DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project essay 3`	Third application essay*

Feature	Description
`project_essay_4`	Fourth application essay
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

In [3]:

```
%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 TfidfVectorizer
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
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm
import os
import chart_studio.plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
from scipy.sparse import hstack,vstack
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import model_selection
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from prettytable import PrettyTable
from sklearn.preprocessing import Normalizer
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
import pdb
from sklearn.cluster import KMeans
from sklearn.feature_selection import SelectKBest,f_classif
import graphviz
from sklearn import tree
from graphviz import Source
from sklearn.externals.six import StringIO
from IPython.display import Image
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
from sklearn.neighbors import KDTree
from sklearn.cluster import DBSCAN
from wordcloud import WordCloud, STOPWORDS
import pydotplus
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\arjun\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

1.1 Reading Data

In [4]:

```
Project_data = pd.read_csv('train_data.csv')
Resource_data = pd.read_csv('resources.csv')
print(Project_data.shape)
print(Resource_data.shape)
```

```
(109248, 17)
(1541272, 4)
```

In [5]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(Project_data.columns)]
#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
Project_data['Date'] = pd.to_datetime(Project_data['project_submitted_datetime'])
Project_data.drop('project_submitted_datetime', axis=1, inplace=True)
Project_data.sort_values(by=['Date'], inplace=True)
```

```
# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
Project_data = Project_data[cols]
Project_data.head(2)
```

Out[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_:
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	

2. Clustering

Choose the best data matrix on which you got the best AUC

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

In [6]:

```
Y_T = Project_data['project_is_approved'].values
Project_data.drop(['project_is_approved'], axis=1, inplace=True)
n_z = len(Project_data)
y_z = np.zeros(n_z, dtype=np.int32)

X = Project_data
# train test split
X_Train, X_Test, Y, Y_Test = train_test_split(X, Y_T, test_size=0.89, random_state=0, stratify=y_z)


print('Shape of X_Train: ',X_Train.shape)
print('Shape of Y: ',Y.shape)
print('Shape of X_Test: ',X_Test.shape)
print('Shape of Y_Test: ',Y_Test.shape)
```

```
Shape of X_Train:  (12017, 16)
Shape of Y:  (12017,)
Shape of X_Test:  (97231, 16)
Shape of Y_Test:  (97231,)
```

1.2 preprocessing of `project_subject_categories`

In [7]:

```
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
catogories_train = list(X_Train['project_subject_categories'].values)
cat_list = []
for i in catogories_train:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
```

```
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

X_Train['clean_categories'] = cat_list
X_Train.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in X_Train['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict_train = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))

print(len(sorted_cat_dict_train))
```

```
9
```

1.3 preprocessing of `project_subject_subcategories`

In [8]:

```
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
sub_catogories_train = list(X_Train['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories_train:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

X_Train['clean_subcategories'] = sub_cat_list
X_Train.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in X_Train['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict_train = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))

print(len(sorted_sub_cat_dict_train))
```

```
30
```

1.3 Text preprocessing

In [9]:

```
# merge two column text dataframe:
X_Train["essay"] = X_Train["project_essay_1"].map(str) +\
                        X_Train["project_essay_2"].map(str) + \
                        X_Train["project_essay_3"].map(str) + \
                        X_Train["project_essay_4"].map(str)
```

In [10]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [11]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've"
,\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [12]:

```
# Combining all the above stundents
# tqdm is for printing the status bar

preprocessed_essays_train = []


for sentance in tqdm(X_Train['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays_train.append(sent.lower().strip())


print("Shape of preprocessed_essays_train after preprocessing",len(preprocessed_essays_train))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 12017/12017
[00:09<00:00, 1327.83it/s]
```

```
Shape of preprocessed_essays_train after preprocessing 12017
```

In [13]:

```
word_count_essay_train = []
for a in tqdm(X_Train["essay"]) :
    b = len(a.split())
    word_count_essay_train.append(b)

X_Train["word_count_essay_train"] = word_count_essay_train
```

```
100%|█████████████████████████████████████████████████████████████████████████| 12017/12017
[00:00<00:00, 46174.73it/s]
```

1.4 Preprocessing of `project_title`

In [14]:

```
preprocessed_titles_train = []
# tqdm is for printing the status bar
for sentance in tqdm(X_Train['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles_train.append(sent.lower().strip())




print("Shape of preprocessed_titles_train after preprocessing",len(preprocessed_titles_train))
```

```
100%|█████████████████████████████████████████████████████████████████████████| 12017/12017
[00:00<00:00, 28576.11it/s]
```

```
Shape of preprocessed_titles_train after preprocessing 12017
```

In [15]:

```
word_count_title_train = []
for a in tqdm(X_Train["project_title"]) :
    b = len(a.split())
    word_count_title_train.append(b)

X_Train["word_count_title_train"] = word_count_title_train
```

```
100%|████████████████████████████████████████████████████████████████████████| 12017/12017
[00:00<00:00, 398473.80it/s]
```

Make Data Model Ready: encoding numerical, categorical features

1.5 Preparing data for models

1.5.1 Vectorizing Categorical data

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

In [16]:

```
# we use count vectorizer to convert the values into one
vectorizer_cat = CountVectorizer(vocabulary=list(sorted_cat_dict_train.keys()), lowercase=False, binary=True)
vectorizer_cat.fit(X_Train['clean_categories'].values)
categories_one_hot_train = vectorizer_cat.transform(X_Train['clean_categories'].values)

print(vectorizer_cat.get_feature_names())
print("Shape of categories_one_hot_train matrix after one hot encodig ",categories_one_hot_train.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of categories_one_hot_train matrix after one hot encodig  (12017, 9)
```

In [17]:

```
# we use count vectorizer to convert the values into one
vectorizer_sub_cat = CountVectorizer(vocabulary=list(sorted_sub_cat_dict_train.keys()), lowercase=False, binary=True)
sub_categories_one_hot_train = vectorizer_sub_cat.fit_transform(X_Train['clean_subcategories'].values)

print(vectorizer_sub_cat.get_feature_names())
print("Shape of sub_categories_one_hot_train matrix after one hot encodig ",sub_categories_one_hot_train.shape)
```

```
['Economics', 'FinancialLiteracy', 'CommunityService', 'ForeignLanguages', 'ParentInvolvement', 'C
ivics_Government', 'Extracurricular', 'Warmth', 'Care_Hunger', 'NutritionEducation',
'PerformingArts', 'SocialSciences', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'EarlyDevelopment', 'Health_LifeScience', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of sub_categories_one_hot_train matrix after one hot encodig  (12017, 30)
```

School State

In [18]:

```
sch1_catogories = list(X_Train['school_state'].values)
school_list = []
for sent in sch1_catogories:
    school_list.append(sent.lower().strip())
X_Train['school_categories'] = school_list
X_Train.drop(['school_state'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter_sch = Counter()
for word in X_Train['school_categories'].values:
    my_counter_sch.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
sch_dict = dict(my_counter_sch)
sorted_sch_dict = dict(sorted(sch_dict.items(), key=lambda kv: kv[1]))

vectorizer_sch = CountVectorizer(vocabulary=list(sorted_sch_dict.keys()), lowercase=False, binary=True)
vectorizer_sch.fit(X_Train['school_categories'].values)
#print(vectorizer.get_feature_names())

sch_one_hot_train = vectorizer_sch.transform(X_Train['school_categories'].values)
print("Shape of sch_one_hot_train matrix after one hot encodig ",sch_one_hot_train.shape)
#-------------------------------------------------------------------------------------------
```

```
Shape of sch_one_hot_train matrix after one hot encodig  (12017, 51)
```

Prefix

In [19]:

```
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
prefix_catogories_train = list(X_Train['teacher_prefix'].values)
prefix_list_train = []
for sent in prefix_catogories_train:
    sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    prefix_list_train.append(sent.lower().strip())
X_Train['prefix_catogories'] = prefix_list_train
X_Train.drop(['teacher_prefix'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter_prefix_train = Counter()
for word in X_Train['prefix_catogories'].values:
    my_counter_prefix_train.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
prefix_dict_train = dict(my_counter_prefix_train)
sorted_prefix_dict_train = dict(sorted(prefix_dict_train.items(), key=lambda kv: kv[1]))

vectorizer_prefix = CountVectorizer(vocabulary=list(sorted_prefix_dict_train.keys()), lowercase=False, binary=True)
vectorizer_prefix.fit(X_Train['prefix_catogories'].values)
#print(vectorizer.get_feature_names())

prefix_one_hot_train = vectorizer_prefix.transform(X_Train['prefix_catogories'].values)
print("Shape of prefix_one_hot_train matrix after one hot encodig ",prefix_one_hot_train.shape)
```

```
Shape of prefix_one_hot_train matrix after one hot encodig  (12017, 5)
```

project_grade_category

In [20]:

```
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
grade_catogories_train = list(X_Train['project_grade_category'].values)
grade_list_train = []
for sent in grade_catogories_train:
    sent = sent.replace('-','_')
    sent = sent.replace(' ','_')
    # sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    grade_list_train.append(sent.lower().strip())

# temp = temp.replace('-','_')
X_Train['new_grade_category'] = grade_list_train
X_Train.drop(['project_grade_category'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter_grade_train = Counter()
for word in X_Train['new_grade_category'].values:
    my_counter_grade_train.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grade_dict_train = dict(my_counter_grade_train)
sorted_grade_dict_train = dict(sorted(grade_dict_train.items(), key=lambda kv: kv[1]))


vectorizer_grade = CountVectorizer(vocabulary=list(sorted_grade_dict_train.keys()), lowercase=False, binary=True)
vectorizer_grade.fit(X_Train['new_grade_category'].values)
#print(vectorizer.get_feature_names())

grade_one_hot_train = vectorizer_grade.transform(X_Train['new_grade_category'].values)
grade_one_hot_train = vectorizer_grade.transform(X_Train['new_grade_category'].values)
print("Shape of grade_one_hot_train matrix after one hot encodig ",grade_one_hot_train.shape)
```

```
Shape of grade_one_hot_train matrix after one hot encodig  (12017, 4)
```

1.5.2 Make Data Model Ready: Vectorizing Numerical features

Price and Quantity data

In [21]:

```
price_data = Resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
X_Train = pd.merge(X_Train, price_data, on='id', how='left')
```

In [22]:

```
price_norm = Normalizer(norm='l2', copy=False)
price_norm.fit(X_Train['price'].values.reshape(1,-1))

price_norm.transform(X_Train['price'].values.reshape(1,-1))

price_norm_train = (X_Train['price'].values.reshape(-1,1))

print("Shape of price_norm_train matrix after one hot encodig ",price_norm_train.shape)
```

```
Shape of price_norm_train matrix after one hot encodig  (12017, 1)
```

In [23]:

```
quantity_norm = Normalizer(norm='l2', copy=False)
quantity_norm.fit(X_Train['quantity'].values.reshape(1,-1))

quantity_norm_train = quantity_norm.transform(X_Train['quantity'].values.reshape(1,-1))

quantity_norm_train = (X_Train['quantity'].values.reshape(-1,1))

print("Shape of quantity_norm_train matrix after one hot encodig ",quantity_norm_train.shape)
```

```
Shape of quantity_norm_train matrix after one hot encodig  (12017, 1)
```

teacher_number_of_previously_posted_projects

In [24]:

```
teacher_prev_post_norm = Normalizer(norm='l2', copy=False)
teacher_prev_post_norm.fit(X_Train['teacher_number_of_previously_posted_projects'].values.reshape(
1,-1))

teacher_prev_post_norm_train =
teacher_prev_post_norm.transform(X_Train['teacher_number_of_previously_posted_projects'].values.re
shape(1,-1))

teacher_prev_post_norm_train =
(X_Train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("Shape of teacher_prev_post_norm_train matrix after one hot encodig
",teacher_prev_post_norm_train.shape)
```

```
Shape of teacher_prev_post_norm_train matrix after one hot encodig  (12017, 1)
```

Title word count

In [25]:

```
title_norm = Normalizer(norm='l2', copy=False)
title_norm.fit(X_Train['word_count_title_train'].values.reshape(1,-1))
```

```
word_count_title_train = title_norm.transform(X_Train['word_count_title_train'].values.reshape(1,-1
))

word_count_title_train = (X_Train['word_count_title_train'].values.reshape(-1,1))

print(word_count_title_train.shape)
```

```
(12017, 1)
```

Essay word count

In [26]:

```
essay_norm = Normalizer(norm='l2', copy=False)
essay_norm.fit(X_Train['word_count_essay_train'].values.reshape(1,-1))
word_count_essay_train = essay_norm.transform(X_Train['word_count_essay_train'].values.reshape(1,-1
))

word_count_essay_train = (X_Train['word_count_essay_train'].values.reshape(-1,1))

print(word_count_essay_train.shape)
```

```
(12017, 1)
```

Sentiment Scores

In [27]:

```
# https://www.geeksforgeeks.org/python-sentiment-analysis-using-vader/
sid = SentimentIntensityAnalyzer()
essays = X_Train['essay']

sentiment_pos_essay_Train = []
sentiment_neg_essay_Train = []
sentiment_neut_essay_Train = []
sentiment_com_essay_Train = []

for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    sentiment_pos_essay_Train.append(res['pos'])
    sentiment_neg_essay_Train.append(res['neg'])
    sentiment_neut_essay_Train.append(res['neu'])
    sentiment_com_essay_Train.append(res['compound'])
X_Train['sentiment_pos_essay_Train'] = sentiment_pos_essay_Train
X_Train['sentiment_neg_essay_Train'] = sentiment_neg_essay_Train
X_Train['sentiment_neut_essay_Train'] = sentiment_neut_essay_Train
X_Train['sentiment_com_essay_Train'] = sentiment_com_essay_Train


sentiment_norm_pos = Normalizer(norm='l2', copy=False)
sentiment_norm_neg = Normalizer(norm='l2', copy=False)
sentiment_norm_neut = Normalizer(norm='l2', copy=False)
sentiment_norm_com = Normalizer(norm='l2', copy=False)

sentiment_norm_pos.fit(X_Train['sentiment_pos_essay_Train'].values.reshape(1,-1))
sentiment_norm_neg.fit(X_Train['sentiment_neg_essay_Train'].values.reshape(1,-1))
sentiment_norm_neut.fit(X_Train['sentiment_neut_essay_Train'].values.reshape(1,-1))
sentiment_norm_com.fit(X_Train['sentiment_com_essay_Train'].values.reshape(1,-1))

senti_pos_ess_Tr_norm = sentiment_norm_pos.transform(X_Train['sentiment_pos_essay_Train'].values.r
eshape(1,-1))
senti_pos_ess_Tr_norm = (X_Train['sentiment_pos_essay_Train'].values.reshape(-1,1))

senti_neg_ess_Tr_norm = sentiment_norm_neg.transform(X_Train['sentiment_neg_essay_Train'].values.r
eshape(1,-1))
senti_neg_ess_Tr_norm = (X_Train['sentiment_neg_essay_Train'].values.reshape(-1,1))

senti_neut_ess_Tr_norm =
sentiment_norm_neut.transform(X_Train['sentiment_neut_essay_Train'].values.reshape(1,-1))
senti_neut_ess_Tr_norm = (X_Train['sentiment_neut_essay_Train'].values.reshape(-1,1))

senti_com_ess_Tr_norm = sentiment_norm_com.transform(X_Train['sentiment_com_essay_Train'].values.r
eshape(1,-1))
eshape(1,-1))
senti_com_ess_Tr_norm = (X_Train['sentiment_com_essay_Train'].values.reshape(-1,1))


print("Shape of senti_pos_ess_Tr_norm matrix after one hot encodig ",senti_pos_ess_Tr_norm.shape)
print("Shape of senti_neg_ess_Tr_norm matrix after one hot encodig ",senti_neg_ess_Tr_norm.shape)
print("Shape of senti_neut_ess_Tr_norm matrix after one hot encodig ",senti_neut_ess_Tr_norm.shape
)
print("Shape of senti_com_ess_Tr_norm matrix after one hot encodig ",senti_com_ess_Tr_norm.shape)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 12017/12017 [00:
41<00:00, 291.53it/s]
```

```
Shape of senti_pos_ess_Tr_norm matrix after one hot encodig  (12017, 1)
Shape of senti_neg_ess_Tr_norm matrix after one hot encodig  (12017, 1)
Shape of senti_neut_ess_Tr_norm matrix after one hot encodig  (12017, 1)
Shape of senti_com_ess_Tr_norm matrix after one hot encodig  (12017, 1)
```

2.3 Make Data Model Ready: encoding essay, and project_title

1.5.3 Vectorizing Text data

1.5.2.2 TFIDF vectorizer

In [28]:

```
vectorizer_essays_tfidf = TfidfVectorizer(min_df=10,max_features=5000)
X_Tr_Es_DS = pd.DataFrame()
X_Tr_Es_DS['essay'] = preprocessed_essays_train
text_tfidf_train = vectorizer_essays_tfidf.fit_transform(preprocessed_essays_train)

print("Shape of matrix after one hot encodig ",text_tfidf_train.shape)
```

```
Shape of matrix after one hot encodig  (12017, 5000)
```

TFIDF vectorizer for Project Title

In [29]:

```
vectorizer_titles_tfidf = TfidfVectorizer(min_df=10,max_features=5000)
X_Tr_Es_DS['title'] = preprocessed_titles_train
title_tfidf_train = vectorizer_titles_tfidf.fit_transform(preprocessed_titles_train)

print("Shape of matrix(title) after one hot encoding ",title_tfidf_train.shape)
```

```
Shape of matrix(title) after one hot encoding  (12017, 779)
```

1.5.4 Merging all the above features

- we need to merge all the vectors i.e catogorical, text, numerical vectors

Merging vectorised Train data

In [30]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_Train =
hstack((categories_one_hot_train,sub_categories_one_hot_train,sch_one_hot_train,grade_one_hot_train
,prefix_one_hot_train, text_tfidf_train,title_tfidf_train, price_norm_train, quantity_norm_train, t
eacher_prev_post_norm_train, word_count_essay_train, word_count_title_train, senti_pos_ess_Tr_norm
,senti_neg_ess_Tr_norm,senti_neut_ess_Tr_norm,senti_com_ess_Tr_norm))
print(X_Train.shape)

print(Y.shape)
```

```
(12017, 5887)
(12017,)
```

Assignment 10: Clustering

- step 1: Choose any vectorizer (data matrix) that you have worked in any of the assignments, and got the best AUC value.
- step 2: Choose any of the feature selection/reduction algorithms ex: selectkbest features, pretrained word vectors, model based feature selection etc and reduce the number of features to 5k features
- step 3: Apply all three kmeans, Agglomerative clustering, DBSCAN
  - **K-Means Clustering:**
    ● Find the best 'k' using the elbow-knee method (plot k vs inertia_)
  - **Agglomerative Clustering:**
    ● Apply agglomerative algorithm and try a different number of clusters like 2,5 etc.
    ● You can take less data points (as this is very computationally expensive one) to perform hierarchical clustering because they do take a considerable amount of time to run.
  - **DBSCAN Clustering:**
    ● Find the best 'eps' using the elbow-knee method.
    ● You can take a smaller sample size for this as well.
- step 4: Summarize each cluster by manually observing few points from each cluster.
- step 5: You need to plot the word cloud with essay text for each cluster for each of algorithms mentioned in step 3.

2.4 Dimensionality Reduction on the selected features

In [31]:

```
# https://stats.stackexchange.com/questions/341332/how-to-scale-for-selectkbest-for-feature-select
ion
X_Train_K = SelectKBest(f_classif, k=5000).fit(X_Train, Y)
X_Train_new = X_Train_K.transform(X_Train)
print(X_Train_new.shape)
```

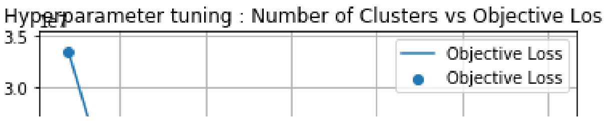
```
(12017, 5000)
```

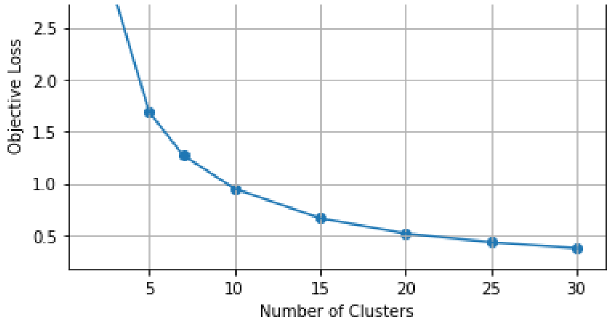
2.5 Apply Kmeans

In [31]:

```
%%time
params=[2,5,7,10,15,20,25,30]
loss = []
for i in tqdm(params):
 kmeans = KMeans(n_clusters=i, random_state=0, n_jobs=-1).fit(X_Train_new)
 loss.append(kmeans.inertia_)


# Performance of model on data for each hyper parameter.
plt.plot(params, loss, label='Objective Loss')
plt.gca()
plt.scatter(params, loss, label='Objective Loss')
plt.legend()
plt.xlabel("Number of Clusters")
plt.ylabel("Objective Loss")
plt.title("Hyperparameter tuning : Number of Clusters vs Objective Loss")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [2:11:31<00:00, 1186.17s/it]
```



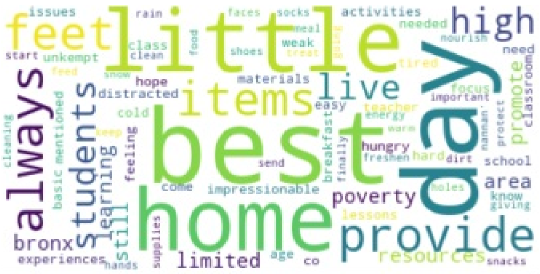


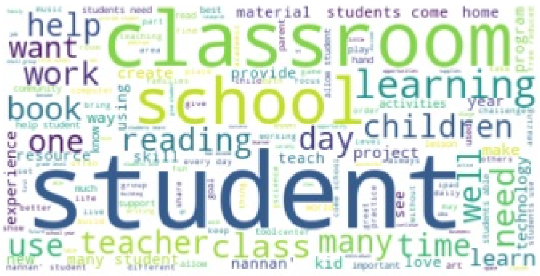
```
CPU times: user 1.25 s, sys: 336 ms, total: 1.58 s
Wall time: 2h 11min 31s
```

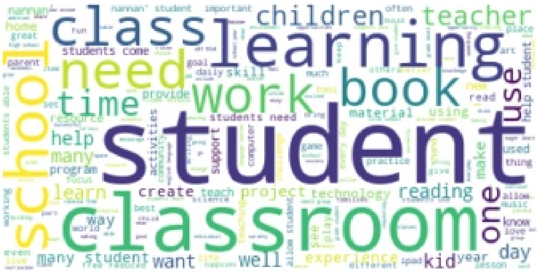
From the above elbow curve the knee point can be observed at 5 and hence the number of clusters taken is 5.

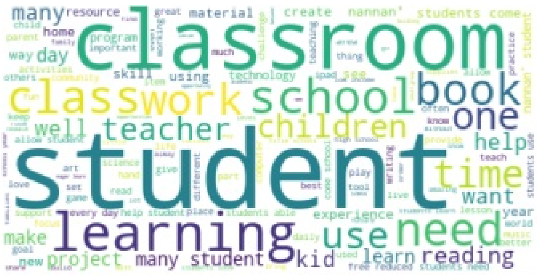
In [32]:

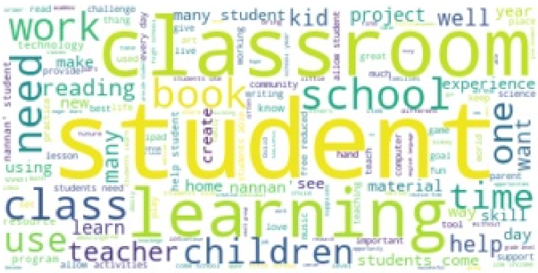
```
k_opt=5
kmeans_opt = KMeans(n_clusters=k_opt, random_state=0, n_jobs=-1).fit(X_Train_new)

pred = kmeans_opt.predict(X_Train_new)

def Cluster_creation(lbl,i_val):
  c = []
  c.append(essays[i])
  return c

essays = X_Tr_Es_DS['essay'].values
stopwords = set(STOPWORDS)
Cluster=[]
for j in range(k_opt):
  for i in range(kmeans_opt.labels_.shape[0]):
    if kmeans_opt.labels_[i] == j:
      c_val=Cluster_creation(kmeans_opt.labels_,i)
      Cluster.append(c_val)
      if(i == j):
        break;
  words=''
  for i in Cluster:
    words+=str(i)

  wordcloud = WordCloud(background_color ='white', stopwords = stopwords, repeat=False).generate(wo
rds)
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```









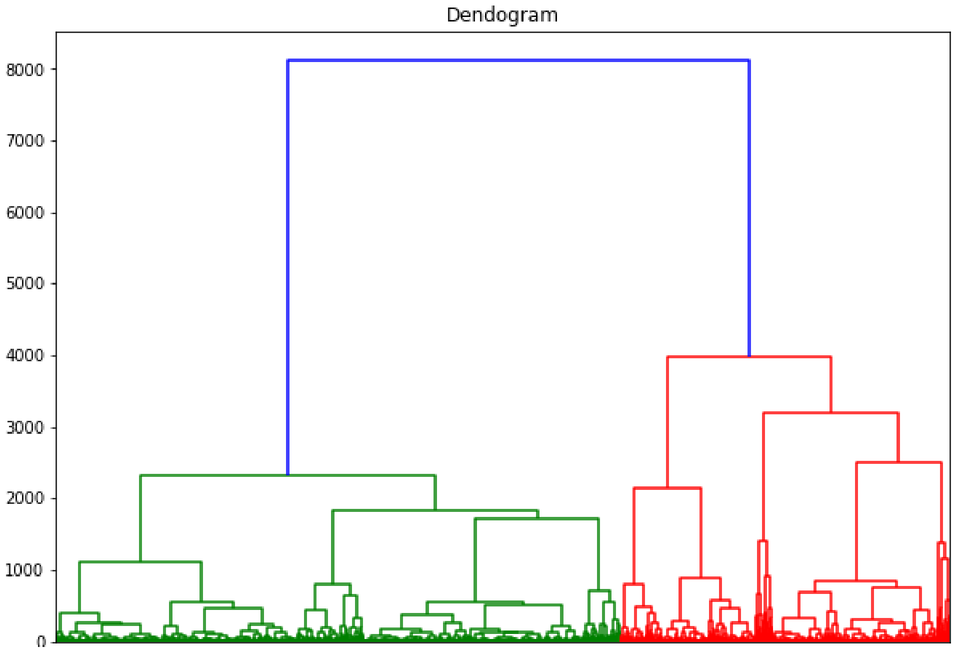


2.6 Apply AgglomerativeClustering

In [33]:

```
%%time
plt.figure(figsize=(10, 7))
plt.title("Dendogram")
dend = shc.dendrogram(shc.linkage(X_Train_new.todense(), method='ward'))
```

```
CPU times: user 6min 40s, sys: 1.1 s, total: 6min 41s
Wall time: 6min 41s
```

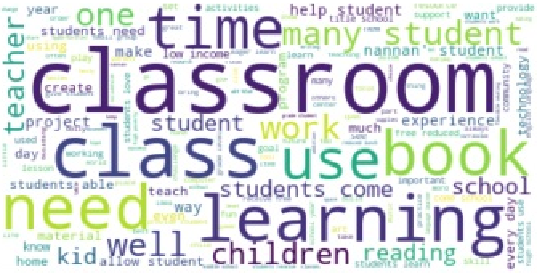


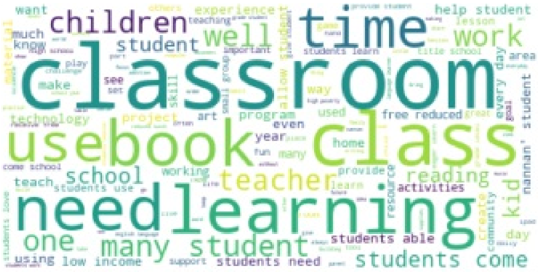
[redacted]

From the above Dendogram the two blue lines connecting the 2 clusters can be observed as the longest lines and hence the number of clusters taken without overfitting is 2.

In [34]:

```
a_opt=2
agg_opt = AgglomerativeClustering(n_clusters=a_opt, affinity='euclidean', linkage='ward').fit(X_Tra
in_new.todense())

def Cluster_creation(lbl,i_val):
  c = []
  c.append(essays[i])
  return c

essays = X_Tr_Es_DS['essay'].values
Cluster=[]
for j in range(a_opt):
  for i in range(agg_opt.labels_.shape[0]):
    if agg_opt.labels_[i] == j:
      c_val=Cluster_creation(agg_opt.labels_,i)
      Cluster.append(c_val)
      if(i == j):
        break;
  words=''
  for i in Cluster:
    words+=str(i)

  wordcloud = WordCloud(background_color ='white', stopwords = stopwords, repeat=False).generate(wo
rds)

  # Display the generated image:
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```





2.7 Apply DBSCAN

In [32]:

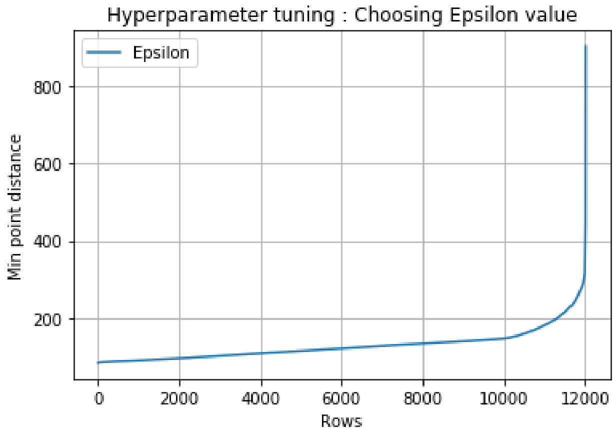
```
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html
X_Train_new_DB=X_Train_new.todense()

minPts = 10000
tree = KDTree(X_Train_new_DB)
idx = 0
mp_dist=[]
```

```
mp_dist = []
D_shape = X_Train_new_DB.shape[0]
for i in tqdm(range(D_shape)):
    dist, ind = tree.query(X_Train_new_DB[i], k=minPts)
    mp_dist.append(max(dist[0]))
mp_dist.sort()
#print(mp_dist)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 12017/12017 [24
:01<00:00,  8.47it/s]
```

In [33]:

```
%%time
params=range(D_shape)

# Performance of model on data for each hyper parameter.
plt.plot(params, mp_dist, label='Epsilon')
plt.gca()
plt.legend()
plt.xlabel("Rows")
plt.ylabel("Min point distance")
plt.title("Hyperparameter tuning : Choosing Epsilon value")
plt.grid()
plt.show()
```



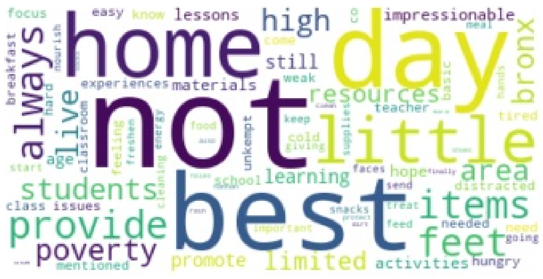
```
Wall time: 1.2 s
```

From the above elbow curve the knee point can be observed at the 200 (approximately). Hence it is selected as the epsilon value.

In [48]:

```
#https://stackoverflow.com/questions/41793963/dbscan-clustering-python-cluster-words
dbscan_opt = DBSCAN(eps=200, min_samples=minPts, n_jobs=-1).fit(X_Train_new)
labels=dbscan_opt.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print('Estimated number of clusters: %d' % n_clusters_)

def Cluster_creation(lbl,i_val):
  c = []
  c.append(essays[i])
  return c

essays = X_Tr_Es_DS['essay'].values
Cluster=[]
for j in range(n_clusters_):
  for i in range(dbscan_opt.labels_.shape[0]):
    if dbscan_opt.labels_[i] == j:
      c_val=Cluster_creation(dbscan_opt.labels_,i)
      Cluster.append(c_val)
      if(i == j):
        break;
  words=''
  for i in Cluster:
    words+=str(i)
  wordcloud = WordCloud(background_color ='white', stopwords = stopwords, repeat=False).generate(wo
rds)
  # Display the generated image:
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```

```
Estimated number of clusters: 1
```



3. Conclusions

In [42]:

```
from prettytable import PrettyTable
pt = PrettyTable()
pt.field_names = ["Vectorizer", "Model", "Number of Clusters", "Number of Features"]
pt.add_row(["TFIDF", "KMeans Clustering", "5", "5000"])
pt.add_row(["TFIDF", "Agglomerative Clustering", "2", "5000"])
pt.add_row(["TFIDF", "DBSCAN Clustering", "1", "5000"])
print(pt)
```

```
+------------+--------------------------+--------------------+--------------------+
| Vectorizer |          Model           | Number of Clusters | Number of Features |
+------------+--------------------------+--------------------+--------------------+
|   TFIDF    |    KMeans Clustering     |         5          |        5000        |
|   TFIDF    | Agglomerative Clustering |         2          |        5000        |
|   TFIDF    |    DBSCAN Clustering     |         1          |        5000        |
+------------+--------------------------+--------------------+--------------------+
```

Summary

"KMeans Clustering" has high space and time complexity when compared to "Agglomerative Clustering" and "DBSCAN Clustering" and "KMeans Clustering" tries to segregating the points into fine tuned clusters which may result in overfitting with test data if not careful.