
grades_prek_2                 :-4.323105852854402
language                      :-4.279373479805674
literacy                      :-4.279373479805674
mrs                           :-4.108672454034739
price                         :-3.4158860889216083
quantity                      :-3.4158860889216083
```

2.6 Conclusions

```
In [22]:
from prettytable import PrettyTable
import prettytable
x=PrettyTable()
x.hrules=prettytable.ALL
x.left_padding_width=3
x.right_padding_width=3
x.field_names=['Vectorizer','model','hyperparameter','AUC']
x.add_row(['BOW','Multinomial Naive Bayes','α-'+str(bow_alpha_best),bow_auc])
x.add_row(['TFIDF','Multinomial Naive Bayes','α-'+str(tfidf_alpha_best),tfidf_auc])

print(x)
```

Vectorizer	model	hyperparameter	AUC
BOW	Multinomial Naive Bayes	α-1	0.7027794588265234
TFIDF	Multinomial Naive Bayes	α-1	0.6440708836081408