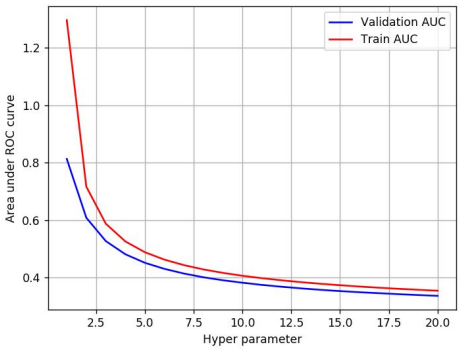
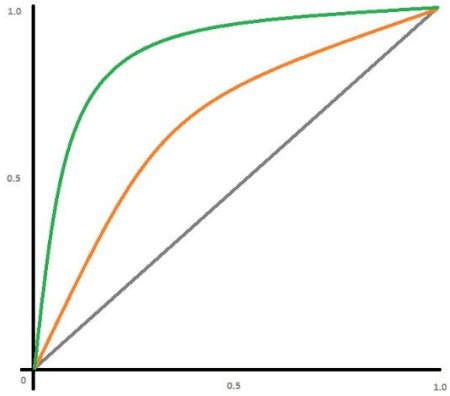
Assignment 6: Apply NB

1. **Apply Multinomial NB on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_essay (BOW)
    - Set 2: categorical, numerical features + preprocessed_essay (TFIDF)
2. **The hyper paramter tuning(find best alpha:smoothing parameter)**
    - Find the best hyper parameter which will give the maximum AUC value
    - find the best hyper paramter using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
    -
3. **Representation of results**
    - 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



    - 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 = ??

4. fine the top 20 features from either from feature Set 1 or feature Set 2 using absolute values of `feature_log_prob_ ` parameter of `MultinomialNB` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print their corresponding feature names
5. 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

2. Naive Bayes

1.1 Loading Data

In [28]:

```
import pandas
data = pandas.read_csv('preprocessed_data.csv')
```

In [29]:

```
data.head(3)
```

Out[29]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_appro
0	ca	mrs	grades_prek_2	53	1
1	ut	ms	grades_3_5	4	1
2	ca	mrs	grades_prek_2	10	1

In [30]:

```
data.describe()
```

Out[30]:

	teacher_number_of_previously_posted_projects	project_is_approved	price
count	109248.000000	109248.000000	109248.000000
mean	11.153165	0.848583	298.119343
std	27.777154	0.358456	367.498030
min	0.000000	0.000000	0.660000
25%	0.000000	1.000000	104.310000
50%	2.000000	1.000000	206.220000
75%	9.000000	1.000000	379.000000
max	451.000000	1.000000	9999.000000

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

In [31]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
from sklearn.model_selection import train_test_split

y = data['project_is_approved']
X = data.drop('project_is_approved',axis=1)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.25, stratify=y_train)
print(X_train.shape,X_cv.shape,X_test.shape,y_train.shape,y_cv.shape,y_test.shape)
```

```
(65548, 8) (21850, 8) (21850, 8) (65548,) (21850,) (21850,)
```

1.3 Make Data Model Ready: encoding essay, and project_title

In [32]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
from sklearn.feature_extraction.text import CountVectorizer

vectorizer_bow = CountVectorizer(ngram_range=(1,4),max_df=10,max_features=10000)
vectorizer_bow.fit(X_train['essay'].values)

X_train_essay_bow = vectorizer_bow.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer_bow.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer_bow.transform(X_test['essay'].values)

print('After Vectorization')
print('='*50)
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('='*50)
print(vectorizer_bow.get_feature_names()[:10])
```

```
After Vectorization
==================================================
(65548, 10000) (65548,)
(21850, 10000) (21850,)
(21850, 10000) (21850,)
==================================================
['10 class', '10 tablets', '10 weeks', '100 english', '100 mile', '10th grade english language', '
113', '12 new', '12 self', '12 self contained']
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [33]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do
```

```
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

# encoding categorical features

# encoding school_state
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values)

X_train_state = vectorizer.transform(X_train['school_state'].values)
X_test_state = vectorizer.transform(X_test['school_state'].values)
X_cv_state = vectorizer.transform(X_cv['school_state'].values)

print(vectorizer.get_feature_names()[:5])

# encoding teacher_prefix
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values)

X_train_teacher_prefix = vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher_prefix = vectorizer.transform(X_test['teacher_prefix'].values)
X_cv_teacher_prefix = vectorizer.transform(X_cv['teacher_prefix'].values)

print(vectorizer.get_feature_names()[:5])

# encoding project_grade_category
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values)

X_train_project_grade_category = vectorizer.transform(X_train['project_grade_category'].values)
X_test_project_grade_category= vectorizer.transform(X_test['project_grade_category'].values)
X_cv_project_grade_category = vectorizer.transform(X_cv['project_grade_category'].values)

# encoding clean_categories
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values)

X_train_clean_categories = vectorizer.transform(X_train['clean_categories'].values)
X_test_clean_categories = vectorizer.transform(X_test['clean_categories'].values)
X_cv_clean_categories = vectorizer.transform(X_cv['clean_categories'].values)

print(vectorizer.get_feature_names()[:5])

# encoding clean_subcategories
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values)

X_train_clean_subcategories = vectorizer.transform(X_train['clean_subcategories'].values)
X_test_clean_subcategories = vectorizer.transform(X_test['clean_subcategories'].values)
X_cv_clean_subcategories = vectorizer.transform(X_cv['clean_subcategories'].values)

print(vectorizer.get_feature_names()[:5])
```

```
['ak', 'al', 'ar', 'az', 'ca']
['dr', 'mr', 'mrs', 'ms', 'teacher']
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language']
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep']
```

In [34]:

```
# encoding numerical features
# encoding price

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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_cv_price_norm = normalizer.transform(X_cv['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(X_cv_price_norm[:10])

# encoding teacher_number_of_previously_posted_projects

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
.reshape(-1,1)
X_cv_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)).res
hape(-1,1)
X_test_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)).r
eshape(-1,1)

print('='*25)
print(X_train_teacher_number_of_previously_posted_projects_norm[:10])
```

```
[[3.52877633e-03]
 [2.28776275e-03]
 [8.63878135e-05]
 [2.11541978e-03]
 [1.71621867e-03]
 [5.71904632e-03]
 [3.15740969e-03]
 [2.29165669e-03]
 [8.28544222e-04]
 [2.45808663e-03]]
=========================
[[0.0002591 ]
 [0.02306013]
 [0.        ]
 [0.00064776]
 [0.00103641]
 [0.00012955]
 [0.        ]
 [0.00155462]
 [0.00155462]
 [0.00323878]]
```

In [35]:

```
# concatenating all features
from scipy.sparse import hstack

X_tr =
hstack((X_train_essay_bow,X_train_state,X_train_teacher_prefix,X_train_project_grade_category,X_tr
ain_clean_categories,X_train_clean_subcategories,X_train_price_norm,X_train_teacher_number_of_previ
usly_posted_projects_norm)).tocsr()
X_crv =
hstack((X_cv_essay_bow,X_cv_state,X_cv_teacher_prefix,X_cv_project_grade_category,X_cv_clean_categc
ries,X_cv_clean_subcategories,X_cv_price_norm,X_cv_teacher_number_of_previously_posted_projects_nor
)).tocsr()
X_te = hstack((X_test_essay_bow,X_test_state,X_test_teacher_prefix,X_test_project_grade_category,X
_test_clean_categories,X_test_clean_subcategories,X_test_price_norm,X_test_teacher_number_of_previc
sly_posted_projects_norm)).tocsr()

print("Final Data matrix")
print('='*50)
print(X_tr.shape, y_train.shape)
print(X_crv.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data matrix
==================================================
(65548, 10101) (65548,)
(21850, 10101) (21850,)
```

```
(21850, 10101) (21850,)
```

In [36]:

```
import pickle

pickle_out = open("BOW_data.pickle","wb")
pickle.dump({'X_tr':X_tr,'X_crv':X_crv,'X_te':X_te,'y_tr':y_train,'y_crv':y_cv,'y_te':y_test},pickl
e_out)
pickle_out.close()
```

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

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

In [37]:

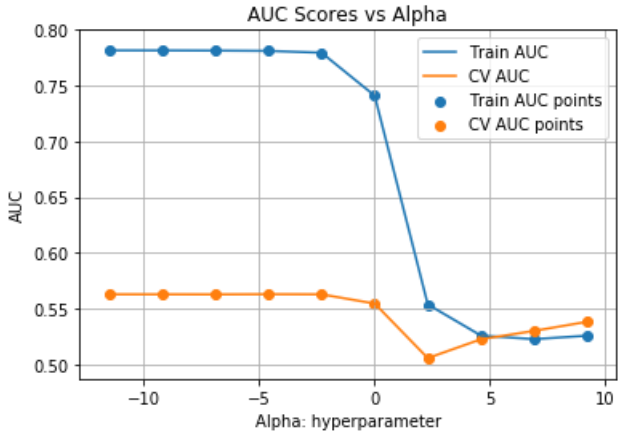
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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 [38]:

```
from sklearn.naive_bayes import MultinomialNB
from tqdm import tqdm
from sklearn.metrics import roc_auc_score
import numpy as np
import matplotlib.pyplot as plt
alphas = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
train_auc = []
cv_auc = []
for alpha in tqdm(alphas):
    clf = MultinomialNB(alpha=alpha, class_prior=[0.5,0.5])
    clf.fit(X_tr, y_train)

    y_train_pred = batch_predict(clf, X_tr)
    y_cv_pred = batch_predict(clf, X_crv)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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(np.log(alphas), train_auc, label='Train AUC')
plt.plot(np.log(alphas), cv_auc, label='CV AUC')
```

```
plt.scatter(np.log(alphas), train_auc, label='Train AUC points')
plt.scatter(np.log(alphas), cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Scores vs Alpha")
plt.grid()
plt.show()
```

```
100%|██████████| 10/10 [00:00<00:00, 13.83it/s]
```

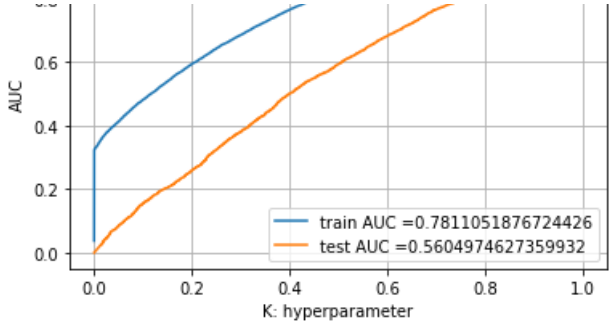


In [39]:

```
# Reference https://stackoverflow.com/questions/29867367/sklearn-multinomial-nb-most-informative-features
def show_most_informative_features(vectorizer, clf, n=20):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
    top = coefs_with_fns[:-(n + 1):-1]
    for (coef_1, fn_1) in top:
        print ("\t%.4f\t%-40s" % (coef_1, fn_1))
```

In [40]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

clf = MultinomialNB(alpha=0.00001, class_prior=[0.5,0.5]) #AUC score for cv is highest at 0.00001
clf.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(clf, X_tr)
y_test_pred = batch_predict(clf, X_te)

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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [41]:

```
# print 20 most relevant features
print(' Top 20 highest predictive features for the positive class')
print('============================================================')
print('              Highest Predictive Features')
print('============================================================')
show_most_informative_features(vectorizer_bow,clf)
```

```
 Top 20 highest predictive features for the positive class
============================================================
              Highest Predictive Features
============================================================
 -9.3251 disc golf
 -9.5615 cadets
 -9.5954 aed
 -9.6305 sports medicine
 -9.6305 queer
 -9.6669 gelli
 -9.7046 community club
 -9.7046 cajon
 -9.7847 tfk
 -9.7847 letters alive
 -9.7847 geocaching
 -9.8272 please please
 -9.8272 apple macbook pro
 -9.8717 medicine students
 -9.9182 vault
 -9.9182 team paws
 -9.9182 sports medicine students
 -9.9182 pole vault
 -9.9182 nspire
 -9.9182 drawing tablets
```

In [42]:

```
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    print('='*50)
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [43]:

```
from sklearn.metrics import confusion_matrix,accuracy_score
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print('Accuracy Score: ',accuracy_score(y_train, predict_with_best_t(y_train_pred, best_t)))
print('='*50)
print("Test confusion matrix")
```

```
print('Test confusion matrix')
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
print('Accuracy Score: ',accuracy_score(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.48291740215287965 for threshold 0.482
==================================================
Train confusion matrix
[[ 7333  2592]
 [19267 36356]]
Accuracy Score:  0.6665191920424727
==================================================
Test confusion matrix
[[ 1573  1735]
 [ 7138 11404]]
Accuracy Score:  0.5939130434782609
```

In [44]:

```
y = data['project_is_approved']
X = data.drop('project_is_approved',axis=1)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.25, stratify=y_train)
print(X_train.shape,X_cv.shape,X_test.shape,y_train.shape,y_cv.shape,y_test.shape)
```

```
(65548, 8) (21850, 8) (21850, 8) (65548,) (21850,) (21850,)
```

In [45]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_Vectorizer = TfidfVectorizer(ngram_range=(1,4),max_df=10,max_features=10000)
tfidf_Vectorizer.fit(X_train['essay'].values)

X_train_essay_tfidf = tfidf_Vectorizer.transform(X_train['essay'].values)
X_cv_essay_tfidf = tfidf_Vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = tfidf_Vectorizer.transform(X_test['essay'].values)

print('After Tfidf Vectorization')
print('='*50)
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('='*50)

print(tfidf_Vectorizer.get_feature_names()[:10])
```

```
After Tfidf Vectorization
==================================================
(65548, 10000) (65548,)
(21850, 10000) (21850,)
(21850, 10000) (21850,)
==================================================
['10 weeks', '100 percent free lunch', '100th day', '100th day school', '102', '106', '12 building
', '12 self', '12 self contained', '12 serious']
```

In [46]:

```
# encoding categorical features using tfidf vectorizer

# encoding school_state
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['school_state'].values)

X_train_state = vectorizer.transform(X_train['school_state'].values)
X_test_state = vectorizer.transform(X_test['school_state'].values)
X_cv_state = vectorizer.transform(X_cv['school_state'].values)

print(vectorizer.get_feature_names()[:5])

# encoding teacher_prefix
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values)
```

```
X_train_teacher_prefix = vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher_prefix = vectorizer.transform(X_test['teacher_prefix'].values)
X_cv_teacher_prefix = vectorizer.transform(X_cv['teacher_prefix'].values)

print(vectorizer.get_feature_names()[:5])

# encoding project_grade_category
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['project_grade_category'].values)

X_train_project_grade_category = vectorizer.transform(X_train['project_grade_category'].values)
X_test_project_grade_category= vectorizer.transform(X_test['project_grade_category'].values)
X_cv_project_grade_category = vectorizer.transform(X_cv['project_grade_category'].values)

# encoding clean_categories
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['clean_categories'].values)

X_train_clean_categories = vectorizer.transform(X_train['clean_categories'].values)
X_test_clean_categories = vectorizer.transform(X_test['clean_categories'].values)
X_cv_clean_categories = vectorizer.transform(X_cv['clean_categories'].values)

print(vectorizer.get_feature_names()[:5])

# encoding clean_subcategories
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values)

X_train_clean_subcategories = vectorizer.transform(X_train['clean_subcategories'].values)
X_test_clean_subcategories = vectorizer.transform(X_test['clean_subcategories'].values)
X_cv_clean_subcategories = vectorizer.transform(X_cv['clean_subcategories'].values)

print(vectorizer.get_feature_names()[:5])
```

```
['ak', 'al', 'ar', 'az', 'ca']
['dr', 'mr', 'mrs', 'ms', 'teacher']
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language']
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep']
```

In [47]:

```
X_tr_tfidf =
hstack((X_train_essay_tfidf,X_train_state,X_train_teacher_prefix,X_train_project_grade_category,X_
train_clean_categories,X_train_clean_subcategories,X_train_price_norm,X_train_teacher_number_of_pre
viously_posted_projects_norm)).tocsr()
X_crv_tfidf = hstack((X_cv_essay_tfidf,X_cv_state,X_cv_teacher_prefix,X_cv_project_grade_category,
X_cv_clean_categories,X_cv_clean_subcategories,X_cv_price_norm,X_cv_teacher_number_of_previously_po
ted_projects_norm)).tocsr()
X_te_tfidf =
hstack((X_test_essay_tfidf,X_test_state,X_test_teacher_prefix,X_test_project_grade_category,X_test_
clean_categories,X_test_clean_subcategories,X_test_price_norm,X_test_teacher_number_of_previously_p
sted_projects_norm)).tocsr()

print("Final Data matrix")
print('='*50)
print(X_tr_tfidf.shape, y_train.shape)
print(X_crv_tfidf.shape, y_cv.shape)
print(X_te_tfidf.shape, y_test.shape)
```

```
Final Data matrix
==================================================
(65548, 10101) (65548,)
(21850, 10101) (21850,)
(21850, 10101) (21850,)
```

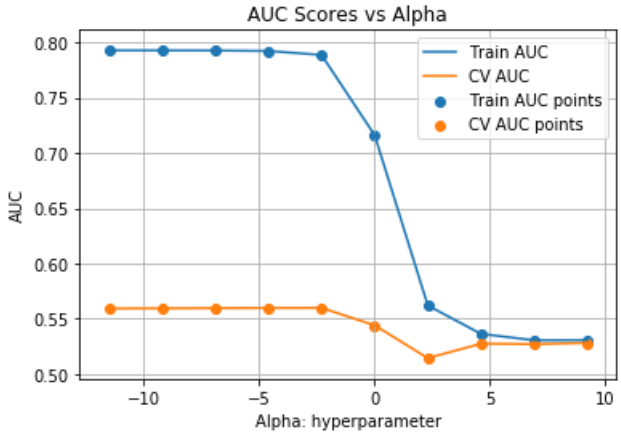
In [48]:

```
pickle_out = open("Tfidf_data.pickle","wb")
pickle.dump({'X_tr_tfidf':X_tr_tfidf,'X_crv_tfidf':X_crv_tfidf,'X_te_tfidf':X_te_tfidf,'y_tr':y_tra
in,'y_crv':y_cv,'y_te':y_test},pickle_out)
pickle_out.close()
```

In [49]:

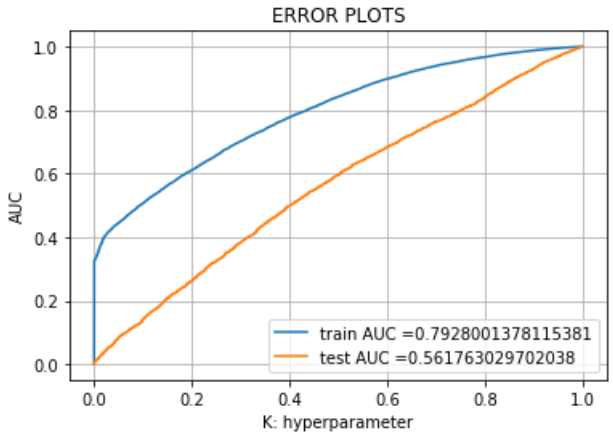
```
alphas = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
train_auc = []
cv_auc = []
for alpha in tqdm(alphas):
    clf = MultinomialNB(alpha=alpha, class_prior=[0.5,0.5])
    clf.fit(X_tr_tfidf, y_train)

    y_train_pred = batch_predict(clf, X_tr_tfidf)
    y_cv_pred = batch_predict(clf, X_crv_tfidf)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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(np.log(alphas), train_auc, label='Train AUC')
plt.plot(np.log(alphas), cv_auc, label='CV AUC')

plt.scatter(np.log(alphas), train_auc, label='Train AUC points')
plt.scatter(np.log(alphas), cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Scores vs Alpha")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████████████████████████████████████████████████| 10/10
[00:00<00:00, 11.77it/s]
```



In [50]:

```
clf = MultinomialNB(alpha=0.00001, class_prior=[0.5,0.5]) #AUC score for cv is highest at 0.00001
clf.fit(X_tr_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(clf, X_tr_tfidf)
y_test_pred = batch_predict(clf, X_te_tfidf)

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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [51]:

```
# print 20 most relevant features
print(' Top 20 highest predictive features for the positive class')
print('============================================================')
print('             Highest Predictive Features')
print('============================================================')
show_most_informative_features(tfidf_Vectorizer,clf)
```

```
 Top 20 highest predictive features for the positive class
============================================================
             Highest Predictive Features
============================================================
 -10.4762 8th graders attend middle
 -10.5366 school 83
 -10.5366 especially reading math
 -10.5508 individual spots
 -10.5581 geocaching
 -10.5588 may never get experience
 -10.5825 show joy
 -10.5825 day safe fun
 -10.5827 see students love
 -10.5832 in integrated
 -10.5971 like google classroom
 -10.5971 day we lot
 -10.6008 projects designed
 -10.6008 lunch despite struggles
 -10.6016 reach students our
 -10.6017 school they love coming
 -10.6073 cots
 -10.6119 interesting reading material
 -10.6120 backgrounds different economic
 -10.6147 available students nannan
```

In [52]:

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print('Accuracy Score: ',accuracy_score(y_train, predict_with_best_t(y_train_pred, best_t)))
print('='*50)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
print('Accuracy Score: ',accuracy_score(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.4952656981728967 for threshold 0.481
==================================================
Train confusion matrix
[[ 7287  2638]
 [18102 37521]]
Accuracy Score:  0.6835906511258925
==================================================
Test confusion matrix
[[ 1568  1740]
 [ 7036 11506]]
Accuracy Score:  0.5983524027459954
```

3. Summary

as mentioned in the step 5 of instructions

In [54]:

```
print('+------------+-------+-----------------------+')
print('| Vectorizer | Model | Hyper Parameter |  AUC |')
print('+------------+-------+-----------------------+')
print('|    BOW     | Brute |    0.00001      | 0.56 |')
print('+------------+-------+-----------------------+')
print('|   TFIDF    | Brute |    0.00001      | 0.56 |')
print('+------------+-------+-----------------------+')
```

```
+------------+-------+-----------------------+
| Vectorizer | Model | Hyper Parameter |  AUC |
+------------+-------+-----------------------+
|    BOW     | Brute |    0.00001      | 0.56 |
+------------+-------+-----------------------+
|   TFIDF    | Brute |    0.00001      | 0.56 |
+------------+-------+-----------------------+
```