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
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
from scipy.sparse import hstack,vstack,csr_matrix
from sklearn.model_selection import PredefinedSplit
from sklearn.metrics import roc_auc_score,confusion_matrix,roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, chi2
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from sklearn.preprocessing import StandardScaler,Normalizer
import scipy
from tqdm import tqdm as tqdm
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
%matplotlib inline
```

1.Reading and Splitting Data

In [2]:

```
#considering 50k points only due to constraints
data=pd.read_csv('/home/sathish/Desktop/appliedai/Donors_choose_preprocessed_data.csv'
,na_filter=False)
data=data.sample(50000,random_state=42)
y=data['project_is_approved'].values
x=data.drop(['project_is_approved'],axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,stratify=y)
```

2. TruncatedSVD

2.1 Selecting top 2000 words from `essay` and `project_title`

In [3]:

```
train_data=x_train['essay']+" "+x_train['project_title']
vectorizer=TfidfVectorizer()
```

```
tfidf_vectors=vectorizer.fit_transform(train_data.values)
sorted_index=np.argsort(vectorizer.idf_)[-2000:]
vocab=np.array(vectorizer.get_feature_names())
top2k_words=vocab[sorted_index]
word_dict=dict(zip(top2k_words,list(range(2000))))
```

2.2 Computing Co-occurance matrix

Check for toy exmaple

In [4]:

```
toy_data=["abc def ijk pqr", "pqr klm opq","lmn pqr xyz abc def pqr abc"]
toy_top2k_words=["abc", "pqr", "def"]
toy_word_dict=dict(zip(toy_top2k_words,list(range(len(toy_top2k_words)))))

size=len(toy_top2k_words)
window_size=2
train_matrix=np.zeros((size,size))
for doc in toy_data:
    doc_words=doc.split()
    len_doc=len(doc_words)
    for i,focus_word in enumerate(doc_words):
        if focus_word not in toy_top2k_words:
            continue
        focus_word_index=toy_word_dict[focus_word]
        start=i-window_size
        end=i+window_size
        for j in range(start,end+1):
            if j<len_doc and j>=0:
                target=doc_words[j]
                if target in toy_top2k_words:
                    target_index=toy_word_dict[target]
                    train_matrix[focus_word_index][target_index]+=1
print(train_matrix)
```

```
[[3. 3. 3.]
 [3. 4. 2.]
 [3. 2. 2.]]
```

In [5]:

```
size=len(top2k_words)
window_size=5
train_matrix=np.zeros((size,size))
for doc in train_data:
    doc_words=doc.split()
    len_doc=len(doc_words)
    for i,focus_word in enumerate(doc_words):
        if focus_word not in top2k_words:
```

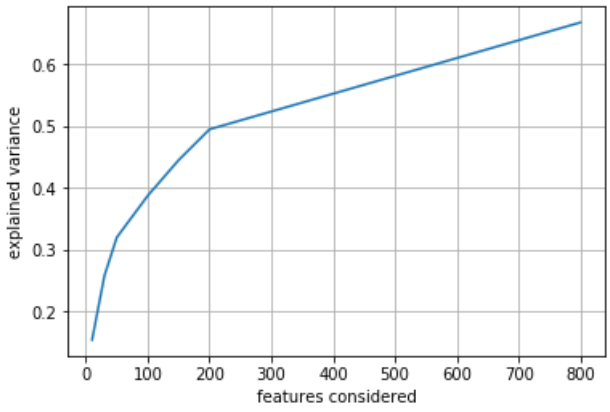
```
            continue
        focus_word_index=word_dict[focus_word]
        start=i-window_size
        end=i+window_size
        for j in range(start,end+1):
            if j<len_doc and j>=0:
                target=doc_words[j]
                if target in top2k_words:
                    target_index=word_dict[target]
                    train_matrix[focus_word_index][target_index]+=1
```

2.3 Applying TruncatedSVD and Calculating Vectors for essay and project_title¶

In [7]:

```
num_components=[10,30,50,100,150,200,300,500,600,700,800]
explained_variance=[]
for num in tqdm(num_components):
    svd = TruncatedSVD(n_components=num,random_state=42)
    svd.fit(train_matrix)
    explained_variance.append(svd.explained_variance_ratio_.sum())
plt.plot(num_components,explained_variance)
plt.xlabel('features considered')
plt.ylabel('explained variance')
plt.grid(True)
plt.show()
```

```
100%|██████████| 11/11 [00:35<00:00,  5.85s/it]
```



In [8]:

```
#we will consider number of components=200 as it is able to explain 50% the variance.
svd = TruncatedSVD(n_components=200,random_state=42)
svd.fit(train_matrix)

keys=top2k_words
values=svd.fit_transform(train_matrix)
```

```
values=svd.fit_transform(train_matrix)
word_vectors=dict(zip(keys,values))

train_avg_word_vectors_title = [];
for sentence in tqdm(x_train['project_title']):
    vector = np.zeros(200)
    cnt_words =0;
    for word in sentence.split():
        if word in word_vectors:
            vector += word_vectors[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_avg_word_vectors_title.append(vector)

train_avg_word_vectors_essay = [];
for sentence in tqdm(x_train['essay']):
    vector = np.zeros(200)
    cnt_words =0;
    for word in sentence.split():
        if word in word_vectors:
            vector += word_vectors[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_avg_word_vectors_essay.append(vector)


test_avg_word_vectors_title = [];
for sentence in tqdm(x_test['project_title']):
    vector = np.zeros(200)
    cnt_words =0;
    for word in sentence.split():
        if word in word_vectors:
            vector += word_vectors[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_avg_word_vectors_title.append(vector)

test_avg_word_vectors_essay = [];
for sentence in tqdm(x_test['essay']):
    vector = np.zeros(200)
    cnt_words =0;
    for word in sentence.split():
        if word in word_vectors:
            vector += word_vectors[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_avg_word_vectors_essay.append(vector)
```

```
100%|██████████| 33500/33500 [00:00<00:00, 260289.44it/s]
100%|██████████| 33500/33500 [00:01<00:00, 27803.13it/s]
100%|██████████| 16500/16500 [00:00<00:00, 290355.81it/s]
100%|██████████| 16500/16500 [00:00<00:00, 26444.48it/s]
```

3. Creating Data Matrix

3.1 one hot encoding the categorical features

In [9]:

```
vectorizer=CountVectorizer()
x_train_state_onehot=vectorizer.fit_transform(x_train['school_state'].values)
x_test_state_onehot=vectorizer.transform(x_test['school_state'].values)


x_train_tprefix_onehot=vectorizer.fit_transform(x_train['teacher_prefix'].values)
x_test_tprefix_onehot=vectorizer.transform(x_test['teacher_prefix'].values)

x_train_grade_onehot=vectorizer.fit_transform(x_train['project_grade_category'].values
)
x_test_grade_onehot=vectorizer.transform(x_test['project_grade_category'].values)

x_train_cat_onehot=vectorizer.fit_transform(x_train['project_subject_categories'].valu
es)
x_test_cat_onehot=vectorizer.transform(x_test['project_subject_categories'].values)

x_train_subcat_onehot=vectorizer.fit_transform(x_train['project_subject_subcategories'
].values)
x_test_subcat_onehot=vectorizer.transform(x_test['project_subject_subcategories'].valu
es)
```

3.2 Normalizing the numerical features

In [10]:

```
standardscaler=StandardScaler()
x_train_price_norm=standardscaler.fit_transform(x_train['price'].values.reshape(-1,1))
x_test_price_norm=standardscaler.transform(x_test['price'].values.reshape(-1,1))

x_train_quantity_norm=standardscaler.fit_transform(x_train['quantity'].values.reshape(
-1,1))
x_test_quantity_norm=standardscaler.transform(x_test['quantity'].values.reshape(-1,1))

x_train_prev_projects_norm=standardscaler.fit_transform(x_train['teacher_number_of_pre
viously_posted_projects'].values.reshape(-1,1))
x_test_prev_projects_norm=standardscaler.transform(x_test['teacher_number_of_previousl
```

```
y_posted_projects'].values.reshape(-1,1))
```

In [11]:

```
def compute_Polarity_Score(sent):
    # function to compute vader sentiment scores and returns list of 4 values which re
present pos,neg,neutral and compound
    analyser = SentimentIntensityAnalyzer()
    scores=analyser.polarity_scores(sent)
    return list(scores.values())

train_polarity_scores=np.array(x_train['essay'].map(compute_Polarity_Score).tolist())
test_polarity_scores=np.array(x_test['essay'].map(compute_Polarity_Score).tolist())



#number of words in title
numwords_title_train=x_train['project_title'].str.split().map(len)
numwords_title_test=x_test['project_title'].str.split().map(len)

#normalizer=Normalizer()
standardscaler=StandardScaler()

x_train_numwords_title=standardscaler.fit_transform(numwords_title_train.values.reshap
e(-1,1))
x_test_numwords_title=standardscaler.transform(numwords_title_test.values.reshape(-1,1
))




#number of words in essay
numwords_essay_train=x_train['essay'].str.split().map(len)
numwords_essay_test=x_test['essay'].str.split().map(len)

x_train_numwords_essay=standardscaler.fit_transform(numwords_essay_train.values.reshap
e(-1,1))
x_test_numwords_essay=standardscaler.transform(numwords_essay_test.values.reshape(-1,1
))
```

3.3 Merging all features

In [12]:

```
#text features
bow_train=hstack((csr_matrix(train_avg_word_vectors_essay),train_avg_word_vectors_titl
e))
bow_test=hstack((csr_matrix(test_avg_word_vectors_essay),test_avg_word_vectors_title))


#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_test_cat=hstack((x_test_state_onehot,x_test_tprefix_onehot,x_test_grade_onehot,x_tes
t_cat_onehot,x_test_subcat_onehot))

#numerical features
x_train_num=hstack((csr_matrix(x_train_price_norm),x_train_quantity_norm,x_train_prev_
projects_norm))
x_test_num=hstack((csr_matrix(x_test_price_norm),x_test_quantity_norm,x_test_prev_proj
ects_norm))

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

4. Apply XGBoost on the Final Features from the above section¶

In [13]:

```
#here I have splitted the total data into train and val in order to evalaute the class
fier.
#But in real world  we have to preprocess validation separately

from xgboost import XGBClassifier
estimators=[5,10,50,100,200,300]
depths=[1,3,5,10,20,30]
train_auc=[]
val_auc=[]
auc_values=[]
for n_estimators in tqdm(estimators):
    for depth in depths:
        clf=XGBClassifier(n_estimators=n_estimators,max_depth=depth,n_jobs=-1,random_s
tate=18)
        clf.fit(final_bow_train,y_train)
        train_pred=clf.predict_proba(final_bow_train)[:,1]
        val_pred=clf.predict_proba(final_bow_test)[:,1]
        auc_values.append([n_estimators,depth,roc_auc_score(y_train,train_pred),'train
'])
        auc_values.append([n_estimators,depth,roc_auc_score(y_test,val_pred),'val'])
a=np.array(auc_values)
df=pd.DataFrame(a,columns=['n_estimators','max_depth','auc','train_val'])
```

```
100%|██████████| 6/6 [06:12<00:00, 84.15s/it]
```

In [14]:

```
df.iplot(kind='scatter3d',x='n_estimators',y='max_depth',z='auc',xTitle='n_estimators'
,yTitle='max_depth',zTitle='auc',categories='train_val',theme='solar')
```

5. Conclusions

- We are getting 67% auc with 50k data points only.
- we are able to preserve 50% of the variance of top two thousand words by just 200 components only.