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*
`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_4:__ "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.

```
%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

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
"""
from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter"""
```

```
'\nfrom plotly import plotly\nimport plotly.offline as offline\nimport plotly.graph_objs as go\noffline.init_notebook_mode()\nfrom collections import Counter'
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%

Enter your authorization code:
··········
Mounted at /content/drive
```

1.1 Reading Data

```
project_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/train_data.csv')
resource_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/resources.csv')
```

```
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 (109248, 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']
```

```
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']
```

	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 Data Analysis

```
# PROVIDE CITATIONS TO YOUR CODE IF YOU TAKE IT FROM ANOTHER WEBSITE.
# https://matplotlib.org/gallery/pie_and_polar_charts/pie_and_donut_labels.html#sphx-glr-gallery-pie-and-polar-charts-pie-and-donut-labels-p


y_value_counts = project_data['project_is_approved'].value_counts()
print("Number of projects thar are approved for funding ", y_value_counts[1], ", (", (y_value_counts[1]/(y_value_counts[1]+y_value_counts[0]
print("Number of projects thar are not approved for funding ", y_value_counts[0], ", (", (y_value_counts[0]/(y_value_counts[1]+y_value_count

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(aspect="equal"))
recipe = ["Accepted", "Not Accepted"]

data = [y_value_counts[1], y_value_counts[0]]

wedges, texts = ax.pie(data, wedgeprops=dict(width=0.5), startangle=-40)

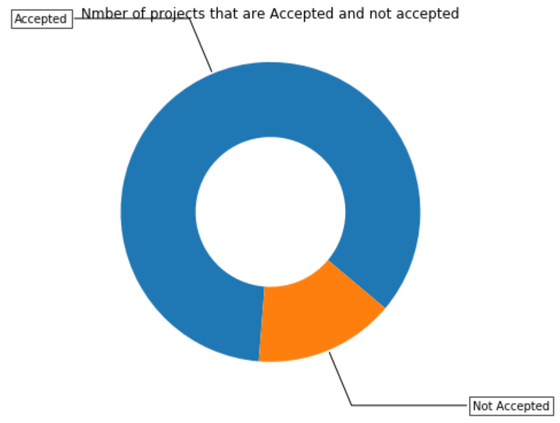
bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
kw = dict(xycoords='data', textcoords='data', arrowprops=dict(arrowstyle="-"),
          bbox=bbox_props, zorder=0, va="center")

for i, p in enumerate(wedges):
    ang = (p.theta2 - p.theta1)/2. + p.theta1
    y = np.sin(np.deg2rad(ang))
    x = np.cos(np.deg2rad(ang))
    horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
    connectionstyle = "angle,angleA=0,angleB={}".format(ang)
    kw["arrowprops"].update({"connectionstyle": connectionstyle})
    ax.annotate(recipe[i], xy=(x, y), xytext=(1.35*np.sign(x), 1.4*y),
                horizontalalignment=horizontalalignment, **kw)

ax.set_title("Nmber of projects that are Accepted and not accepted")

plt.show()
```

```
Number of projects thar are approved for funding  92706 , ( 84.85830404217927 %)
Number of projects thar are not approved for funding  16542 , ( 15.141695957820739 %)
```



1.2.1 Univariate Analysis: School State

```
# Pandas dataframe groupby count, mean: https://stackoverflow.com/a/19385591/4084039

temp = pd.DataFrame(project_data.groupby("school_state")["project_is_approved"].apply(np.mean)).reset_index()
# if you have data which contain only 0 and 1, then the mean = percentage (think about it)
temp.columns = ['state_code', 'num_proposals']

'''# How to plot US state heatmap: https://datascience.stackexchange.com/a/9620

scl = [[0.0, 'rgb(242,240,247)'],[0.2, 'rgb(218,218,235)'],[0.4, 'rgb(188,189,220)'],\
            [0.6, 'rgb(158,154,200)'],[0.8, 'rgb(117,107,177)'],[1.0, 'rgb(84,39,143)']]

data = [ dict(
        type='choropleth',
        colorscale = scl,
        autocolorscale = False,
        locations = temp['state_code'],
        z = temp['num_proposals'].astype(float),
        locationmode = 'USA-states',
        text = temp['state_code'],
        marker = dict(line = dict (color = 'rgb(255,255,255)',width = 2)),
        colorbar = dict(title = "% of pro")
    ) ]

layout = dict(
        title = 'Project Proposals % of Acceptance Rate by US States',
        geo = dict(
            scope='usa',
            projection=dict( type='albers usa' ),
            showlakes = True,
            lakecolor = 'rgb(255, 255, 255)',
        ),
    )

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='us-map-heat-map')
'''
```

```
'# How to plot US state heatmap: https://datascience.stackexchange.com/a/9620\n\nscl = [[0.0, \'rgb(242,240,247)\'],[0.2, \'rgb(218,218,235)\'],[0.4, \'rgb(188,189,220)\'],            [0.6, \'rgb(158,154,200)\'],[0.8, \'rgb(117,107,177)\'],[1.0, \'
```

```
# https://www.csi.cuny.edu/sites/default/files/pdf/administration/ops/2letterstabbrev.pdf
temp.sort_values(by=['num_proposals'], inplace=True)
print("States with lowest % approvals")
print(temp.head(5))
print('='*50)
print("States with highest % approvals")
print(temp.tail(5))
```

```
States with lowest % approvals
   state_code  num_proposals
46         VT       0.800000
7          DC       0.802326
43         TX       0.813142
26         MT       0.816327
18         LA       0.831245
==================================================
States with highest % approvals
   state_code  num_proposals
30         NH       0.873563
35         OH       0.875152
47         WA       0.876178
28         ND       0.888112
8          DE       0.897959
```

```
#stacked bar plots matplotlib: https://matplotlib.org/gallery/lines_bars_and_markers/bar_stacked.html
def stack_plot(data, xtick, col2='project_is_approved', col3='total'):
    ind = np.arange(data.shape[0])

    plt.figure(figsize=(20,5))
    p1 = plt.bar(ind, data[col3].values)
    p2 = plt.bar(ind, data[col2].values)

    plt.ylabel('Projects')
    plt.title('Number of projects aproved vs rejected')
    plt.xticks(ind, list(data[xtick].values))
    plt.legend((p1[0], p2[0]), ('total', 'accepted'))
    plt.show()


def univariate_barplots(data, col1, col2='project_is_approved', top=False):
    # Count number of zeros in dataframe python: https://stackoverflow.com/a/51540521/4084039
    temp = pd.DataFrame(project_data.groupby(col1)[col2].agg(lambda x: x.eq(1).sum())).reset_index()

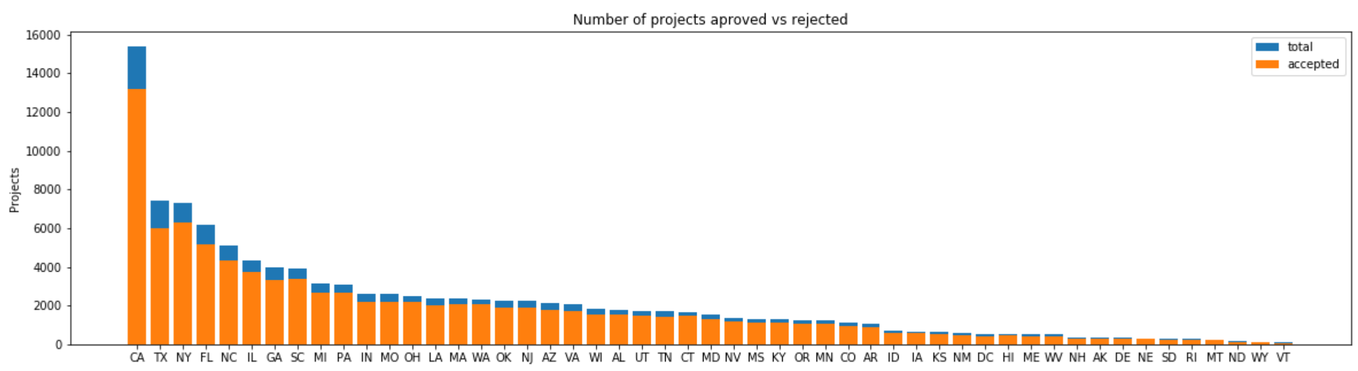
    # Pandas dataframe grouby count: https://stackoverflow.com/a/19385591/4084039
    temp['total'] = pd.DataFrame(project_data.groupby(col1)[col2].agg({'total':'count'})).reset_index()['total']
    temp['Avg'] = pd.DataFrame(project_data.groupby(col1)[col2].agg({'Avg':'mean'})).reset_index()['Avg']

    temp.sort_values(by=['total'],inplace=True, ascending=False)

    if top:
        temp = temp[0:top]

    stack_plot(temp, xtick=col1, col2=col2, col3='total')
    print(temp.head(5))
    print("="*50)
    print(temp.tail(5))


univariate_barplots(project_data, 'school_state', 'project_is_approved', False)
```




```
   school_state  project_is_approved  total       Avg
4            CA                13205  15388  0.858136
43           TX                 6014   7396  0.813142
34           NY                 6291   7318  0.859661
9            FL                 5144   6185  0.831690
27           NC                 4353   5091  0.855038
==================================================
   school_state  project_is_approved  total       Avg
39           RI                  243    285  0.852632
26           MT                  200    245  0.816327
28           ND                  127    143  0.888112
50           WY                   82     98  0.836735
46           VT                   64     80  0.800000
```

SUMMARY: Every state has greater than 80% success rate in approval

1.2.2 Univariate Analysis: teacher_prefix

↳ 1 cell hidden

1.2.3 Univariate Analysis: project_grade_category

↳ 1 cell hidden

1.2.4 Univariate Analysis: project_subject_categories

↳ 6 cells hidden

1.2.5 Univariate Analysis: project_subject_subcategories

↳ 6 cells hidden

1.2.6 Univariate Analysis: Text features (Title)

↳ 4 cells hidden

1.2.7 Univariate Analysis: Text features (Project Essay's)

↳ 4 cells hidden

1.2.8 Univariate Analysis: Cost per project

↳ 13 cells hidden

1.3 Text preprocessing

1.3.1 Essay Text

↳ 12 cells hidden

1.3.3 teacher_prefix

```
#upperCase to lowercase
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.lower()
#removing puntuation from the column
#https://stackoverflow.com/questions/39782418/remove-punctuations-in-pandas
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.replace('[^\w\s]','')
```

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

```
mrs        57269
ms         38955
mr         10648
teacher     2360
dr            13
Name: teacher_prefix, dtype: int64
```

1.3.4 project_grade_category

```
#upperCase to lowercase
project_data['project_grade_category'] = project_data['project_grade_category'].str.lower()
#removing puntuation from the column
#https://stackoverflow.com/questions/39782418/remove-punctuations-in-pandas
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace('[^\w\s]','_')
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace(' ','_')

project_data.project_grade_category.value_counts()
```

```
grades_prek_2    44225
grades_3_5       37137
grades_6_8       16923
grades_9_12      10963
Name: project_grade_category, dtype: int64
```

1.3.5 school_state

```
#upperCase to lowercase
project_data['school_state'] = project_data['school_state'].str.lower()
project_data.school_state.value_counts()
```

```
ca    15388
tx     7396
ny     7318
fl     6185
nc     5091
il     4350
ga     3963
sc     3936
mi     3161
pa     3109
in     2620
mo     2576
oh     2467
la     2394
ma     2389
wa     2334
ok     2276
nj     2237
az     2147
va     2045
wi     1827
al     1762
ut     1731
tn     1688
ct     1663
md     1514
nv     1367
ms     1323
ky     1304
or     1242
mn     1208
co     1111
ar     1049
id      693
ia      666
ks      634
nm      557
dc      516
hi      507
me      505
wv      503
nh      348
ak      345
de      343
ne      309
sd      300
ri      285
mt      245
nd      143
wy       98
vt       80
Name: school_state, dtype: int64
```

```
#upperCase to lowercase
project_data['clean_categories'] = project_data['clean_categories'].str.lower()
project_data.clean_categories.value_counts()
```

```
literacy_language                       23655
math_science                            17072
literacy_language math_science          14636
health_sports                           10177
music_arts                               5180
specialneeds                             4226
literacy_language specialneeds           3961
appliedlearning                          3771
math_science literacy_language           2289
appliedlearning literacy_language        2191
history_civics                           1851
math_science specialneeds                1840
literacy_language music_arts             1757
math_science music_arts                  1642
appliedlearning specialneeds             1467
history_civics literacy_language         1421
health_sports specialneeds               1391
warmth care_hunger                       1309
math_science appliedlearning             1220
appliedlearning math_science             1052
literacy_language history_civics          809
health_sports literacy_language           803
appliedlearning music_arts                758
math_science history_civics               652
literacy_language appliedlearning         636
appliedlearning health_sports             608
math_science health_sports                414
history_civics math_science               322
history_civics music_arts                 312
specialneeds music_arts                   302
health_sports math_science                271
history_civics specialneeds               252
health_sports appliedlearning             192
appliedlearning history_civics            178
health_sports music_arts                  155
music_arts specialneeds                   138
literacy_language health_sports            72
health_sports history_civics               43
history_civics appliedlearning             42
specialneeds health_sports                 42
health_sports warmth care_hunger           23
specialneeds warmth care_hunger            23
music_arts health_sports                   19
music_arts history_civics                  18
history_civics health_sports               13
math_science warmth care_hunger            11
music_arts appliedlearning                 10
appliedlearning warmth care_hunger         10
literacy_language warmth care_hunger        9
music_arts warmth care_hunger               2
history_civics warmth care_hunger           1
Name: clean_categories, dtype: int64
```

```
project_data['clean_subcategories'] = project_data['clean_subcategories'].str.lower()
project_data.clean_subcategories.value_counts()
```

```
literacy                                9486
literacy mathematics                    8325
literature_writing mathematics          5923
literacy literature_writing             5571
mathematics                             5379
                                        ...
communityservice financialliteracy         1
esl economics                               1
esl teamsports                              1
civics_government foreignlanguages          1
economics foreignlanguages                  1
Name: clean_subcategories, Length: 401, dtype: int64
```

1. 4 Preparing data for models

```
project_data.columns
```

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', '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', 'price',
       'quantity'],
      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

- quantity : numerical

```
- teacher_number_of_previously_posted_projects : numerical
```

1.4.1 Vectorizing Categorical data

↳ 8 cells hidden

1.4.2 Vectorizing Text data

↳ 20 cells hidden

1.4.3 Vectorizing Numerical features

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()
price_scalar.fit(project_data['price'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
price_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1))
```

teacher_number_of_previously_posted_projects

```
previously_posted_scalar = StandardScaler()
previously_posted_scalar.fit(project_data['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)) # finding the mean and stand
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
previously_posted_standardized = previously_posted_scalar.transform(project_data['teacher_number_of_previously_posted_projects'].values.resh

previously_posted_standardized

price_standardized
```

1.4.4 Merging all the above features

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

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
print(previously_posted_standardized.shape)
#print(tfidf_w2v_vectors_project_title.shape)
#print(tfidf_w2v_vectors.shape)
#print(avg_w2v_vectors_pro_title.shape)
#print(avg_w2v_vectors.shape)

print(project_title_bow.shape)
print(text_tfidf.shape)
print(project_title_tfidf.shape)


# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)

project_data.columns

#BoW dataset
X_BoW = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized,previously_posted_standardized,
                project_grade_category_one_hot,state_one_hot ,project_title_bow,teacher_prefix_one_hot))
X_BoW.shape

#TFIDF dataset
X_TFIDF = hstack((categories_one_hot, sub_categories_one_hot, text_tfidf, price_standardized,previously_posted_standardized,
            project_grade_category_one_hot,state_one_hot ,project_title_tfidf,teacher_prefix_one_hot))
X_TFIDF.shape

#AVG W2V dataset
X_avg_w2v = hstack((categories_one_hot, sub_categories_one_hot, avg_w2v_vectors, price_standardized,previously_posted_standardized,

            project_grade_category_one_hot,state_one_hot ,avg_w2v_vectors_pro_title,teacher_prefix_one_hot))
X_avg_w2v.shape

#TFIDF W2V
X_tfidf_w2v = hstack((categories_one_hot, sub_categories_one_hot, tfidf_w2v_vectors, price_standardized,previously_posted_standardized,
                project_grade_category_one_hot,state_one_hot ,tfidf_w2v_vectors_project_title,teacher_prefix_one_hot))
X_tfidf_w2v.shape

y = project_data['project_is_approved']
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
2. **Hyper paramter tuning to find best K**
   - Find the best hyper parameter which results in the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
   - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points
4. **[Task-2]**
   - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features
   - ```
     from sklearn.datasets import load_digits
     from sklearn.feature_selection import SelectKBest, chi2
     X, y = load_digits(return_X_y=True)
     X.shape
     X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
     X_new.shape
     ========
     output:
     (1797, 64)
     (1797, 20)
     ```
   - Repeat the steps 2 and 3 on the data matrix after feature selection
5. **Conclusion**
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

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.

2. K Nearest Neighbor

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

import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
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

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

import pickle
from tqdm import tqdm
import os
from sklearn.neighbors import KNeighborsClassifier



from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%

Enter your authorization code:
··········
Mounted at /content/drive
```

```
data  = pd.read_csv('/content/drive/My Drive/Colab Notebooks/preprocessed_data.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109248 entries, 0 to 109247
Data columns (total 9 columns):
school_state                                    109248 non-null object
teacher_prefix                                  109248 non-null object
project_grade_category                          109248 non-null object
teacher_number_of_previously_posted_projects    109248 non-null int64
project_is_approved                             109248 non-null int64
clean_categories                                109248 non-null object
clean_subcategories                             109248 non-null object
essay                                           109248 non-null object
price                                           109248 non-null float64
dtypes: float64(1), int64(2), object(6)
memory usage: 7.5+ MB
```

```
project_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/train_data.csv')


# 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', 'each', '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', "doesn'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"]
```

```
# 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


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

```
100%|██████████| 109248/109248 [00:02<00:00, 43037.38it/s]
```

```
data['project_title'] = preprocessed_project_title


data.head(1)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_subcategories	essay	price	project_title
0	ca	mrs	grades_prek_2	53	1	math_science	appliedsciences health_lifescience	i fortunate enough use fairy tale stem kits cl...	725.05	educational support english learners home

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

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

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	essay	price	project_title
0	ca	mrs	grades_prek_2	53	math_science	appliedsciences health_lifescience	i fortunate enough use fairy tale stem kits cl...	725.05	educational support english learners home

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

school_state

```
# we use count vectorizer to convert the values into one hot encoded features

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['school_state'].values)

states_one_hot_X_train = vectorizer.transform(X_train['school_state'].values)
states_one_hot_X_cv = vectorizer.transform(X_cv['school_state'].values)
states_one_hot_X_test = vectorizer.transform(X_test['school_state'].values)

print("="*50)
print("Shape of matrix after one hot encoding(school_state) ")
print("="*50)
print("X_train  :",states_one_hot_X_train.shape)
print("X_cv     :",states_one_hot_X_cv.shape)
print("X_test   :",states_one_hot_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Shape of matrix after one hot encoding(school_state)
==================================================
X_train  : (49041, 51)
X_cv     : (24155, 51)
X_test   : (36052, 51)
==================================================
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', '
==================================================
```

teacher_prefix

```
# we use count vectorizer to convert the values into one hot encoded features

vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['teacher_prefix'].values)

teacher_prefix_one_hot_X_train = vectorizer.transform(X_train['teacher_prefix'].values)
teacher_prefix_one_hot_X_cv = vectorizer.transform(X_cv['teacher_prefix'].values)
teacher_prefix_one_hot_X_test = vectorizer.transform(X_test['teacher_prefix'].values)


print("="*50)
print("Shape of matrix after one hot encoding(teacher_prefix) ")
print("="*50)
print("X_train  :",teacher_prefix_one_hot_X_train.shape)
print("X_cv     :",teacher_prefix_one_hot_X_cv.shape)
print("X_test   :",teacher_prefix_one_hot_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Shape of matrix after one hot encoding(teacher_prefix)
==================================================
X_train  : (49041, 5)
X_cv     : (24155, 5)
X_test   : (36052, 5)
==================================================
['dr', 'mr', 'mrs', 'ms', 'teacher']
==================================================
```

project_grade_category

```
# we use count vectorizer to convert the values into one hot encoded features

vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['project_grade_category'].values)

project_grade_one_hot_X_train = vectorizer.transform(X_train['project_grade_category'].values)
project_grade_one_hot_X_cv = vectorizer.transform(X_cv['project_grade_category'].values)
project_grade_one_hot_X_test = vectorizer.transform(X_test['project_grade_category'].values)

print("="*50)
print("Shape of matrix after one hot encoding(project_grade_category) ")
print("="*50)
print("X_train  :",project_grade_one_hot_X_train.shape)
print("X_cv     :",project_grade_one_hot_X_cv.shape)
print("X_test   :",project_grade_one_hot_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Shape of matrix after one hot encoding(project_grade_category)
==================================================
X_train  : (49041, 4)
X_cv     : (24155, 4)
X_test   : (36052, 4)
==================================================
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
==================================================
```

clean_categories

```
# we use count vectorizer to convert the values into one hot encoded features

vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['clean_categories'].values)

clean_categories_one_hot_X_train = vectorizer.transform(X_train['clean_categories'].values)
clean_categories_one_hot_X_cv = vectorizer.transform(X_cv['clean_categories'].values)
clean_categories_one_hot_X_test = vectorizer.transform(X_test['clean_categories'].values)

print("="*50)
print("Shape of matrix after one hot encoding(clean_categories) ")
print("="*50)
print("X_train  :",clean_categories_one_hot_X_train.shape)
print("X_cv     :",clean_categories_one_hot_X_cv.shape)
print("X_test   :",clean_categories_one_hot_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Shape of matrix after one hot encoding(clean_categories)
==================================================
X_train  : (49041, 9)
X_cv     : (24155, 9)
X_test   : (36052, 9)
==================================================
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
==================================================
```

clean_subcategories

```
# we use count vectorizer to convert the values into one hot encoded features

vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)

clean_subcategories_one_hot_X_train = vectorizer.transform(X_train['clean_subcategories'].values)
clean_subcategories_one_hot_X_cv = vectorizer.transform(X_cv['clean_subcategories'].values)
clean_subcategories_one_hot_X_test = vectorizer.transform(X_test['clean_subcategories'].values)

print("="*50)
print("Shape of matrix after one hot encoding(clean_categories) ")
print("="*50)
print("X_train  :",clean_subcategories_one_hot_X_train.shape)
print("X_cv     :",clean_subcategories_one_hot_X_cv.shape)
print("X_test   :",clean_subcategories_one_hot_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Shape of matrix after one hot encoding(clean_categories)
==================================================
X_train  : (49041, 30)
X_cv     : (24155, 30)
X_test   : (36052, 30)
==================================================
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym
==================================================
```

teacher_number_of_previously_posted_projects

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

previously_posted_X_train = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
previously_posted_X_cv = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
previously_posted_X_test = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("="*50)
print("Shape of matrix after vectorizations(teacher_number_of_previously_posted_projects) ")
print("="*50)
print("X_train  :",previously_posted_X_train.shape)
print("X_cv     :",previously_posted_X_cv.shape)
print("X_test   :",previously_posted_X_test.shape)
print("="*50)
```

```
==================================================
Shape of matrix after vectorizations(teacher_number_of_previously_posted_projects)
==================================================
X_train  : (49041, 1)
X_cv     : (24155, 1)
X_test   : (36052, 1)
==================================================
```

price

```
normalizer = Normalizer()

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

price_X_train = normalizer.transform(X_train['price'].values.reshape(-1,1))
price_X_cv = normalizer.transform(X_cv['price'].values.reshape(-1,1))
price_X_test = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("="*50)
print("Shape of matrix after vectorizations(price) ")
print("="*50)
print("X_train  :",price_X_train.shape)
print("X_cv     :",price_X_cv.shape)
print("X_test   :",price_X_test.shape)
print("="*50)
```

```
==================================================
Shape of matrix after vectorizations(price)
==================================================
X_train  : (49041, 1)
X_cv     : (24155, 1)
X_test   : (36052, 1)
==================================================
```

2.3 Make Data Model Ready: encoding eassay, and project_title

Essay - Bag of Words

```
vectorizer = CountVectorizer(min_df=10,binary=True)
vectorizer.fit(X_train['essay'].values)

essay_bag_of_words_X_train = vectorizer.transform(X_train['essay'].values)
essay_bag_of_words_X_cv = vectorizer.transform(X_cv['essay'].values)
essay_bag_of_words_X_test = vectorizer.transform(X_test['essay'].values)

print("="*50)
print("Bag of Words")
print("Shape of matrix after vectorizations (essay)")
print("="*50)
print("X_train  :",essay_bag_of_words_X_train.shape)
print("X_cv     :",essay_bag_of_words_X_cv.shape)
print("X_test   :",essay_bag_of_words_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Bag of Words
Shape of matrix after vectorizations (essay)
==================================================
X_train  : (49041, 12177)
X_cv     : (24155, 12177)
X_test   : (36052, 12177)
==================================================
['00', '000', '10', '100', '1000', '101', '103', '104', '105', '10th', '11', '110', '1100', '112', '115', '11th', '12', '120', '1200', '125', '12th', '13', '130', '1300', '13th', '14', '140', '1400', '15', '150', '1500', '16', '160', '1600', '17',
==================================================
```

Essay - Tf-idf

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['essay'].values)

essay_tfidf_X_train = vectorizer.transform(X_train['essay'].values)
essay_tfidf_X_cv = vectorizer.transform(X_cv['essay'].values)
essay_tfidf_X_test = vectorizer.transform(X_test['essay'].values)

print("="*50)
print("Tfidf Vectorizer")
print("Shape of matrix after vectorizations (essay)")
print("="*50)
print("X_train  :",essay_tfidf_X_train.shape)
print("X_cv     :",essay_tfidf_X_cv.shape)
print("X_test   :",essay_tfidf_X_test.shape)
print("="*50)
#print(vectorizer.get_feature_names())
#print("="*50)
```

```
==================================================
Tfidf Vectorizer
Shape of matrix after vectorizations (essay)
==================================================
X_train  : (49041, 12177)
X_cv     : (24155, 12177)
X_test   : (36052, 12177)
==================================================
```

Eassay-Average Word2Vec

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('/content/drive/My Drive/Colab Notebooks/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())


# average Word2Vec
# compute average word2vec for each review.
essay_avg_w2v_X_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    essay_avg_w2v_X_train.append(vector)
print()
```

```
100%|██████████| 49041/49041 [00:13<00:00, 3670.29it/s]
```

```
essay_avg_w2v_X_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    essay_avg_w2v_X_cv.append(vector)
print()
```

```
100%|██████████| 24155/24155 [00:06<00:00, 3742.64it/s]
```

```
essay_avg_w2v_X_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) #as #word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    essay_avg_w2v_X_test.append(vector)
print()
```

```
100%|██████████| 36052/36052 [00:09<00:00, 3662.30it/s]
```

```
print("="*50)
print("Average Word2Vec")
print("Shape of matrix after vectorizations (essay)")
print("="*50)
print("X_train  :(",len(essay_avg_w2v_X_train),",",len(essay_avg_w2v_X_train[0]),")")
print("X_cv  :(",len(essay_avg_w2v_X_cv),",",len(essay_avg_w2v_X_cv[0]),")")
print("X_test  :(",len(essay_avg_w2v_X_test),",",len(essay_avg_w2v_X_test[0]),")")
print("="*50)
```

```
==================================================
Average Word2Vec
Shape of matrix after vectorizations (essay)
==================================================
X_train  :( 49041 , 300 )
X_cv  :( 24155 , 300 )
X_test  :( 36052 , 300 )
==================================================
```

TFIDF W2V

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

```
# average Word2Vec
# compute average word2vec for each review.
essay_tfidf_w2v_X_train= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    essay_tfidf_w2v_X_train.append(vector)

print()
```

```
100%|██████████| 49041/49041 [01:34<00:00, 516.99it/s]
```

```
# average Word2Vec
# compute average word2vec for each review.
essay_tfidf_w2v_X_cv= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    essay_tfidf_w2v_X_cv.append(vector)

print()
```

```
100%|██████████| 24155/24155 [00:46<00:00, 520.83it/s]
```

```
# average Word2Vec
# compute average word2vec for each review.
essay_tfidf_w2v_X_test=[]; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    essay_tfidf_w2v_X_test.append(vector)

print()
```

```
100%|██████████| 36052/36052 [01:09<00:00, 519.24it/s]
```

```
print("="*50)
print("Average Word2Vec")
print("Shape of matrix after vectorizations (essay)")
print("="*50)
print("X_train  :(",len(essay_tfidf_w2v_X_train),",",len(essay_tfidf_w2v_X_train[0]),")")
print("X_cv  :(",len(essay_tfidf_w2v_X_cv),",",len(essay_tfidf_w2v_X_cv[0]),")")
print("X_test  :(",len(essay_tfidf_w2v_X_test),",",len(essay_tfidf_w2v_X_test[0]),")")
print("="*50)
```

```
==================================================
Average Word2Vec
Shape of matrix after vectorizations (essay)
==================================================
X_train  :( 49041 , 300 )
X_cv  :( 24155 , 300 )
X_test  :( 36052 , 300 )
==================================================
```

project_title-BoW

```
vectorizer = CountVectorizer(min_df=10,binary=True)
vectorizer.fit(X_train['project_title'].values)

project_title_bag_of_words_X_train = vectorizer.transform(X_train['project_title'].values)
project_title_bag_of_words_X_cv = vectorizer.transform(X_cv['project_title'].values)
project_title_bag_of_words_X_test = vectorizer.transform(X_test['project_title'].values)

print("="*50)
print("Bag of Words")
print("Shape of matrix after vectorizations (project_title)")
print("="*50)
print("X_train  :",project_title_bag_of_words_X_train.shape)
print("X_cv     :",project_title_bag_of_words_X_cv.shape)
print("X_test   :",project_title_bag_of_words_X_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
print("="*50)
```

```
==================================================
Bag of Words
Shape of matrix after vectorizations (project_title)
==================================================
X_train  : (49041, 2084)
X_cv     : (24155, 2084)
X_test   : (36052, 2084)
==================================================
['04', '05', '10', '100', '101', '13', '16', '1st', '2016', '2017', '21st', '2nd', '3d', '3doodler', '3rd', '4th', '5th', '60', '6th', '7th', '8th', 'abc', 'abcs', 'abilities', 'aboard', 'about', 'above', 'academic', 'academics', 'academy', 'access
==================================================
```

Project_title TfidfVectorizer

```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['project_title'].values)

project_title_tfidf_X_train = vectorizer.transform(X_train['project_title'].values)
project_title_tfidf_X_cv = vectorizer.transform(X_cv['project_title'].values)
project_title_tfidf_X_test = vectorizer.transform(X_test['project_title'].values)

print("="*50)
print("Tfidf Vectorizer")
print("Shape of matrix after vectorizations (project_title)")
print("="*50)
print("X_train  :",project_title_tfidf_X_train.shape)
print("X_cv     :",project_title_tfidf_X_cv.shape)
print("X_test   :",project_title_tfidf_X_test.shape)
print("="*50)
#print(vectorizer.get_feature_names())
#print("="*50)
```

```
==================================================
Tfidf Vectorizer
Shape of matrix after vectorizations (project_title)
==================================================
X_train  : (49041, 2084)
X_cv     : (24155, 2084)
X_test   : (36052, 2084)
==================================================
```

Project_title- Average word2vec

```
# compute average word2vec for each review.
project_title_avg_w2v_X_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    project_title_avg_w2v_X_train.append(vector)
print()
```

```
100%|██████████| 49041/49041 [00:00<00:00, 62189.10it/s]
```

```
project_title_avg_w2v_X_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    project_title_avg_w2v_X_cv.append(vector)
print()
```

```
100%|██████████| 24155/24155 [00:00<00:00, 58372.90it/s]
```

```
project_title_avg_w2v_X_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title'].values): # for each review/sentence
    vector = np.zeros(300) #as #word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    project_title_avg_w2v_X_test.append(vector)
print()
```

```
100%|██████████| 36052/36052 [00:00<00:00, 61696.32it/s]
```

```
print("="*50)
print("Average Word2Vec")
print("Shape of matrix after vectorizations (project_title)")
print("="*50)
print("X_train  :(",len(project_title_avg_w2v_X_train),",",len(project_title_avg_w2v_X_train[0]),")")
print("X_cv  :(",len(project_title_avg_w2v_X_cv),",",len(project_title_avg_w2v_X_cv[0]),")")
print("X_test  :(",len(project_title_avg_w2v_X_test),",",len(project_title_avg_w2v_X_test[0]),")")
print("="*50)
```

```
==================================================
Average Word2Vec
Shape of matrix after vectorizations (project_title)
==================================================
X_train  :( 49041 , 300 )
X_cv  :( 24155 , 300 )
X_test  :( 36052 , 300 )
==================================================
```

Project_title- average Word2Vec

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())


# average Word2Vec
# compute average word2vec for each review.
project_title_tfidf_w2v_X_train= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    project_title_tfidf_w2v_X_train.append(vector)

print()
```

```
100%|██████████| 49041/49041 [00:01<00:00, 29952.82it/s]
```

```
# average Word2Vec
# compute average word2vec for each review.
project_title_tfidf_w2v_X_cv= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
```

```
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        project_title_tfidf_w2v_X_cv.append(vector)

    print()
```

```
100%|██████████| 24155/24155 [00:00<00:00, 30805.29it/s]
```

```
    # average Word2Vec
    # compute average word2vec for each review.
    project_title_tfidf_w2v_X_test=[]; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(X_test['project_title'].values): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                vec = model[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        project_title_tfidf_w2v_X_test.append(vector)

    print()
```

```
100%|██████████| 36052/36052 [00:01<00:00, 25347.05it/s]
```

```
    print("="*50)
    print("Average Word2Vec")
    print("Shape of matrix after vectorizations (project_title)")
    print("="*50)
    print("X_train  :(",len(project_title_tfidf_w2v_X_train),",",len(project_title_tfidf_w2v_X_train[0]),")")
    print("X_cv     :(",len(project_title_tfidf_w2v_X_cv),",",len(project_title_tfidf_w2v_X_cv[0]),")")
    print("X_test   :(",len(project_title_tfidf_w2v_X_test),",",len(project_title_tfidf_w2v_X_test[0]),")")
    print("="*50)
```

```
==================================================
Average Word2Vec
Shape of matrix after vectorizations (project_title)
==================================================
X_train  :( 49041 , 300 )
X_cv     :( 24155 , 300 )
X_test   :( 36052 , 300 )
==================================================
```

Combining all feature vecters into one Vecter

```
from scipy.sparse import hstack
#Bag_of_words
X_train_Bow = hstack((states_one_hot_X_train,teacher_prefix_one_hot_X_train,project_grade_one_hot_X_train,clean_categories_one_hot_X_train,
clean_subcategories_one_hot_X_train,previously_posted_X_train,price_X_train,essay_bag_of_words_X_train,project_title_bag_of_words_X_train)).

X_cv_Bow = hstack((states_one_hot_X_cv, teacher_prefix_one_hot_X_cv,project_grade_one_hot_X_cv,clean_categories_one_hot_X_cv,
clean_subcategories_one_hot_X_cv,previously_posted_X_cv,price_X_cv,essay_bag_of_words_X_cv,project_title_bag_of_words_X_cv)).tocsr()

X_test_Bow = hstack((states_one_hot_X_test, teacher_prefix_one_hot_X_test,project_grade_one_hot_X_test,clean_categories_one_hot_X_test,
clean_subcategories_one_hot_X_test,previously_posted_X_test,price_X_test,essay_bag_of_words_X_test,project_title_bag_of_words_X_test)).tocsr


X_train_tfidf = hstack((states_one_hot_X_train,teacher_prefix_one_hot_X_train,project_grade_one_hot_X_train,clean_categories_one_hot_X_train
clean_subcategories_one_hot_X_train,previously_posted_X_train,price_X_train,essay_tfidf_X_train,project_title_tfidf_X_train)).tocsr()

X_cv_tfidf = hstack((states_one_hot_X_cv, teacher_prefix_one_hot_X_cv,project_grade_one_hot_X_cv,clean_categories_one_hot_X_cv,
clean_subcategories_one_hot_X_cv,previously_posted_X_cv,price_X_cv,essay_tfidf_X_cv,project_title_tfidf_X_cv)).tocsr()

X_test_tfidf = hstack((states_one_hot_X_test, teacher_prefix_one_hot_X_test,project_grade_one_hot_X_test,clean_categories_one_hot_X_test,
clean_subcategories_one_hot_X_test,previously_posted_X_test,price_X_test,essay_tfidf_X_test,project_title_tfidf_X_test)).tocsr()


X_train_avg_w2v = hstack((states_one_hot_X_train,teacher_prefix_one_hot_X_train,project_grade_one_hot_X_train,clean_categories_one_hot_X_tra
clean_subcategories_one_hot_X_train,previously_posted_X_train,price_X_train,essay_avg_w2v_X_train,project_title_avg_w2v_X_train)).tocsr()

X_cv_avg_w2v= hstack((states_one_hot_X_cv, teacher_prefix_one_hot_X_cv,project_grade_one_hot_X_cv,clean_categories_one_hot_X_cv,
clean_subcategories_one_hot_X_cv,previously_posted_X_cv,price_X_cv,essay_avg_w2v_X_cv,project_title_avg_w2v_X_cv)).tocsr()

X_test_avg_w2v = hstack((states_one_hot_X_test, teacher_prefix_one_hot_X_test,project_grade_one_hot_X_test,clean_categories_one_hot_X_test,
clean_subcategories_one_hot_X_test,previously_posted_X_test,price_X_test,essay_avg_w2v_X_test,project_title_avg_w2v_X_test)).tocsr()


X_train_tfidf_w2v = hstack((states_one_hot_X_train,teacher_prefix_one_hot_X_train,project_grade_one_hot_X_train,clean_categories_one_hot_X_t
clean_subcategories_one_hot_X_train,previously_posted_X_train,price_X_train,essay_tfidf_w2v_X_train,project_title_tfidf_w2v_X_train)).tocsr(

X_cv_tfidf_w2v= hstack((states_one_hot_X_cv, teacher_prefix_one_hot_X_cv,project_grade_one_hot_X_cv,clean_categories_one_hot_X_cv,
clean_subcategories_one_hot_X_cv,previously_posted_X_cv,price_X_cv,essay_tfidf_w2v_X_cv,project_title_tfidf_w2v_X_cv)).tocsr()

X_test_tfidf_w2v = hstack((states_one_hot_X_test, teacher_prefix_one_hot_X_test,project_grade_one_hot_X_test,clean_categories_one_hot_X_test
clean_subcategories_one_hot_X_test,previously_posted_X_test,price_X_test,essay_tfidf_w2v_X_test,project_title_tfidf_w2v_X_test)).tocsr()


def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

2.4 Appling KNN on different kind of featurization as mentioned in the instructions

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

2.4.1 Applying KNN brute force on BOW, SET 1

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_train_Bow, y_train)

    y_train_pred = batch_predict(neigh, X_train_Bow)
    y_cv_pred = batch_predict(neigh, X_cv_Bow)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

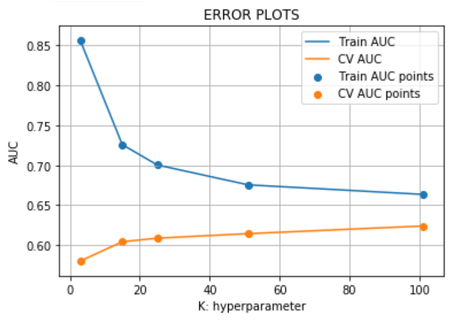
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
plt.show()
```

```
100%|██████████| 5/5 [17:18<00:00, 206.89s/it]
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_train_tfidf, y_train)

    y_train_pred = batch_predict(neigh, X_train_tfidf)
    y_cv_pred = batch_predict(neigh, X_cv_tfidf)

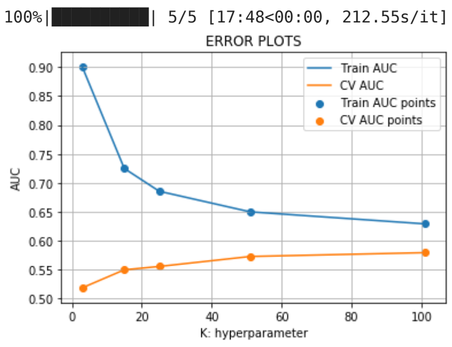
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
100%|██████████| 5/5 [17:48<00:00, 212.55s/it]
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

```
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_train_avg_w2v, y_train)

    y_train_pred = batch_predict(neigh, X_train_avg_w2v)
    y_cv_pred = batch_predict(neigh, X_cv_avg_w2v)

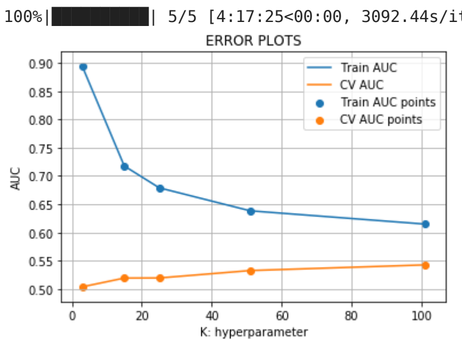
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
100%|██████████| 5/5 [4:17:25<00:00, 3092.44s/it]
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_train_tfidf_w2v, y_train)

    y_train_pred = batch_predict(neigh, X_train_tfidf_w2v)
    y_cv_pred = batch_predict(neigh, X_cv_tfidf_w2v)

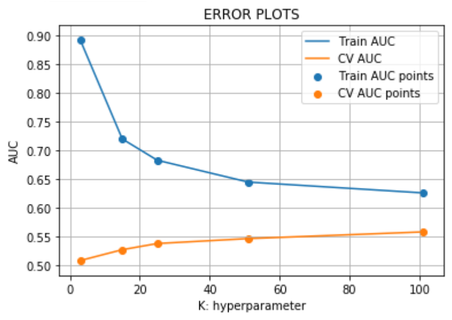
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
100%|██████████| 5/5 [4:13:40<00:00, 3050.22s/it]
```



2.5 Feature selection with `SelectKBest`

BOW

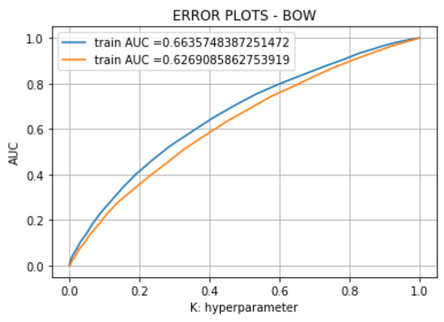
```
best_k = 101

neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_Bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_Bow)
y_test_pred = batch_predict(neigh, X_test_Bow)

train_fpr_bow, train_tpr_bow, tr_thresholds_bow = roc_curve(y_train, y_train_pred)
test_fpr_bow, test_tpr_bow, te_thresholds_bow = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr_bow, train_tpr_bow, label="train AUC ="+str(auc(train_fpr_bow, train_tpr_bow)))
plt.plot(test_fpr_bow, test_tpr_bow, label="train AUC ="+str(auc(test_fpr_bow, test_tpr_bow)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS - BOW")
plt.grid()
plt.show()
```

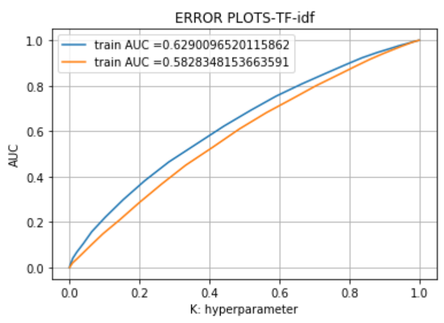



```
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_tfidf)
y_test_pred = batch_predict(neigh, X_test_tfidf)

train_fpr_tfidf, train_tpr_tfidf, tr_thresholds_tfidf = roc_curve(y_train, y_train_pred)
test_fpr_tfidf, test_tpr_tfidf, te_thresholds_tfidf = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC ="+str(auc(train_fpr_tfidf, train_tpr_tfidf)))
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="train AUC ="+str(auc(test_fpr_tfidf, test_tpr_tfidf)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS-TF-idf")
plt.grid()
plt.show()
```

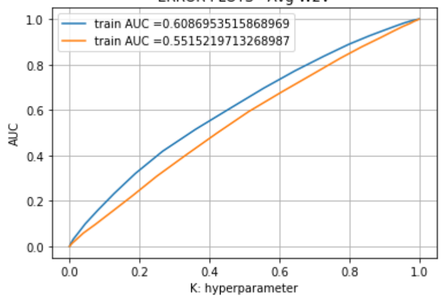



```
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_avg_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_avg_w2v)
y_test_pred = batch_predict(neigh, X_test_avg_w2v)

train_fpr_avg_w2v, train_tpr_avg_w2v, tr_thresholds_avg_w2v = roc_curve(y_train, y_train_pred)
test_fpr_avg_w2v, test_tpr_avg_w2v, te_thresholds_avg_w2v = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr_avg_w2v, train_tpr_avg_w2v, label="train AUC ="+str(auc(train_fpr_avg_w2v, train_tpr_avg_w2v)))
plt.plot(test_fpr_avg_w2v, test_tpr_avg_w2v, label="train AUC ="+str(auc(test_fpr_avg_w2v, test_tpr_avg_w2v)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS - Avg W2V")
plt.grid()
plt.show()
```



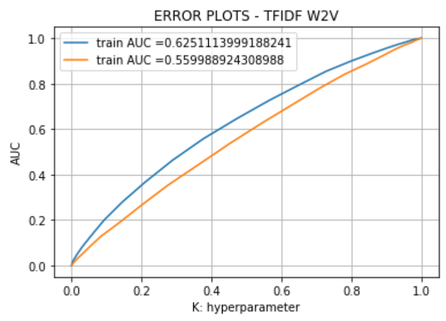

```
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_tfidf_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_tfidf_w2v)
y_test_pred = batch_predict(neigh, X_test_tfidf_w2v)

train_fpr_tfidf_w2v, train_tpr_tfidf_w2v, tr_thresholds_tfidf_w2v = roc_curve(y_train, y_train_pred)
test_fpr_tfidf_w2v, test_tpr_tfidf_w2v, te_thresholds_tfidf_w2v = roc_curve(y_test, y_test_pred)



plt.plot(train_fpr_tfidf_w2v, train_tpr_tfidf_w2v, label="train AUC ="+str(auc(train_fpr_tfidf_w2v, train_tpr_tfidf_w2v)))
plt.plot(test_fpr_tfidf_w2v, test_tpr_tfidf_w2v, label="train AUC ="+str(auc(test_fpr_tfidf_w2v, test_tpr_tfidf_w2v)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS - TFIDF W2V")
plt.grid()
plt.show()
```



3. Conclusions

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "HyperParameter", "AUC_Train","AUC_Test"]
x.add_row(["BOW", "Brute", best_k, str(auc(train_fpr_bow, train_tpr_bow)),str(auc(test_fpr_bow, test_tpr_bow))])
x.add_row(["TF-idf", "Brute", best_k,  str(auc(train_fpr_tfidf, train_tpr_tfidf)),str(auc(test_fpr_tfidf, test_tpr_tfidf))])
x.add_row(["Avg W2V", "Brute", best_k,  str(auc(train_fpr_avg_w2v, train_tpr_avg_w2v)),str(auc(test_fpr_avg_w2v, test_tpr_avg_w2v)) ])
x.add_row(["TFIDF W2V", "Brute", best_k, str(auc(train_fpr_tfidf_w2v, train_tpr_tfidf_w2v)),str(auc(test_fpr_tfidf_w2v, test_tpr_tfidf_w2v))


print(x)
```

```
+------------+-------+---------------+--------------------+--------------------+
| Vectorizer | Model | HyperParameter |     AUC_Train      |      AUC_Test      |
+------------+-------+---------------+--------------------+--------------------+
|    BOW     | Brute |      101      | 0.6635748387251472 | 0.6269085862753919 |
|   TF-idf   | Brute |      101      | 0.6290096520115862 | 0.5828348153663591 |
|  Avg W2V   | Brute |      101      | 0.6086953515868969 | 0.5515219713268987 |
| TFIDF W2V  | Brute |      101      | 0.6251113999188241 | 0.559988924308988  |
+------------+-------+---------------+--------------------+--------------------+
```