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](https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **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 [1]:

```
%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

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
```

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv',nrows=25000)
resource_data = pd.read_csv('resources.csv')
```

In [3]:

```
project_data.shape
```

Out[3]:

```
(25000, 17)
```

In [4]:

```
project_data['project_is_approved'].value_counts()
```

Out[4]:

```
1    21184
0     3816
Name: project_is_approved, dtype: int64
```

In [5]:

```
resource_data.shape
```

Out[5]:

```
(1541272, 4)
```

In [6]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (25000, 17)
--------------------------------------------------
```

```
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [7]:

```
# 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[7]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Grades PreK-2	L

In [8]:

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

Out[8]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

1.2 preprocessing of `project_subject_categories`

In [9]:

```
print(project_data['project_subject_categories'].head(5))
```

```
473                       Applied Learning
23374                  Literacy & Language
7176      Math & Science, Applied Learning
5145                   Literacy & Language
2521                   Literacy & Language
Name: project_subject_categories, dtype: object
```

In [10]:

```
catogories = list(project_data['project_subject_categories'].values)
# 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
cat_list = []
for i in catogories:
    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())

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

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

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

In [11]:

```
print(project_data['clean_categories'].head(5))
```

```
473                      AppliedLearning
23374                  Literacy_Language
7176      Math_Science AppliedLearning
5145                   Literacy_Language
2521                   Literacy_Language
Name: clean_categories, dtype: object
```

1.3 preprocessing of `project_subject_subcategories`

In [12]:

```
print(project_data['project_subject_subcategories'].head(5))
```

```
473                         Early Development
23374                           ESL, Literacy
7176      Applied Sciences, Early Development
5145                                 Literacy
2521           Literacy, Literature & Writing
Name: project_subject_subcategories, dtype: object
```

In [13]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# 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_cat_list = []
for i in sub_catogories:
    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())

project_data['clean_subcategories'] = sub_cat_list
project_data.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 project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

In [14]:

```
print(project_data['clean_subcategories'].head(5))
```

```
473                        EarlyDevelopment
23374                          ESL Literacy
7176      AppliedSciences EarlyDevelopment
5145                               Literacy
2521            Literacy Literature_Writing
Name: clean_subcategories, dtype: object
```

1.4 preprocessing of `school_state`

In [15]:

```
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())

state_dict = dict(my_counter)
sorted_state_dict = dict(sorted(state_dict.items(), key=lambda kv: kv[1]))
```

In [16]:

```
print(project_data['school_state'].head(5))
```

```
473      GA
23374    CA
7176     OH
5145     CA
2521     NJ
Name: school_state, dtype: object
```

1.5 preprocessing of `project_grade_category`

In [17]:

```
preproc = []
# tqdm is for printing the status bar
for sent in project_data['project_grade_category']:
    sent = sent.replace('Grades ', '')
    sent = sent.replace('PreK-2', 'PreKto2')
    sent = sent.replace('3-5', '3to5')
    sent = sent.replace('6-8', '6to8')
    sent = sent.replace('9-12', '9to12')
    preproc.append(sent)
project_data['project_grade_category']=preproc
```

In [18]:

```
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.split())

grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))
```

In [19]:

```
print(project_data['project_grade_category'].head(5))
```

```
473      PreKto2
23374    PreKto2
7176     PreKto2
5145        3to5
2521     PreKto2
Name: project_grade_category, dtype: object
```

1.6 preprocessing of `teacher_prefix`

In [20]:

```
print(type(project_data['teacher_prefix']))
```

```
<class 'pandas.core.series.Series'>
```

In [21]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].astype(str)
preproc = []
# tqdm is for printing the status bar
for sent in project_data['teacher_prefix']:
    sent = sent.replace('Mr.', 'Mr')
    sent = sent.replace('Mrs.', 'Mrs')
    sent = sent.replace('Dr.', 'Dr')
    sent = sent.replace('Ms.', 'Ms')
    sent = sent.replace('nan','Mr')
    preproc.append(sent)
project_data['teacher_prefix']=preproc
```

In [22]:

```
#['Teacher', 'Mrs.', 'Dr.', 'Mr.', 'Ms.']
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('')
my_counter = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter.update(word.split())

teacher_dict = dict(my_counter)
sorted_teacher_dict = dict(sorted(teacher_dict.items(), key=lambda kv: kv[1]))
```

In [23]:

```
print(project_data['teacher_prefix'].value_counts())
```

```
Mrs        13046
Ms          9019
Mr          2392
Teacher      543
```

```
Name: teacher_prefix, dtype: int64
```

1.3 Preprocessing of Essays

In [24]:

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

In [25]:

```
# 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 [26]:

```
# 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 [27]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
```

```
# tqdm is for printing the status bar
for sentance in tqdm(project_data['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.append(sent.lower().strip())
```

```
100%|██████████| 25000/25000 [00:22<00:00, 1094.46it/s]
```

In [28]:

```
# after preprocesing
preprocessed_essays[2000]
```

Out[28]:

```
'students hardworking dedicated students come school every day spite daily challenges face live hi
gh poverty neighborhood every student receives free breakfast lunch school surrounding
neighborhood gang infested often violent not deter students coming school seeing named malala stud
ents came back school new found appreciation free public education students read malala aloud toge
ther classmates several weeks read book use kindle fires research project education lack thereof c
ountries united states choose country not provide free public education children country limits ed
ucation boys may also choose research syrian refugee crisis affecting children involved keeping sc
hool research done go computer lab type essay reflecting research goal students gain deep understa
nding important education privileged country get come school free nannan'
```

In [29]:

```
project_data['essay']=preprocessed_essays
```

1.4 Preprocessing of `project_title`

In [30]:

```
# similarly you can preprocess the titles also
preprocessed_titles = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_title'].values):
    sent = decontracted(sentence)
    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.append(sent.lower().strip())
```

```
100%|██████████| 25000/25000 [00:00<00:00, 27092.48it/s]
```

In [31]:

```
project_data['project_title']=preprocessed_titles
```

1.5 Preparing data for models

In [32]:

```
project_data.columns
```

Out[32]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay'],
      dtype='object')
```

we are going to consider

- school_state : categorical data
- clean_categories : categorical data
- clean_subcategories : categorical data
- project_grade_category : categorical data
- teacher_prefix : categorical data

- project_title : text data
- text : text data
- project_resource_summary: text data (optinal)

- quantity : numerical (optinal)
- teacher_number_of_previously_posted_projects : numerical
- price : numerical

In [33]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f,encoding = "ISO-8859-1")
    glove_words =  set(model.keys())
```

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.

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

Clustering

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

In [34]:

```
project_data_new=project_data.copy()
```

In [35]:

```
y_train = project_data['project_is_approved']
```

In [36]:

```
print(y_train.shape)
```

```
(25000,)
```

In [37]:

```
project_data.drop(['project_is_approved'],axis=1,inplace=True)
```

In [38]:

```
X=project_data
print(X.shape)
```

```
(25000, 17)
```

2.3 Make Data Model Ready: encoding numerical and categorical features

Vectorizing Numerical features

In [39]:

```
features=[]
```

In [40]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
```

In [41]:

```
price_data.head(5)
```

Out[41]:

	id	quantity	price
0	p000001	7	459.56
1	p000002	21	515.89
2	p000003	4	298.97
3	p000004	98	1113.69
4	p000005	8	485.99

In [42]:

```
X_train=pd.merge(X,price_data,on='id',how='left')
```

In [43]:

```
X_train=X_train.fillna(0)
```

Normalizing the numerical features: Price

Normalizing the numerical features: Price

In [44]:

```
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))
features += ['price']
print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print("="*100)
```

```
After vectorizations
(25000, 1) (25000,)
====================================================================================================
```

Normalizing the numerical features: Number of previously posted projects

In [45]:

```
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_train_project_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))
features += ['teacher_number_of_previously_posted_projects']
print("After vectorizations")
print(X_train_project_norm.shape, y_train.shape)
print("="*100)
```

```
After vectorizations
(25000, 1) (25000,)
====================================================================================================
```

Vectorizing Categorical features

- school_state : categorical data
- clean_categories : categorical data
- clean_subcategories : categorical data
- project_grade_category : categorical data
- teacher_prefix : categorical data

Vectorizing Categorical features: project grade category

In [46]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

In [47]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.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 = vectorizer.transform(X_train['project_grade_category'].values)
features += vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(25000, 4) (25000,)
['PreKto2', '3to5', '6to8', '9to12']
====================================================================================================
```

Vectorizing Categorical features: teacher prefix

In [48]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_dict.keys()), lowercase=False, binary=
True)
vectorizer.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 = vectorizer.transform(X_train['teacher_prefix'].values)
features += vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(25000, 4) (25000,)
['Mrs', 'Teacher', 'Mr', 'Ms']
====================================================================================================
```

In [49]:

```
type(vectorizer.get_feature_names())
```

Out[49]:

```
list
```

Vectorizing Categorical features: school state

In [50]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.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 = vectorizer.transform(X_train['school_state'].values)
features += vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(25000, 51) (25000,)
['AZ', 'WV', 'PA', 'DE', 'MD', 'UT', 'IN', 'ID', 'MT', 'KS', 'CT', 'FL', 'MO', 'IL', 'KY', 'NJ', 'N
M', 'VA', 'MS', 'MA', 'CO', 'OH', 'CA', 'DC', 'TN', 'GA', 'NC', 'IA', 'WI', 'OK', 'SC', 'SD', 'ND',
'AR', 'NY', 'ME', 'OR', 'AL', 'RI', 'VT', 'MI', 'LA', 'NV', 'MN', 'NH', 'NE', 'AK', 'HI', 'WY', 'WA
', 'TX']
====================================================================================================
```

Vectorizing Categorical features: clean categories

In [51]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
vectorizer.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_cat_ohe = vectorizer.transform(X_train['clean_categories'].values)
features += vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_cat_ohe.shape, y_train.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(25000, 9) (25000,)
['Health_Sports', 'Literacy_Language', 'Math_Science', 'History_Civics', 'AppliedLearning',
'Warmth', 'Care_Hunger', 'SpecialNeeds', 'Music_Arts']
====================================================================================================
```

Vectorizing Categorical features: clean subcategories

In [52]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.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_sub_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
features += vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_sub_ohe.shape, y_train.shape)
print(vectorizer.get_feature_names())
print("="*100);
```

```
After vectorizations
(25000, 30) (25000,)
['Civics_Government', 'Other', 'NutritionEducation', 'Care_Hunger', 'CommunityService', 'Music', '
Health_Wellness', 'FinancialLiteracy', 'EarlyDevelopment', 'CharacterEducation', 'Literacy', 'Extr
acurricular', 'VisualArts', 'TeamSports', 'Literature_Writing', 'Economics', 'ParentInvolvement',
'PerformingArts', 'SpecialNeeds', 'Mathematics', 'EnvironmentalScience', 'Health_LifeScience', 'So
cialSciences', 'History_Geography', 'ForeignLanguages', 'AppliedSciences', 'ESL', 'Warmth',
'College_CareerPrep', 'Gym_Fitness']
====================================================================================================
```

2.2 Make Data Model Ready: encoding eassay, and project_title

In [53]:

```
features_tfidf = features
```

Encoding of Text Data

In [54]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

TFIDF of Essay

In [55]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
```

In [56]:

```
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data
```

Out[56]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

In [57]:

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
```

In [58]:

```
features_tfidf += vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print("="*100)
```

```
After vectorizations
(25000, 5000) (25000,)
====================================================================================================
```

TFIDF of Title

In [59]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
```

In [60]:

```
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train data
```

Out[60]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

In [61]:

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['project_title'].values)
```

In [62]:

```
print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print("="*100)
```

```
After vectorizations
(25000, 2199) (25000,)
====================================================================================================
```

Creating Data Matrix

In [63]:

```
# Please write all the code with proper documentation


# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf,X_train_title_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_t
rain_grade_ohe,X_train_cat_ohe,X_train_sub_ohe, X_train_price_norm,X_train_project_norm)).tocsr()

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

```
Final Data matrix
(25000, 7299) (25000,)
====================================================================================================
```

In [64]:

```
from sklearn.feature_selection import SelectKBest,chi2,f_classif
best_feature=SelectKBest(score_func=f_classif,k=5000)
```

In [65]:

```
best_feature.fit(X_tr,y_train)
```

Out[65]:

```
SelectKBest(k=5000, score_func=<function f_classif at 0x7f2cedfcd7b8>)
```

In [66]:

```
X_tr=best_feature.transform(X_tr)
```

In [67]:

```
X_tr.shape
```

Out[67]:

```
(25000, 5000)
```

KMeans Clustering

Hyperparameter Tuning

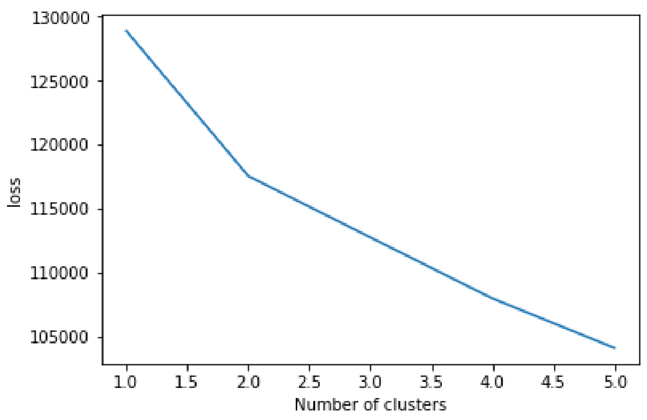
In [68]:

```
from sklearn.cluster import KMeans
```

In [69]:

```
loss = []
for k in tqdm(range(1,6)):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X_tr)
    loss.append(kmeans.inertia_)
plt.figure()
plt.plot(range(1,6),loss)
plt.xlabel("Number of clusters")
plt.ylabel("loss")
plt.show()
```

```
100%|██████████| 5/5 [12:58<00:00, 165.12s/it]
```



In [70]:

```
best_k=2
```

In [71]:

```
kmeans = KMeans(n_clusters=best_k)
kmeans.fit(X_tr)
```

Out[71]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [72]:

```
cluster_1=[]
cluster_2=[]
```

In [73]:

```
for i in range(X_tr.shape[0]):
    if kmeans.labels_[i] == 0:
        cluster_1.append(project_data_new.iloc[i])
    elif kmeans.labels_[i] == 1:
        cluster_2.append(project_data_new.iloc[i])
```

In [74]:

```
pos=0
neg=0
total_points=len(cluster_1)
for i in cluster_1:
    if i['project_is_approved']==1:
        pos=pos+1
    else:
        neg=neg+1
print("Projects Approved %age in cluster 1 ",(pos/total_points*100))
print("Projects Not Approved %age in cluster 1 ",(neg/total_points*100))
```

```
Projects Approved %age in cluster 1  83.13160126435896
Projects Not Approved %age in cluster 1  16.868398735641048
```

In [75]:

```
pos=0
neg=0
total_points=len(cluster_2)
for i in cluster_2:
    if i['project_is_approved']==1:
        pos=pos+1
    else:
        neg=neg+1
print("Projects Approved %age in cluster 2 ",(pos/total_points*100))
print("Projects Not Approved %age in cluster 2 ",(neg/total_points*100))
```

```
Projects Approved %age in cluster 2  86.46604040236096
Projects Not Approved %age in cluster 2  13.53395959763904
```

Word Cloud

Cluster 1

In [76]:

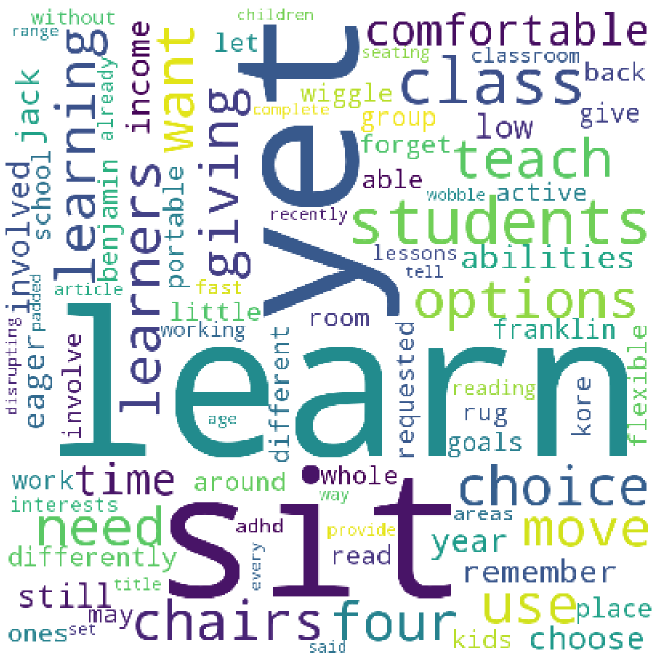
```
from wordcloud import WordCloud

essay_wc=cluster_1[0]['essay']


wordcloud = WordCloud(width = 800, height = 800, background_color ='white', min_font_size = 10).gen
erate(essay_wc)


plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



Cluster 2

In [77]:

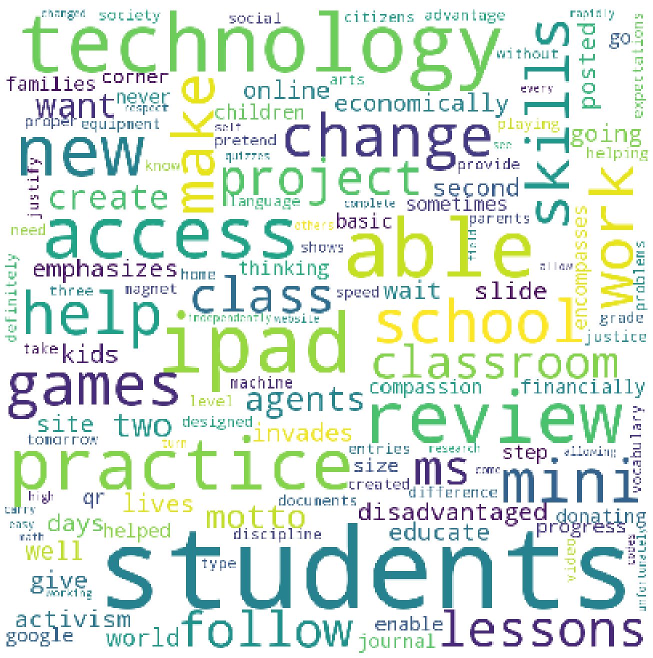
```
from wordcloud import WordCloud

essay_wc=cluster_2[0]['essay']


wordcloud = WordCloud(width = 800, height = 800, background_color ='white', min_font_size = 10).gen
erate(essay_wc)
```

```
plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



Most Frequent Words in Both the Clusters

Cluster 1

In [78]:

```
from collections import Counter
preprocessed=[]
data=[]
for i in range(len(cluster_1)):
    data.append(cluster_1[i]['essay'])
for sentance in (data):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed.append(sent.lower().strip())
preprocessed=" ".join(preprocessed)

split_words = preprocessed.split()

Counter = Counter(split_words)

most_frequent = Counter.most_common(10)
print(most_frequent)
```

```
[('students', 95183), ('school', 31498), ('learning', 20345), ('classroom', 18053), ('not',
15458), ('learn', 15270), ('help', 14424), ('nannan', 12442), ('many', 12261), ('need', 11290)]
```

Cluster 2

In [79]:

```
from collections import Counter
preprocessed=[]
```

```
data=[]
for i in range(len(cluster_2)):
    data.append(cluster_2[i]['essay'])
for sentance in (data):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed.append(sent.lower().strip())
preprocessed=" ".join(preprocessed)

split_words = preprocessed.split()

Counter = Counter(split_words)

most_frequent = Counter.most_common(10)
print(most_frequent)
```

```
[('students', 86001), ('school', 26776), ('classroom', 20722), ('learning', 20261), ('reading',
18175), ('not', 14630), ('help', 13668), ('learn', 13542), ('books', 12515), ('many', 12035)]
```

Agglomerative Clustering

In [80]:

```
from sklearn.cluster import AgglomerativeClustering
model = AgglomerativeClustering(n_clusters = 2)
model.fit(X_tr.toarray())# To convert sparse matrix to dense: x.to_array()
```

Out[80]:

```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=2,
            pooling_func='deprecated')
```

In [81]:

```
cluster_1=[]
cluster_2=[]
```

In [83]:

```
for i in range(X_tr.shape[0]):
    if model.labels_[i] == 0:
        cluster_1.append(project_data_new.iloc[i])
    elif model.labels_[i] == 1:
        cluster_2.append(project_data_new.iloc[i])
```

In [84]:

```
pos=0
neg=0
total_points=len(cluster_1)
for i in cluster_1:
    if i['project_is_approved']==1:
        pos=pos+1
    else:
        neg=neg+1
print("Projects Approved %age in cluster 1 ",(pos/total_points*100))
print("Projects Not Approved %age in cluster 1 ",(neg/total_points*100))
```

```
Projects Approved %age in cluster 1  83.06689025526364
Projects Not Approved %age in cluster 1  16.933109744736353
```

In [85]:

```
pos=0
```

```
pos=0
neg=0
total_points=len(cluster_2)
for i in cluster_2:
    if i['project_is_approved']==1:
        pos=pos+1
    else:
        neg=neg+1
print("Projects Approved %age in cluster 2 ",(pos/total_points*100))
print("Projects Not Approved %age in cluster 2 ",(neg/total_points*100))
```

```
Projects Approved %age in cluster 2  86.66954457155191
Projects Not Approved %age in cluster 2  13.33045542844809
```

Word Cloud

Cluster 1

In [100]:

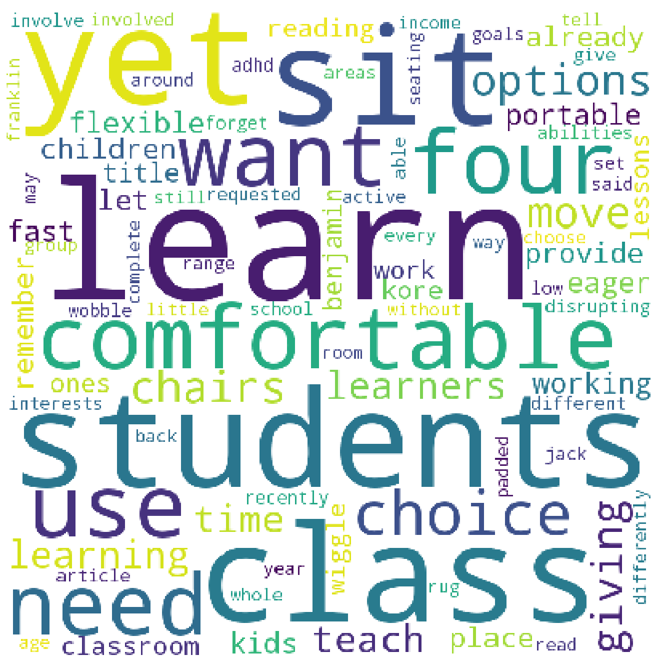
```
from wordcloud import WordCloud

essay_wc=cluster_1[0]['essay']


wordcloud = WordCloud(width = 800, height = 800, background_color ='white', min_font_size = 10).gen
erate(essay_wc)


plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



Cluster 2

In [99]:

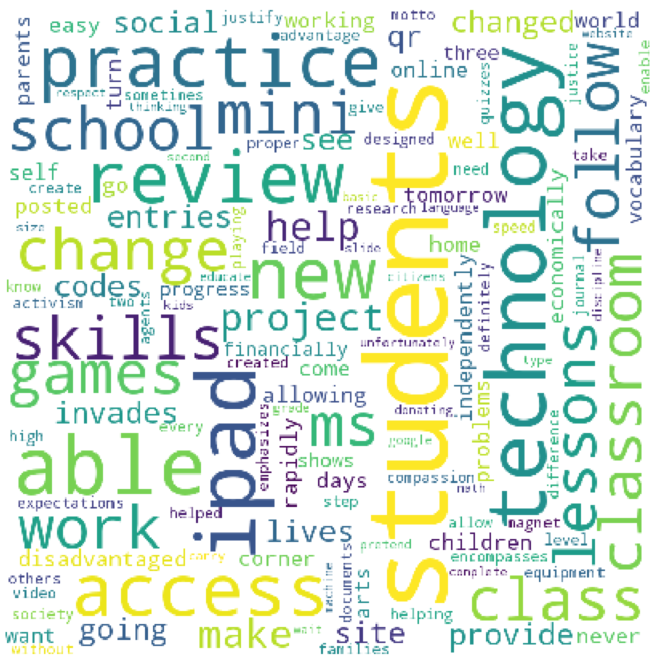
```
from wordcloud import WordCloud

essay_wc=cluster_2[0]['essay']
```

```
wordcloud = WordCloud(width = 800, height = 800, background_color ='white', min_font_size = 10).gen
erate(essay_wc)


plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



Most Frequent Words in Both the Clusters

Cluster 1

In [90]:

```
from collections import Counter
preprocessed=[]
data=[]
for i in range(len(cluster_1)):
    data.append(cluster_1[i]['essay'])
for sentance in (data):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed.append(sent.lower().strip())
preprocessed=" ".join(preprocessed)

split_words = preprocessed.split()

Counter = Counter(split_words)

most_frequent = Counter.most_common(10)
print(most_frequent)
```

```
[('students', 196904), ('school', 64375), ('learning', 42684), ('classroom', 38093), ('learn',
31751), ('not', 31712), ('help', 30003), ('nannan', 25718), ('many', 25418), ('need', 23329)]
```

Cluster 2

In [93]:

```
from collections import Counter
preprocessed=[]
data=[]
for i in range(len(cluster_2)):
    data.append(cluster_2[i]['essay'])
for sentance in (data):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed.append(sent.lower().strip())
preprocessed=" ".join(preprocessed)

split_words = preprocessed.split()

Counter = Counter(split_words)

most_frequent = Counter.most_common(10)
print(most_frequent)
```

```
[('students', 165464), ('school', 52173), ('classroom', 39457), ('learning', 38528), ('reading', 3
4198), ('not', 28464), ('help', 26181), ('learn', 25873), ('books', 23760), ('many', 23174)]
```

DBSCAN Clustering

In [80]:

```
X_tr=X_tr[:1000]
```

In [81]:

```
project_data_new=project_data_new[:1000]
```

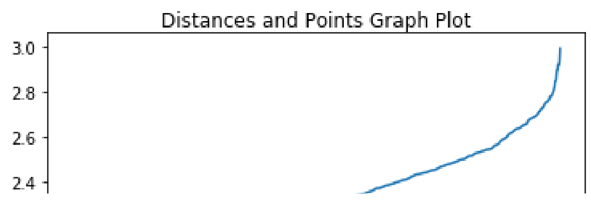
In [87]:

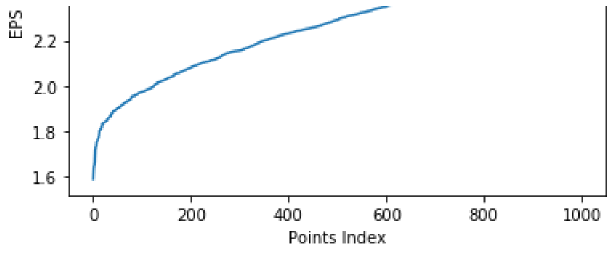
```
#https://datascience.stackexchange.com/questions/10162/knn-distance-plot-for-determining-eps-of-db
scan
#https://stackoverflow.com/questions/12893492/choosing-eps-and-minpts-for-dbscan-r
#Some references from where help was taken
min_pts=10
kth_dist=[]
dist = []
for row in tqdm(X_tr):
    row_dist = np.sort(np.sum((X_tr.toarray()-row.toarray())**2,axis=1),axis=None)
    dist.append(row_dist[min_pts])
kth_dist=np.sort(np.sqrt(np.array(dist)))
points = [x for x in range(X_tr.shape[0])]
```

```
1000it [00:56, 17.32it/s]
```

In [88]:

```
plt.plot(points, kth_dist)
plt.xlabel('Points Index')
plt.ylabel('EPS')
plt.title('Distances and Points Graph Plot')
plt.show()
```





In [89]:

```
best_eps=2.70
```

In [91]:

```
from sklearn.cluster import DBSCAN
eps=best_eps
model = DBSCAN(eps=eps,min_samples=10)
model.fit(X_tr)
```

Out[91]:

```
DBSCAN(algorithm='auto', eps=2.7, leaf_size=30, metric='euclidean',
    metric_params=None, min_samples=10, n_jobs=None, p=None)
```

In [92]:

```
#https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html
labels=model.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
```

```
Estimated number of clusters: 1
Estimated number of noise points: 2
```

In [93]:

```
clusters_1=[]
```

In [103]:

```
for i in range(X_tr.shape[0]):
    if model.labels_[i] == 1:
        clusters_1.append(project_data_new.iloc[i])
```

In [104]:

```
pos=0
neg=0
total_points=len(clusters_1)
for i in clusters_1:
    if i['project_is_approved']==1:
        pos=pos+1
    else:
        neg=neg+1
print("Projects Approved %age in cluster 1 ",(pos/total_points*100))
print("Projects Not Approved %age in cluster 1 ",(neg/total_points*100))
```

```
Projects Approved %age in cluster 1  87.374749498998
Projects Not Approved %age in cluster 1  12.625250501002002
```

Word Cloud

Cluster 1

In [105]:

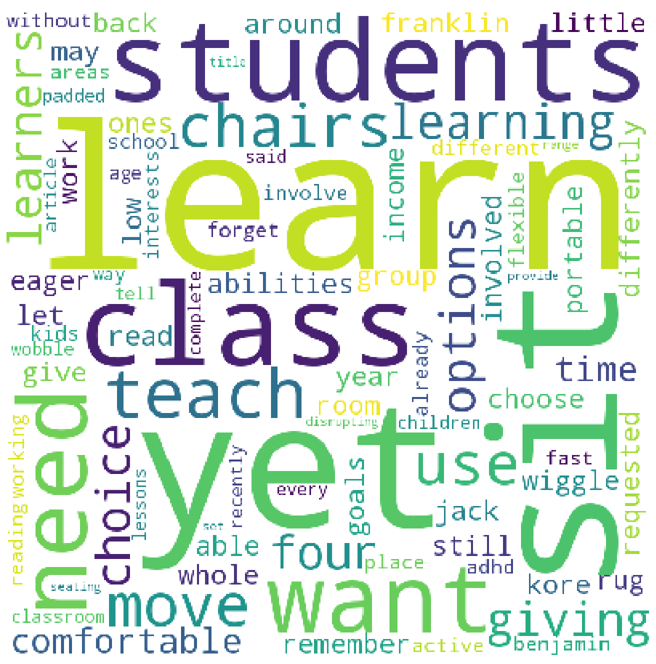
```
from wordcloud import WordCloud

essay_wc=clusters_1[0]['essay']


wordcloud = WordCloud(width = 800, height = 800, background_color ='white', min_font_size = 10).gen
erate(essay_wc)


plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



Most Frequent Words in the Cluster

Cluster 1

In [106]:

```
from collections import Counter
preprocessed=[]
data=[]
for i in range(len(clusters_1)):
    data.append(clusters_1[i]['essay'])
for sentance in (data):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed.append(sent.lower().strip())
preprocessed=" ".join(preprocessed)

split_words = preprocessed.split()

Counter = Counter(split_words)

most_frequent = Counter.most_common(10)
print(most_frequent)
```

```
[('students', 15380), ('school', 4786), ('learning', 3282), ('classroom', 3204), ('not', 2766), ('
help', 2442), ('learn', 2386), ('reading', 2164), ('use', 2046), ('work', 1932)]
```

Conclusion

In [107]:

```
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
x=PrettyTable()
x.field_names=["Vectorizer","Model","Cluster Count"]
x.add_row(["TFIDF","KMeans Tree",2])
x.add_row(["TFIDF","Agglomerative Clustering Tree",2])
x.add_row(["TFIDF","DBSCAN ",1])
print(x)
```

```
+------------+-------------------------------+---------------+
| Vectorizer |             Model             | Cluster Count |
+------------+-------------------------------+---------------+
|   TFIDF    |          KMeans Tree          |       2       |
|   TFIDF    | Agglomerative Clustering Tree |       2       |
|   TFIDF    |            DBSCAN             |       1       |
+------------+-------------------------------+---------------+
```