Importing libraries

In [2]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
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
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

warnings.filterwarnings("ignore")
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter

from matplotlib import pyplot
from scipy.sparse import coo_matrix
from scipy.sparse import csr_matrix
from scipy.sparse import hstack
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
```

1. Importing Data

In [3]:

```
X = pd.read_csv('DonorsChoose_processed.csv')
project_data = pd.read_csv('train_data.csv')
project_data=project_data[0:50000]
y = project_data['project_is_approved'].values
```

2. Splitting Data

```
# 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)

print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(22445, 11) (22445,)
(11055, 11) (11055,)
(16500, 11) (16500,)
```

3. Vectorizing data

3.1 BOW Vectorization of Clean_essay

In [5]:

```
from sklearn.feature_extraction.text import CountVectorizer
vec_essay = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vec_essay.fit(X_train['clean_essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vec_essay.transform(X_train['clean_essay'].values)
X_cv_essay_bow = vec_essay.transform(X_cv['clean_essay'].values)
X_test_essay_bow = vec_essay.transform(X_test['clean_essay'].values)

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)
```

```
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
```

- https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

3.2 one hot encoding the catogorical features: state

In [6]:

```
vec_state = CountVectorizer(min_df=10)
vec_state.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vec_state.transform(X_train['school_state'].values)
X_cv_state_ohe = vec_state.transform(X_cv['school_state'].values)
X_test_state_ohe = vec_state.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(vec_state.get_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', 'k
s', '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
```

3.3 one hot encoding the catogorical features: teacher_prefix

In [7]:

```
vec_tpre = CountVectorizer(min_df=10)
vec_tpre.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vec_tpre.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vec_tpre.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vec_tpre.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(vec_tpre.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

3.4 one hot encoding the catogorical features: project_grade

In [8]:

```
vec_grade = CountVectorizer(min_df=10)
vec_grade.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vec_grade.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vec_grade.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vec_grade.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(vec_grade.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 3) (22445,)
(11055, 3) (11055,)
(16500, 3) (16500,)
['12', 'grades', 'prek']
====================================================================================================
```

3.5 one hot encoding the catogorical features: project_title

In [9]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_ohe = vec_title.transform(X_train['clean_title'].values)
X_cv_title_ohe = vec_title.transform(X_cv['clean_title'].values)
```

```
print("After vectorizations")
print(X_train_title_ohe.shape, y_train.shape)
print(X_cv_title_ohe.shape, y_cv.shape)
print(X_test_title_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1677) (22445,)
(11055, 1677) (11055,)
(16500, 1677) (16500,)
====================================================================================================
```

3.6 one hot encoding the catogorical features: project_category

In [10]:

```
vec_cate = CountVectorizer(min_df=10)
vec_cate.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_category_ohe = vec_cate.transform(X_train['clean_categories'].values)
X_cv_category_ohe = vec_cate.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vec_cate.transform(X_test['clean_title'].values)

print("After vectorizations")
print(X_train_category_ohe.shape, y_train.shape)
print(X_cv_category_ohe.shape, y_cv.shape)
print(X_test_category_ohe.shape, y_test.shape)
print(vec_cate.get_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_arts', 'specialneeds', 'warmth']
====================================================================================================
```

3.7 one hot encoding the catogorical features: project_subcategory

In [11]:

```
vec_scate = CountVectorizer(min_df=10)
vec_scate.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategory_ohe = vec_scate.transform(X_train['clean_subcategories'].values)
X_cv_subcategory_ohe = vec_scate.transform(X_cv['clean_subcategories'].values)
X_test_subcategory_ohe = vec_scate.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
print(X_cv_subcategory_ohe.shape, y_cv.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
print(vec_scate.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
```

==========

3.8 Normalizing the numerical features: Price

In [12]:

```
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,)
==========
```

3.9 Normalizing the numerical features: teacher no of previously posted projects

In [13]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_post_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].v
alues.reshape(-1,1))
X_cv_post_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.
reshape(-1,1))
X_test_post_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_post_norm.shape, y_train.shape)
print(X_cv_post_norm.shape, y_cv.shape)
print(X_test_post_norm.shape, y_test.shape)
print("="*100)
```

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

3.10 Normalizing the numerical features: resource summary

In [14]:

```
normalizer = Normalizer()
normalizer.fit(X_train['Is_digit_present'].values.reshape(-1,1))

X_train_digit_norm = normalizer.transform(X_train['Is_digit_present'].values.reshape(-1,1))
X_cv_digit_norm = normalizer.transform(X_cv['Is_digit_present'].values.reshape(-1,1))
X_test_digit_norm = normalizer.transform(X_test['Is_digit_present'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_digit_norm.shape, y_train.shape)
print(X_cv_digit_norm.shape, y_cv.shape)
print(X_test_digit_norm.shape, y_test.shape)
print("="*100)
```

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

3.11 Normalizing the numerical features: quantity

In [15]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['quantity'].values.reshape(-1,1))

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_quantity_norm = normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

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

Concatenation of all the vectorized data

In [16]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

X_tr =
hstack((X_train_essay_bow,X_train_state_ohe,X_train_teacher_ohe,X_train_grade_ohe,X_train_title_ohe
,X_train_category_ohe,X_train_subcategory_ohe))
X_cr = hstack((X_cv_essay_bow,X_cv_state_ohe,X_cv_teacher_ohe,X_cv_grade_ohe,X_cv_title_ohe,X_cv_ca
tegory_ohe,X_cv_subcategory_ohe))
X_te =
hstack((X_test_essay_bow,X_test_state_ohe,X_test_teacher_ohe,X_test_grade_ohe,X_test_title_ohe,X_te
st_category_ohe,X_test_subcategory_ohe))

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 6774) (22445,)
```

==========

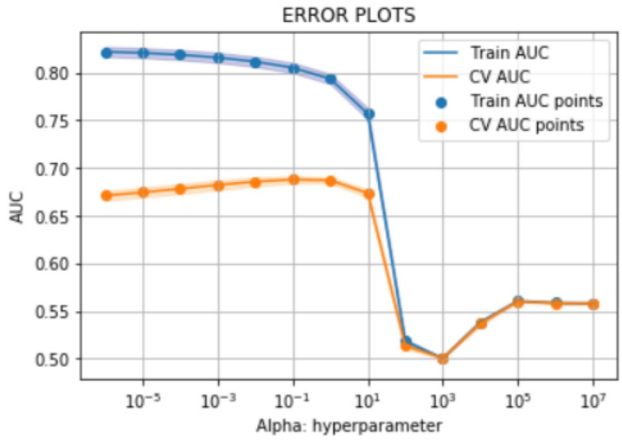
Applying MultinomialNB on BOW

In [25]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html


naive = MultinomialNB()
parameters = {'alpha':[0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000,100000,1000000,10
000000]}
clf = GridSearchCV(naive, parameters, cv=3, scoring='roc_auc')
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [26]:

```
plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color=
'darkorange')

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
pyplot.xscale('log')

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



From the above graph we can see that for the value of alpha less than 2, it is over fitting the data and for the value of k more than 2, it is underfitting the data. So the best value of k will be 2 for this

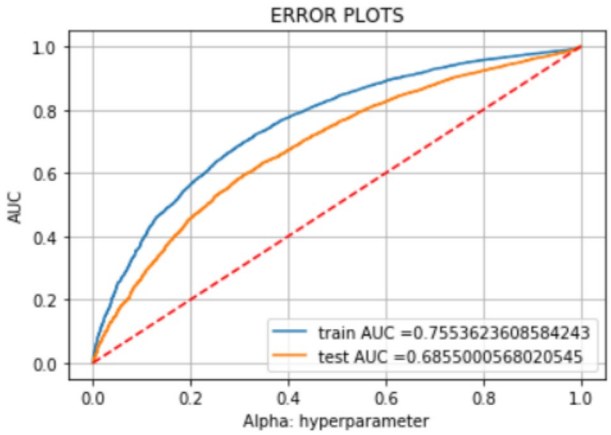
In [18]:

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

```
naive = MultinomialNB(alpha=best_alpha, class_prior=[1,1], fit_prior=False)
naive.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 = naive.predict_proba(X_tr)
preds = y_train_pred[:,1] #code copied from https://stackoverflow.com/questions/25009284/how-to-pl
ot-roc-curve-in-python to remove the error-bad input shape
y_test_pred = naive.predict_proba(X_te)
preds2=y_test_pred[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, preds)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, preds2)
```

In [35]:

```
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.plot([0, 1], [0, 1],'r--') #code copied from -https://stackoverflow.com/questions/25009284/how-
to-plot-roc-curve-in-python
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [20]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]

    # (tpr*(1-fpr)) will be maximum if your 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))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [21]:

```
from sklearn.metrics import confusion_matrix
train_conf=confusion_matrix(y_train, predict(preds, tr_thresholds, train_fpr, train_fpr))
test_conf=confusion_matrix(y_test, predict(preds2, tr_thresholds, test_fpr, test_fpr))
```

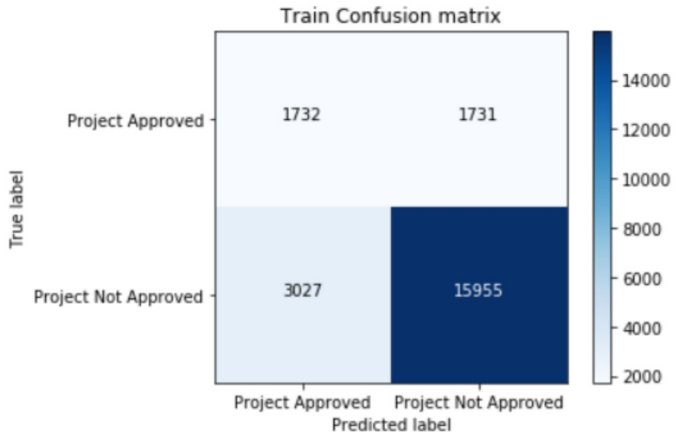
In [22]:

```
#sns heatmap confusion matrix -https://scikit-
learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
class_names=['Project Approved','Project Not Approved']
```

In [23]:

```
import itertools
plot_confusion_matrix(train_conf, classes=class_names,
                      title='Train Confusion matrix')
```

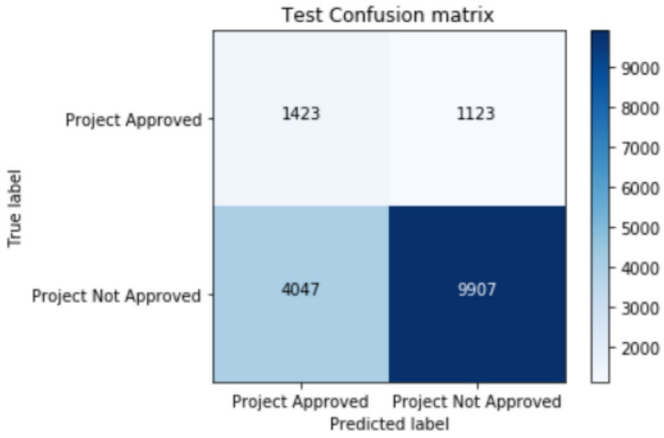
```
Confusion matrix, without normalization
[[ 1732  1731]
 [ 3027 15955]]
```



In [24]:

```
plot_confusion_matrix(test_conf, classes=class_names,
```

```
Confusion matrix, without normalization
[[1423 1123]
 [4047 9907]]
```



2.4.1.1 Top 10 important features of both the classes

In [25]:

```
#naive bayes important feature- https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-bayes

def top10(vec,naive2,n=10):
    feature =vec.get_feature_names()
    top10_Not_Approved = sorted(zip(naive2.feature_log_prob_[0], feature))[:n]
    top10_Approved = sorted(zip(naive2.feature_log_prob_[1], feature))[:n]
    print("Top 10 important features in essay of Not Approved project")
    for log_prob, string in top10_Not_Approved:
        print(log_prob, string)
    print("="*100)
    print("Top 10 important features in essay of Approved project")
    for log_prob, string in top10_Approved:
        print(log_prob, string)
```

In [26]:

```
top10(vec_essay,naive)
```

```
Top 10 important features in essay of Not Approved project
-11.65753303958813 kindle fire
-11.475211482794174 chairs help
-11.321060802966915 chairs allow
-11.187529410342393 chromebooks allow
-11.187529410342393 classroom rug
-11.187529410342393 macbook
-11.187529410342393 opportunity move
-11.187529410342393 technology learning
-11.069746374686009 book bins
-11.069746374686009 bottles
====================================================================================================

Top 10 important features in essay of Approved project
-10.888338931354124 hands learners
-10.510576425795294 materials provide
-10.455918013257431 materials allow students
-10.445335903926892 impacts
-10.434864604059598 make choices
-10.424501817024051 help foster
-10.424501817024051 lives better
-10.424501817024051 science classroom
-10.414245316856862 group setting
-10.414245316856862 healthy eating
```

Conclusion

On applying MultinomialNB BOW on DonorsChoose dataset with alpha value as 2 we got the AUC value of 0.68 on test data

In [36]:

```
#code copied from -http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Hyper Parameter (alpha)", "AUC"]
x.add_row(["BOW",  2, 0.68])
x.add_row(["TFIDF",  2, 0.64])
print(x)
```

```
+------------+------------------------+------+
| Vectorizer | Hyper Parameter (alpha) | AUC  |
+------------+------------------------+------+
|    BOW     |            2           | 0.68 |
|   TFIDF    |            2           | 0.64 |
+------------+------------------------+------+
```