prabhudayala@gmail.com_3

April 23, 2019

1 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

How to scale current manual processes and resources to screen 500,000 projects so that they ca
<li>How to increase the consistency of project vetting across different volunteers to improve
<li>How to focus volunteer time on the applications that need the most assistance</li>
</ul>
```

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.

1.1 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`

Feature	Description
`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.

1.1.1 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 [23]: %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
```

1.2 1.1 Reading Data

```
In [24]: project_data = pd.read_csv('train_data.csv')
         resource_data = pd.read_csv('resources.csv')
```

```
In [25]: 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']
```

In [26]:
```
# 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.c

#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/408
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[26]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_subject_categories	project_subject_subcategories	project_title	project_essay_1	project_essay_2
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Math & Science	Applied Sciences, Health & Life Science	Engineering STEAM into the Primary Classroom	I have been fortunate enough to use the Fairy ...	My students come from a variety of backgrounds...
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Special Needs	Special Needs	Sensory Tools for Focus	Imagine being 8-9 years old. You're in your th...	Most of my students have autism, anxiety, anot...

```
                                                project_essay_3  \
55660  Each month I try to do several science or STEM...
76127  It is tough to do more than one thing at a tim...

                                                project_essay_4  \
55660  It is challenging to develop high quality scie...
76127  When my students are able to calm themselves d...

                                     project_resource_summary  \
55660  My students need STEM kits to learn critical s...
76127  My students need Boogie Boards for quiet senso...

       teacher_number_of_previously_posted_projects  project_is_approved
55660                                            53                   1
76127                                             4                   1
```

In [27]:
```
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[27]:
```
         id                                          description  quantity  \
0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1
1  p069063        Bouncy Bands for Desks (Blue support pipes)         3

    price
0  149.00
1   14.95
```

In [28]:
```
project_data.columns
```

Out[28]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', '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'],
      dtype='object')
```

In [29]:
```
# join two dataframes in python:
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_
price_data.head(2)
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [30]:
```
project_data.columns
```

```
Out[30]: Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
                'Date', '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',
                'price', 'quantity'],
               dtype='object')
```

```
In [31]: print(project_data.shape)
         project_data = project_data.dropna(subset=['teacher_prefix'])
         print(project_data.shape)
```

```
(109248, 19)
(109245, 19)
```

1.3 1.2 preprocessing of `project_subject_categories`

```
In [32]: catogories = list(project_data['project_subject_categories'].values)
         # remove special characters from list of strings python: https://stackoverflow.com/a/

         # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
         # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-st
         # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-py
         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", "Warm
                 if 'The' in j.split(): # this will split each of the catogory based on space
                     j=j.replace('The','') # if we have the words "The" we are going to replac
                 j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:
                 temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing sp
                 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]))
```

1.4 1.3 preprocessing of project_subject_subcategories

In [33]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-st
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-py

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", "Warm
        if 'The' in j.split(): # this will split each of the catogory based on space
            j=j.replace('The','') # if we have the words "The" we are going to replac
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing sp
        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/22898595/40840
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]))
```

1.5 1.3 Text preprocessing

In [34]:
```
# 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 [35]:
```
project_data.head(2)
```

Out[35]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5

```
                                  project_title  \
0  Engineering STEAM into the Primary Classroom
1                       Sensory Tools for Focus

                                     project_essay_1  \
0  I have been fortunate enough to use the Fairy ...
1  Imagine being 8-9 years old. You're in your th...

                                     project_essay_2  \
0  My students come from a variety of backgrounds...
1  Most of my students have autism, anxiety, anot...

                                     project_essay_3  \
0  Each month I try to do several science or STEM...
1  It is tough to do more than one thing at a tim...

                                     project_essay_4  \
0  It is challenging to develop high quality scie...
1  When my students are able to calm themselves d...

                             project_resource_summary  \
0  My students need STEM kits to learn critical s...
1  My students need Boogie Boards for quiet senso...

   teacher_number_of_previously_posted_projects  project_is_approved   price  \
0                                            53                    1  725.05
1                                             4                    1  213.03

   quantity clean_categories                clean_subcategories  \
0         4     Math_Science  AppliedSciences Health_LifeScience
1         8     SpecialNeeds                         SpecialNeeds

                                              essay
0  I have been fortunate enough to use the Fairy ...
1  Imagine being 8-9 years old. You're in your th...
```

In [36]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [37]:
```
# 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 [38]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
Some of my students come from difficult family lives, but they do not let that stop them. We ha
==================================================
```

In [39]:
```
# 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'
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', '
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
            'won', "won't", 'wouldn', "wouldn't"]
```

In [40]:
```
# 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%|| 109245/109245 [00:44<00:00, 2441.81it/s]
```

```
In [41]: # after preprocesing
         preprocessed_essays[20000]
```

```
Out[41]: 'students come difficult family lives not let stop built community classroom allows st
```

```
In [42]: project_data["essay"]=preprocessed_essays
         project_data.essay.values[20000]
```

```
Out[42]: 'students come difficult family lives not let stop built community classroom allows st
```

1.4 Preprocessing of project_title

```
In [43]: from tqdm import tqdm
         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.lower() not in stopwords)
             preprocessed_project_title.append(sent.lower().strip())
```

```
100%|| 109245/109245 [00:02<00:00, 54330.08it/s]
```

```
In [44]: print(project_data['project_title'].values[20000])
         project_data['project_title']=preprocessed_project_title
         print(project_data['project_title'].values[20000])
```

```
Wiggle While We Learn
wiggle learn
```

2 Assignment 3: Apply KNN

```
<li><strong>[Task-1] Apply KNN(brute force version) on these feature sets</strong>
    <ul>
        <li><font color='red'>Set 1</font>: categorical, numerical features + project_title(BOW
        <li><font color='red'>Set 2</font>: categorical, numerical features + project_title(TFI
        <li><font color='red'>Set 3</font>: categorical, numerical features + project_title(AVG
        <li><font color='red'>Set 4</font>: categorical, numerical features + project_title(TFI
    </ul>
</li>
```

```
<br>
<li><strong>Hyper paramter tuning to find best K</strong>
    <ul>
<li>Find the best hyper parameter which results in the maximum <a href='https://www.appliedaic
<li>Find the best hyper paramter using k-fold cross validation (or) simple cross validation da
<li>Use gridsearch-cv or randomsearch-cv or  write your own for loops to do this task</li>
    </ul>
</li>
<br>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once you find the best hyper parameter, you need to train your model-M using the best hype
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<li><strong> [Task-2] </strong>
    <ul>
        <li>Select top 2000 features from feature <font color='red'>Set 2</font> using <a href=
```

and then apply KNN on top of these features

```
        <li>
            <pre>
            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)
            </pre>
        </li>
        <li>Repeat the steps 2 and 3 on the data matrix after feature selection</li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
```

```
</li>
    </ul>
```

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

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

```
In [45]: sampling=True
         undersampling=False
         if (not sampling):
             print("Total data ",project_data.shape)

         else:
             if(sampling and undersampling):
                 print("Total data ",project_data.shape)
                 project_data_negative=project_data[project_data.project_is_approved==0]
                 project_data_positive=project_data[project_data.project_is_approved==1]
                 project_data_positive=project_data_positive.sample(n=project_data_negative.sha
                 print("Positive points: ",project_data_positive.shape[0])
                 print("Negaitive points: ",project_data_negative.shape[0])
                 project_data=pd.concat([project_data_positive,project_data_negative])
             else:
                 print("Total data ",project_data.shape)
                 project_data_negative=project_data[project_data.project_is_approved==0]
                 project_data_positive=project_data[project_data.project_is_approved==1]
                 project_data_negative=project_data_negative.sample(n=project_data_positive.sha
                 print("Positive points: ",project_data_positive.shape[0])
                 print("Negaitive points: ",project_data_negative.shape[0])
                 project_data=pd.concat([project_data_positive,project_data_negative])

         data_point_size=50000
         project_data=project_data.sample(n=data_point_size,random_state=42,replace=True)
         print("positive and negative counts")
         print(project_data.project_is_approved.value_counts())
         project_data_Y=project_data.project_is_approved
         project_data_X=project_data.drop(columns=['project_is_approved'])
         print("After sampling: ",project_data_X.shape)
```

```
Total data  (109245, 20)
Positive points:  92703
```

```
Negaitive points:  92703
positive and negative counts
1    25022
0    24978
Name: project_is_approved, dtype: int64
After sampling:  (50000, 19)
```

```
In [46]: from sklearn.model_selection import train_test_split
         project_data_X_train,project_data_X_test,project_data_Y_train,project_data_Y_test=tra
```

```
In [47]: print(project_data_X_train.shape)
         print(project_data_X_test.shape)
         print(project_data_Y_train.shape)
         print(project_data_Y_test.shape)
```

```
(40000, 19)
(10000, 19)
(40000,)
(10000,)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

2.2.1 Categorical features

```
In [48]: from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False
         vectorizer.fit(project_data_X_train['clean_categories'].values)
         print(vectorizer.get_feature_names())

         #for train data
         categories_one_hot_train = vectorizer.transform(project_data_X_train['clean_categorie
         print("Shape of matrix after one hot encodig ",categories_one_hot_train.shape)


         #for test
         categories_one_hot_test = vectorizer.transform(project_data_X_test['clean_categories'
         print("Shape of matrix after one hot encodig ",categories_one_hot_test.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'I
Shape of matrix after one hot encodig  (40000, 9)
Shape of matrix after one hot encodig  (10000, 9)
```

```
In [49]: vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fa
         vectorizer.fit(project_data_X_train['clean_subcategories'].values)
         print(vectorizer.get_feature_names())

         #for train data
```

```
        sub_categories_one_hot_train = vectorizer.transform(project_data_X_train['clean_subcat
        print("Shape of matrix after one hot encodig ",sub_categories_one_hot_train.shape)

        #for test
        sub_categories_one_hot_test = vectorizer.transform(project_data_X_test['clean_subcateg
        print("Shape of matrix after one hot encodig ",sub_categories_one_hot_test.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
Shape of matrix after one hot encodig  (40000, 30)
Shape of matrix after one hot encodig  (10000, 30)
```

```
In [50]: project_data_X_train.teacher_prefix = project_data_X_train.teacher_prefix.replace(np.n
         print(project_data_X_train.teacher_prefix.value_counts())
         project_data_X_test.teacher_prefix = project_data_X_test.teacher_prefix.replace(np.nan
         print(project_data_X_test.teacher_prefix.value_counts())
```

```
Mrs.       20638
Ms.        14449
Mr.         3920
Teacher      984
Dr.            9
Name: teacher_prefix, dtype: int64
Mrs.       5097
Ms.        3657
Mr.        1002
Teacher     242
Dr.           2
Name: teacher_prefix, dtype: int64
```

```
In [51]: # we use count vectorizer to convert the values into one hot encoded features
         vectorizer = CountVectorizer(vocabulary=list(project_data_X_train['teacher_prefix'].un
         vectorizer.fit(project_data_X_train['teacher_prefix'].values)
         print(vectorizer.get_feature_names())

         teacher_prefix_one_hot_train = vectorizer.transform(project_data_X_train['teacher_pref
         print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot_train.shape)

         teacher_prefix_one_hot_test = vectorizer.transform(project_data_X_test['teacher_prefix
         print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot_test.shape)
```

```
['Ms.', 'Mrs.', 'Mr.', 'Teacher', 'Dr.']
Shape of matrix after one hot encodig  (40000, 5)
Shape of matrix after one hot encodig  (10000, 5)
```

```
In [52]: # we use count vectorizer to convert the values into one hot encoded features
         vectorizer = CountVectorizer(vocabulary=list(project_data_X_train['project_grade_categ
```

```
        vectorizer.fit(project_data_X_train['project_grade_category'].values)
        print(vectorizer.get_feature_names())

        project_grade_category_one_hot_train = vectorizer.transform(project_data_X_train['proj
        print("Shape of matrix after one hot encodig ",project_grade_category_one_hot_train.sh

        project_grade_category_one_hot_test = vectorizer.transform(project_data_X_test['projec
        print("Shape of matrix after one hot encodig ",project_grade_category_one_hot_test.sha
```

```
['Grades 3-5', 'Grades 9-12', 'Grades 6-8', 'Grades PreK-2']
Shape of matrix after one hot encodig  (40000, 4)
Shape of matrix after one hot encodig  (10000, 4)
```

```
In [53]: # we use count vectorizer to convert the values into one hot encoded features
         vectorizer = CountVectorizer(vocabulary=list(project_data_X_train['school_state'].unic
         vectorizer.fit(project_data_X_train['school_state'].values)
         print(vectorizer.get_feature_names())


         school_state_one_hot_train = vectorizer.transform(project_data_X_train['school_state']
         print("Shape of matrix after one hot encodig ",school_state_one_hot_train.shape)

         school_state_one_hot_test = vectorizer.transform(project_data_X_test['school_state'].v
         print("Shape of matrix after one hot encodig ",school_state_one_hot_test.shape)
```

```
['TN', 'PA', 'GA', 'TX', 'FL', 'NC', 'IL', 'NY', 'MO', 'VA', 'WA', 'LA', 'DC', 'KY', 'CA', 'WI
Shape of matrix after one hot encodig  (40000, 51)
Shape of matrix after one hot encodig  (10000, 51)
```

2.2.2 Numerical features

```
In [54]: # check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
         # standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.p
         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.   ..
         # Reshape your data either using array.reshape(-1, 1)

         #price_scalar = StandardScaler()
         #price_scalar.fit(project_data_X_train['price'].values.reshape(-1,1)) # finding the m
         #print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.v

         # Now standardize the data with above maen and variance.
         #price_standardized_train =project_data_X_train['price']# price_scalar.transform(proj
         # Now standardize the data with above maen and variance.
         #price_standardized_test = project_data_X_test['price']#price_scalar.transform(projec
```

In [55]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.
from sklearn.preprocessing import MinMaxScaler

# 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.   ..
# Reshape your data either using array.reshape(-1, 1)

price_scalar = MinMaxScaler()
price_scalar.fit(project_data_X_train['price'].values.reshape(-1,1)) # finding the me
print(price_scalar.data_max_)
print(price_scalar.data_min_)
# Now standardize the data with above maen and variance.
price_standardized_train =price_scalar.transform(project_data_X_train['price'].values
# Now standardize the data with above maen and variance.
price_standardized_test =price_scalar.transform(project_data_X_test['price'].values.re
```

```
[9999.]
[0.69]
```

In [56]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.
from sklearn.preprocessing import StandardScaler,normalize

# 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.   ..
# Reshape your data either using array.reshape(-1, 1)

#price_scalar = StandardScaler()
#price_scalar.fit(project_data_X_train['teacher_number_of_previously_posted_projects'
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.v

# Now standardize the data with above maen and variance.
#teacher_number_of_previously_posted_projects_standardized_train = normalize(project_

# Now standardize the data with above maen and variance.
#teacher_number_of_previously_posted_projects_standardized_test = normalize(project_d
```

In [57]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.
from sklearn.preprocessing import MinMaxScaler

# 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.   ..
```

```
        # Reshape your data either using array.reshape(-1, 1)

        price_scalar = MinMaxScaler()
        price_scalar.fit(project_data_X_train['teacher_number_of_previously_posted_projects'])
        print(price_scalar.data_max_)
        print(price_scalar.data_min_)
        # Now standardize the data with above maen and variance.
        teacher_number_of_previously_posted_projects_standardized_train = price_scalar.transfo

        # Now standardize the data with above maen and variance.
        teacher_number_of_previously_posted_projects_standardized_test = price_scalar.transfor
```

```
[433.]
[0.]
```

2.3 Make Data Model Ready: encoding eassay, and project_title

```
In [58]: vectorizer = CountVectorizer(min_df=10)
         vectorizer.fit(project_data_X_train.essay.values)

         text_bow_train=vectorizer.fit_transform(project_data_X_train.essay.values)
         print(text_bow_train.shape)

         text_bow_test=vectorizer.transform(project_data_X_test.essay.values)
         print(text_bow_test.shape)
```

```
(40000, 11061)
(10000, 11061)
```

```
In [60]: # Similarly you can vectorize for title also
         vectorizer = CountVectorizer(min_df=10)
         vectorizer.fit(project_data_X_train.project_title.values)

         title_text_bow_train=vectorizer.fit_transform(project_data_X_train.project_title.value
         print(title_text_bow_train.shape)

         title_text_bow_test=vectorizer.transform(project_data_X_test.project_title.values)
         print(title_text_bow_test.shape)
```

```
(40000, 1768)
(10000, 1768)
```

```
In [61]: from sklearn.feature_extraction.text import TfidfVectorizer
         vectorizer = TfidfVectorizer(min_df=10)
         vectorizer.fit(project_data_X_train.essay.values)
```

```
        text_tfidf_train=vectorizer.fit_transform(project_data_X_train.essay.values)
        print(text_tfidf_train.shape)

        text_tfidf_test=vectorizer.transform(project_data_X_test.essay.values)
        print(text_tfidf_test.shape)
```

```
(40000, 11061)
(10000, 11061)
```

In [62]:
```
        # Similarly you can vectorize for title also
        from sklearn.feature_extraction.text import TfidfVectorizer
        vectorizer = TfidfVectorizer(min_df=10)
        vectorizer.fit(project_data_X_train.project_title.values)

        title_text_tfidf_train=vectorizer.fit_transform(project_data_X_train.project_title.va
        print(title_text_tfidf_train.shape)

        title_text_tfidf_test=vectorizer.transform(project_data_X_test.project_title.values)
        print(title_text_tfidf_test.shape)
```

```
(40000, 1768)
(10000, 1768)
```

In [63]:
```
        # Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
        def loadGloveModel(gloveFile):
            print ("Loading Glove Model")
            f = open(gloveFile,'r', encoding="utf8")
            model = {}
            for line in tqdm(f):
                splitLine = line.split()
                word = splitLine[0]
                embedding = np.array([float(val) for val in splitLine[1:]])
                model[word] = embedding
            print ("Done.",len(model)," words loaded!")
            return model
        # borrowed from https://therenegadecoder.com/code/how-to-check-if-a-file-exists-in-py
        import os
        exists = os.path.isfile('./glove_vectors')
        if(not exists):
            model = loadGloveModel('glove.42B.300d.txt')

            '''# ============================
            Output:

            Loading Glove Model
            1917495it [06:32, 4879.69it/s]
            Done. 1917495  words loaded!
```

```
            # ============================'''

            words = []
            for i in preproced_texts:
                words.extend(i.split(' '))

            for i in preproced_titles:
                words.extend(i.split(' '))
            print("all the words in the coupus", len(words))
            words = set(words)
            print("the unique words in the coupus", len(words))

            inter_words = set(model.keys()).intersection(words)
            print("The number of words that are present in both glove vectors and our coupus"
                  len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

            words_courpus = {}
            words_glove = set(model.keys())
            for i in words:
                if i in words_glove:
                    words_courpus[i] = model[i]
            print("word 2 vec length", len(words_courpus))


            # stronging variables into pickle files python: http://www.jessicayung.com/how-to

            import pickle
            with open('glove_vectors', 'wb') as f:
                pickle.dump(words_courpus, f)
        else:
            print("glove already exists. No need to compute")
```

```
glove already exists. No need to compute
```

```
In [64]: with open('glove_vectors', 'rb') as f:
             model = pickle.load(f)
             glove_words =  set(model.keys())
```

```
In [65]: # average Word2Vec
         # compute average word2vec for each review.
         avg_w2v_vectors_essay_train = []; # the avg-w2v for each sentence/review is stored in
         for sentence in tqdm(project_data_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
            avg_w2v_vectors_essay_train.append(vector)

        print(len(avg_w2v_vectors_essay_train))
        print(len(avg_w2v_vectors_essay_train[0]))
```

```
100%|| 40000/40000 [00:08<00:00, 4978.50it/s]


40000
300
```

```
In [66]: # average Word2Vec
         # compute average word2vec for each review.
         avg_w2v_vectors_essay_test = []; # the avg-w2v for each sentence/review is stored in
         for sentence in tqdm(project_data_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
             avg_w2v_vectors_essay_test.append(vector)

         print(len(avg_w2v_vectors_essay_test))
         print(len(avg_w2v_vectors_essay_test[0]))
```

```
100%|| 10000/10000 [00:02<00:00, 4857.94it/s]


10000
300
```

```
In [67]: # average Word2Vec
         # compute average word2vec for each title.
         title_avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in
         for sentence in tqdm(project_data_X_train.project_title.values): # for each review/se
             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
        title_avg_w2v_vectors_train.append(vector)

    print(len(title_avg_w2v_vectors_train))
    print(len(title_avg_w2v_vectors_train[0]))
```

```
100%|| 40000/40000 [00:00<00:00, 92412.11it/s]
```

```
40000
300
```

In [68]:
```
# average Word2Vec
# compute average word2vec for each title.
title_avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(project_data_X_test.project_title.values): # for each review/sen
    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
    title_avg_w2v_vectors_test.append(vector)

print(len(title_avg_w2v_vectors_test))
print(len(title_avg_w2v_vectors_test[0]))
```

```
100%|| 10000/10000 [00:00<00:00, 89525.09it/s]
```

```
10000
300
```

In [69]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(project_data_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_)))
essay_tfidf_words = set(tfidf_model.get_feature_names())
```

In [70]:
```
# average Word2Vec
# compute average word2vec for each review.
essay_tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored
```

```
for sentence in tqdm(project_data_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 essay_tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((s
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) #
            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_vectors_train.append(vector)

print(len(essay_tfidf_w2v_vectors_train))
print(len(essay_tfidf_w2v_vectors_train[0]))
```

```
100%|| 40000/40000 [00:56<00:00, 709.83it/s]


40000
300
```

In [71]:
```
# average Word2Vec
# compute average word2vec for each review.
essay_tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored i
for sentence in tqdm(project_data_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 essay_tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((s
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) #
            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_vectors_test.append(vector)

print(len(essay_tfidf_w2v_vectors_test))
print(len(essay_tfidf_w2v_vectors_test[0]))
```

```
100%|| 10000/10000 [00:14<00:00, 708.70it/s]


10000
300
```

In [72]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(project_data_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_)))
essay_tfidf_words = set(tfidf_model.get_feature_names())
```

In [73]:
```
# average Word2Vec
# compute average word2vec for each review.
title_tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(project_data_X_train.project_title.values): # for each review/se
    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 essay_tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((s
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) #
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    title_tfidf_w2v_vectors_train.append(vector)

print(len(title_tfidf_w2v_vectors_train))
print(len(title_tfidf_w2v_vectors_train[0]))
```

```
100%|| 40000/40000 [00:00<00:00, 46312.91it/s]


40000
300
```

In [74]:
```
# average Word2Vec
# compute average word2vec for each review.
title_tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored i
for sentence in tqdm(project_data_X_test.project_title.values): # for each review/sen
    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 essay_tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((s
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) #
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
```

```
            title_tfidf_w2v_vectors_test.append(vector)

        print(len(title_tfidf_w2v_vectors_test))
        print(len(title_tfidf_w2v_vectors_test[0]))
```

```
100%|| 10000/10000 [00:00<00:00, 43977.25it/s]


10000
300
```

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

2.0.1 2.4.1 Applying KNN brute force on BOW, SET 1

```
In [76]: # 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 mati
         BOW = hstack((categories_one_hot_train, sub_categories_one_hot_train,school_state_one
         print(BOW.shape)
         BOW_test= hstack((categories_one_hot_test, sub_categories_one_hot_test,school_state_o
         print(BOW_test.shape)
```

```
(40000, 12926)
(10000, 12926)
```

```
In [77]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.model_selection import GridSearchCV
         model=KNeighborsClassifier(algorithm='brute')
         a=np.arange(1,150,4)
         print(a)
         parameters = {'n_neighbors': a }
         clf = GridSearchCV(model, parameters, scoring='roc_auc',n_jobs=4,verbose=20)
         clf.fit(BOW,project_data_Y_train)
```

```
[  1   5   9  13  17  21  25  29  33  37  41  45  49  53  57  61  65  69
  73  77  81  85  89  93  97 101 105 109 113 117 121 125 129 133 137 141
 145 149]
Fitting 3 folds for each of 38 candidates, totalling 114 fits


[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   1 tasks      | elapsed:  2.1min
[Parallel(n_jobs=4)]: Done   2 tasks      | elapsed:  2.2min
[Parallel(n_jobs=4)]: Done   3 tasks      | elapsed:  2.2min
```

```
[Parallel(n_jobs=4)]: Done   4 tasks      | elapsed:  2.3min
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:  4.5min
[Parallel(n_jobs=4)]: Done   6 tasks      | elapsed:  4.5min
[Parallel(n_jobs=4)]: Done   7 tasks      | elapsed:  4.5min
[Parallel(n_jobs=4)]: Done   8 tasks      | elapsed:  4.6min
[Parallel(n_jobs=4)]: Done   9 tasks      | elapsed:  6.6min
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:  6.6min
[Parallel(n_jobs=4)]: Done  11 tasks      | elapsed:  6.6min
[Parallel(n_jobs=4)]: Done  12 tasks      | elapsed:  6.7min
[Parallel(n_jobs=4)]: Done  13 tasks      | elapsed:  8.7min
[Parallel(n_jobs=4)]: Done  14 tasks      | elapsed:  8.7min
[Parallel(n_jobs=4)]: Done  15 tasks      | elapsed:  8.8min
[Parallel(n_jobs=4)]: Done  16 tasks      | elapsed:  8.9min
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed: 10.8min
[Parallel(n_jobs=4)]: Done  18 tasks      | elapsed: 10.9min
[Parallel(n_jobs=4)]: Done  19 tasks      | elapsed: 10.9min
[Parallel(n_jobs=4)]: Done  20 tasks      | elapsed: 11.0min
[Parallel(n_jobs=4)]: Done  21 tasks      | elapsed: 13.2min
[Parallel(n_jobs=4)]: Done  22 tasks      | elapsed: 13.2min
[Parallel(n_jobs=4)]: Done  23 tasks      | elapsed: 13.3min
[Parallel(n_jobs=4)]: Done  24 tasks      | elapsed: 13.4min
[Parallel(n_jobs=4)]: Done  25 tasks      | elapsed: 15.5min
[Parallel(n_jobs=4)]: Done  26 tasks      | elapsed: 15.6min
[Parallel(n_jobs=4)]: Done  27 tasks      | elapsed: 15.7min
[Parallel(n_jobs=4)]: Done  28 tasks      | elapsed: 15.7min
[Parallel(n_jobs=4)]: Done  29 tasks      | elapsed: 17.9min
[Parallel(n_jobs=4)]: Done  30 tasks      | elapsed: 18.0min
[Parallel(n_jobs=4)]: Done  31 tasks      | elapsed: 18.0min
[Parallel(n_jobs=4)]: Done  32 tasks      | elapsed: 18.1min
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed: 20.2min
[Parallel(n_jobs=4)]: Done  34 tasks      | elapsed: 20.3min
[Parallel(n_jobs=4)]: Done  35 tasks      | elapsed: 20.4min
[Parallel(n_jobs=4)]: Done  36 tasks      | elapsed: 20.4min
[Parallel(n_jobs=4)]: Done  37 tasks      | elapsed: 22.4min
[Parallel(n_jobs=4)]: Done  38 tasks      | elapsed: 22.5min
[Parallel(n_jobs=4)]: Done  39 tasks      | elapsed: 22.6min
[Parallel(n_jobs=4)]: Done  40 tasks      | elapsed: 22.7min
[Parallel(n_jobs=4)]: Done  41 tasks      | elapsed: 24.8min
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed: 24.9min
[Parallel(n_jobs=4)]: Done  43 tasks      | elapsed: 25.0min
[Parallel(n_jobs=4)]: Done  44 tasks      | elapsed: 25.1min
[Parallel(n_jobs=4)]: Done  45 tasks      | elapsed: 27.2min
[Parallel(n_jobs=4)]: Done  46 tasks      | elapsed: 27.3min
[Parallel(n_jobs=4)]: Done  47 tasks      | elapsed: 27.3min
[Parallel(n_jobs=4)]: Done  48 tasks      | elapsed: 27.4min
[Parallel(n_jobs=4)]: Done  49 tasks      | elapsed: 29.5min
[Parallel(n_jobs=4)]: Done  50 tasks      | elapsed: 29.6min
[Parallel(n_jobs=4)]: Done  51 tasks      | elapsed: 29.7min
```

```
[Parallel(n_jobs=4)]: Done  52 tasks      | elapsed: 29.8min
[Parallel(n_jobs=4)]: Done  53 tasks      | elapsed: 31.9min
[Parallel(n_jobs=4)]: Done  54 tasks      | elapsed: 32.0min
[Parallel(n_jobs=4)]: Done  55 tasks      | elapsed: 32.0min
[Parallel(n_jobs=4)]: Done  56 tasks      | elapsed: 32.1min
[Parallel(n_jobs=4)]: Done  57 tasks      | elapsed: 34.3min
[Parallel(n_jobs=4)]: Done  58 tasks      | elapsed: 34.3min
[Parallel(n_jobs=4)]: Done  59 tasks      | elapsed: 34.4min
[Parallel(n_jobs=4)]: Done  60 tasks      | elapsed: 34.5min
[Parallel(n_jobs=4)]: Done  61 tasks      | elapsed: 36.6min
[Parallel(n_jobs=4)]: Done  62 tasks      | elapsed: 36.7min
[Parallel(n_jobs=4)]: Done  63 tasks      | elapsed: 36.7min
[Parallel(n_jobs=4)]: Done  64 tasks      | elapsed: 36.8min
[Parallel(n_jobs=4)]: Done  65 tasks      | elapsed: 39.0min
[Parallel(n_jobs=4)]: Done  66 tasks      | elapsed: 39.0min
[Parallel(n_jobs=4)]: Done  67 tasks      | elapsed: 39.1min
[Parallel(n_jobs=4)]: Done  68 tasks      | elapsed: 39.2min
[Parallel(n_jobs=4)]: Done  69 tasks      | elapsed: 41.3min
[Parallel(n_jobs=4)]: Done  70 tasks      | elapsed: 41.4min
[Parallel(n_jobs=4)]: Done  71 tasks      | elapsed: 41.5min
[Parallel(n_jobs=4)]: Done  72 tasks      | elapsed: 41.5min
[Parallel(n_jobs=4)]: Done  73 tasks      | elapsed: 43.6min
[Parallel(n_jobs=4)]: Done  74 tasks      | elapsed: 43.7min
[Parallel(n_jobs=4)]: Done  75 tasks      | elapsed: 43.7min
[Parallel(n_jobs=4)]: Done  76 tasks      | elapsed: 43.8min
[Parallel(n_jobs=4)]: Done  77 tasks      | elapsed: 46.0min
[Parallel(n_jobs=4)]: Done  78 tasks      | elapsed: 46.0min
[Parallel(n_jobs=4)]: Done  79 tasks      | elapsed: 46.1min
[Parallel(n_jobs=4)]: Done  80 tasks      | elapsed: 46.2min
[Parallel(n_jobs=4)]: Done  81 tasks      | elapsed: 48.3min
[Parallel(n_jobs=4)]: Done  82 tasks      | elapsed: 48.4min
[Parallel(n_jobs=4)]: Done  83 tasks      | elapsed: 48.5min
[Parallel(n_jobs=4)]: Done  84 tasks      | elapsed: 48.5min
[Parallel(n_jobs=4)]: Done  85 tasks      | elapsed: 50.6min
[Parallel(n_jobs=4)]: Done  86 tasks      | elapsed: 50.7min
[Parallel(n_jobs=4)]: Done  87 tasks      | elapsed: 50.8min
[Parallel(n_jobs=4)]: Done  88 tasks      | elapsed: 50.8min
[Parallel(n_jobs=4)]: Done  89 tasks      | elapsed: 52.9min
[Parallel(n_jobs=4)]: Done  90 tasks      | elapsed: 53.0min
[Parallel(n_jobs=4)]: Done  91 tasks      | elapsed: 53.0min
[Parallel(n_jobs=4)]: Done  92 tasks      | elapsed: 53.1min
[Parallel(n_jobs=4)]: Done  93 tasks      | elapsed: 55.3min
[Parallel(n_jobs=4)]: Done  94 tasks      | elapsed: 55.4min
[Parallel(n_jobs=4)]: Done  95 tasks      | elapsed: 55.4min
[Parallel(n_jobs=4)]: Done  96 tasks      | elapsed: 55.5min
[Parallel(n_jobs=4)]: Done  97 tasks      | elapsed: 57.6min
[Parallel(n_jobs=4)]: Done  98 tasks      | elapsed: 57.7min
[Parallel(n_jobs=4)]: Done  99 tasks      | elapsed: 57.8min
```

```
[Parallel(n_jobs=4)]: Done 100 tasks      | elapsed: 57.9min
[Parallel(n_jobs=4)]: Done 101 tasks      | elapsed: 60.0min
[Parallel(n_jobs=4)]: Done 102 tasks      | elapsed: 60.1min
[Parallel(n_jobs=4)]: Done 103 tasks      | elapsed: 60.2min
[Parallel(n_jobs=4)]: Done 104 tasks      | elapsed: 60.3min
[Parallel(n_jobs=4)]: Done 105 tasks      | elapsed: 62.3min
[Parallel(n_jobs=4)]: Done 106 tasks      | elapsed: 62.4min
[Parallel(n_jobs=4)]: Done 107 tasks      | elapsed: 62.5min
[Parallel(n_jobs=4)]: Done 114 out of 114 | elapsed: 66.0min finished
```

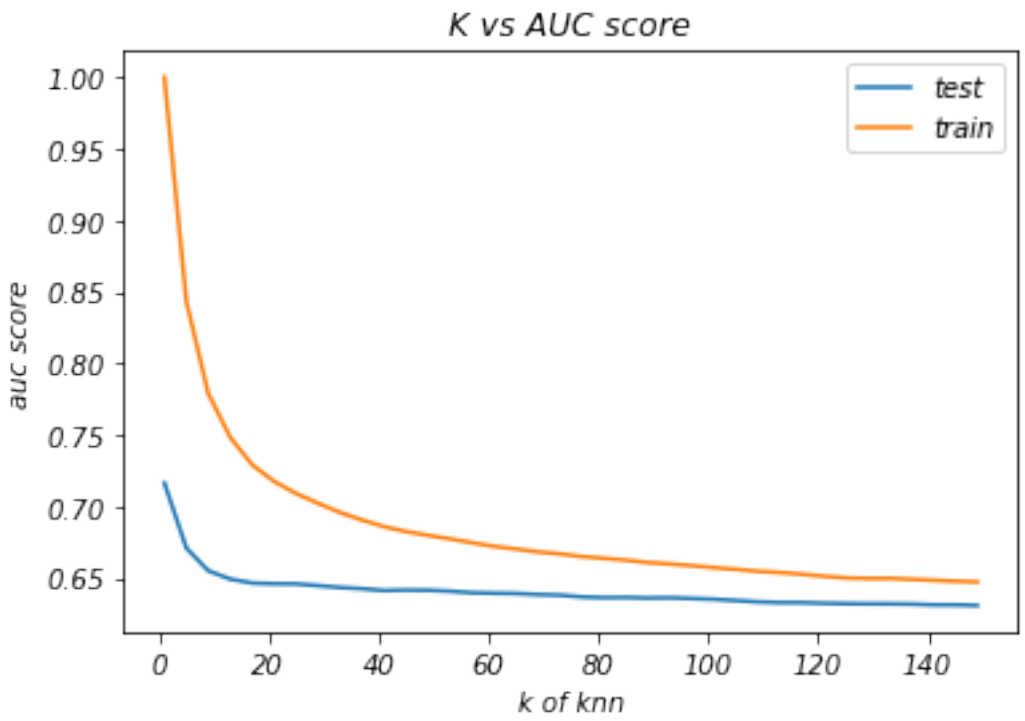
```
Out[77]: GridSearchCV(cv='warn', error_score='raise-deprecating',
                estimator=KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkows
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform'),
                fit_params=None, iid='warn', n_jobs=4,
                param_grid={'n_neighbors': array([  1,   5,   9,  13,  17,  21,  25,  29,  33,
                 53,  57,  61,  65,  69,  73,  77,  81,  85,  89,  93,  97, 101,
                105, 109, 113, 117, 121, 125, 129, 133, 137, 141, 145, 149])},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='roc_auc', verbose=20)
```

```
In [78]: clf.best_score_
         k=a
         auc_cv=clf.cv_results_['mean_test_score']
         auc_train=clf.cv_results_['mean_train_score']
         x=np.argsort(auc_cv)
         optimal_value=k[x[-1]]
         print("optimal value is: ",optimal_value)
```

```
optimal value is:  1
```

```
In [79]: k=a
         auc_cv=clf.cv_results_['mean_test_score']
         auc_train=clf.cv_results_['mean_train_score']
         plt.plot(k,auc_cv)
         plt.plot(k,auc_train)
         plt.title('K vs AUC score')
         plt.xlabel('k of knn')
         plt.ylabel('auc score')
         plt.legend({"test":"","train":""})
```

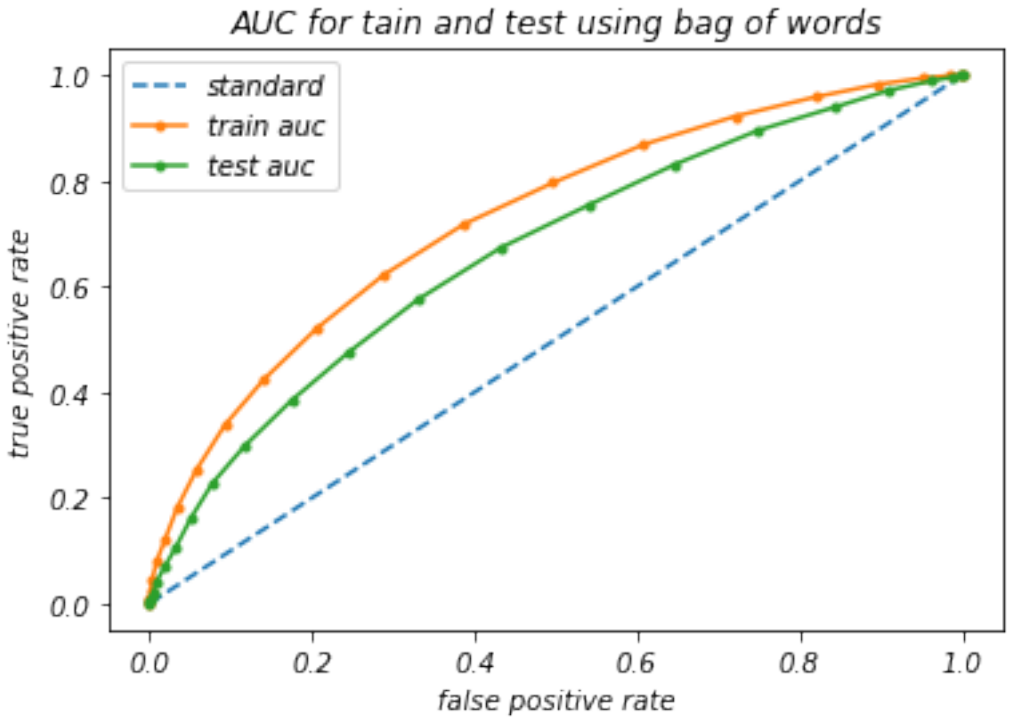
```
Out[79]: <matplotlib.legend.Legend at 0x1fb0af6d400>
```




```
In [80]: from sklearn.neighbors import KNeighborsClassifier
         model=KNeighborsClassifier(n_neighbors=21,algorithm='brute',n_jobs=4)
         model.fit(BOW,project_data_Y_train)

Out[80]: KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=4, n_neighbors=21, p=2,
                    weights='uniform')

In [82]: #https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classi
         from sklearn.metrics import roc_curve
         from sklearn.metrics import roc_auc_score
         from tqdm import tqdm

         probs_test = model.predict_proba(BOW_test.todense())
         # keep probabilities for the positive outcome only
         probs_test = probs_test[:, 1]
         auc_test = roc_auc_score(project_data_Y_test, probs_test)
         print('AUC: %.3f' % auc_test)
         fpr, tpr, thresholds = roc_curve(project_data_Y_test, probs_test)

         probs_train = model.predict_proba(BOW)
         # keep probabilities for the positive outcome only
         probs_train = probs_train[:, 1]
         auc_train = roc_auc_score(project_data_Y_train, probs_train)
```

```
print('AUC: %.3f' % auc_train)
fpr1, tpr1, thresholds1 = roc_curve(project_data_Y_train, probs_train)


plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr1, tpr1, marker='.')
plt.plot(fpr, tpr, marker='.')

plt.legend({"standard":"","train auc":"","test auc":""})
plt.title("AUC for tain and test using bag of words")
plt.xlabel("false positive rate")
plt.ylabel("true positive rate")
plt.show()
```

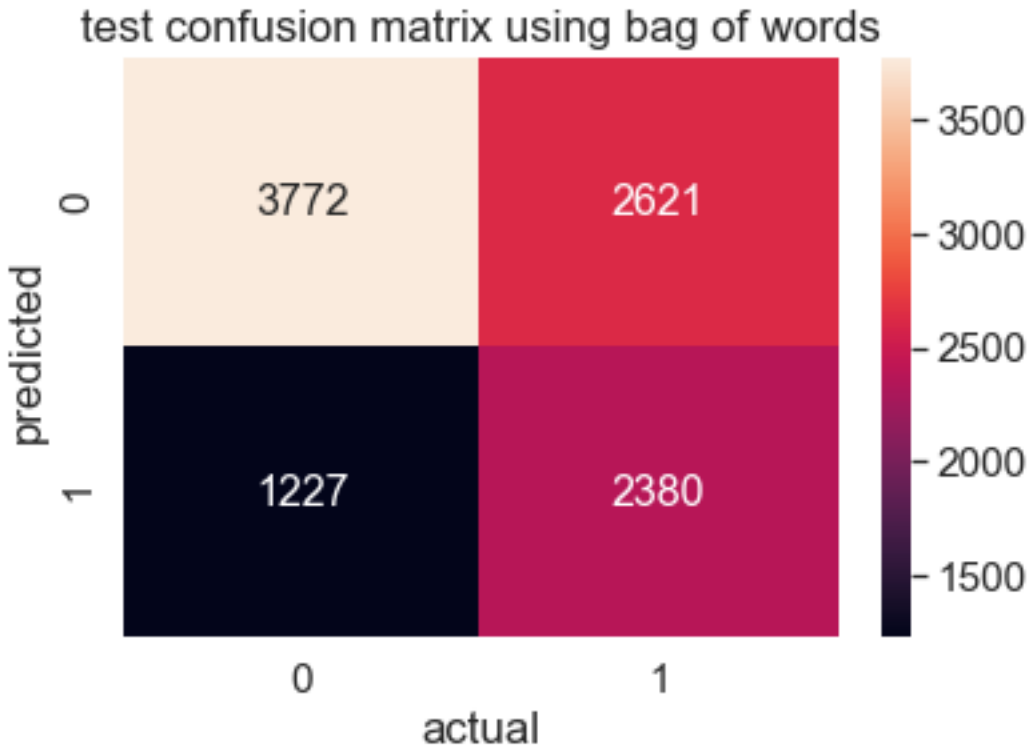
```
AUC: 0.667
AUC: 0.729
```



In [83]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#compute confudion matrix values and plot
from sklearn.metrics import confusion_matrix
predicted_bow_test=model.predict(BOW_test)
tn, fp, fn, tp = confusion_matrix(project_data_Y_test,predicted_bow_test).ravel()
print(tn, fp, fn, tp)
```

```
        print("true positive rate",(tp/(tp+fn)))
        print("true negaitive rate",(tn/(tn+fp)))
        matrix=[[tn,fn],[fp,tp]]
        print(matrix)
        df_cm = pd.DataFrame(matrix, range(2),
                             range(2))
        #plt.figure(figsize = (10,7))
        sns.set(font_scale=1.4)#for label size
        sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')# font size
        plt.title("test confusion matrix using bag of words")
        plt.xlabel("actual")
        plt.ylabel("predicted")
        plt.show()
```

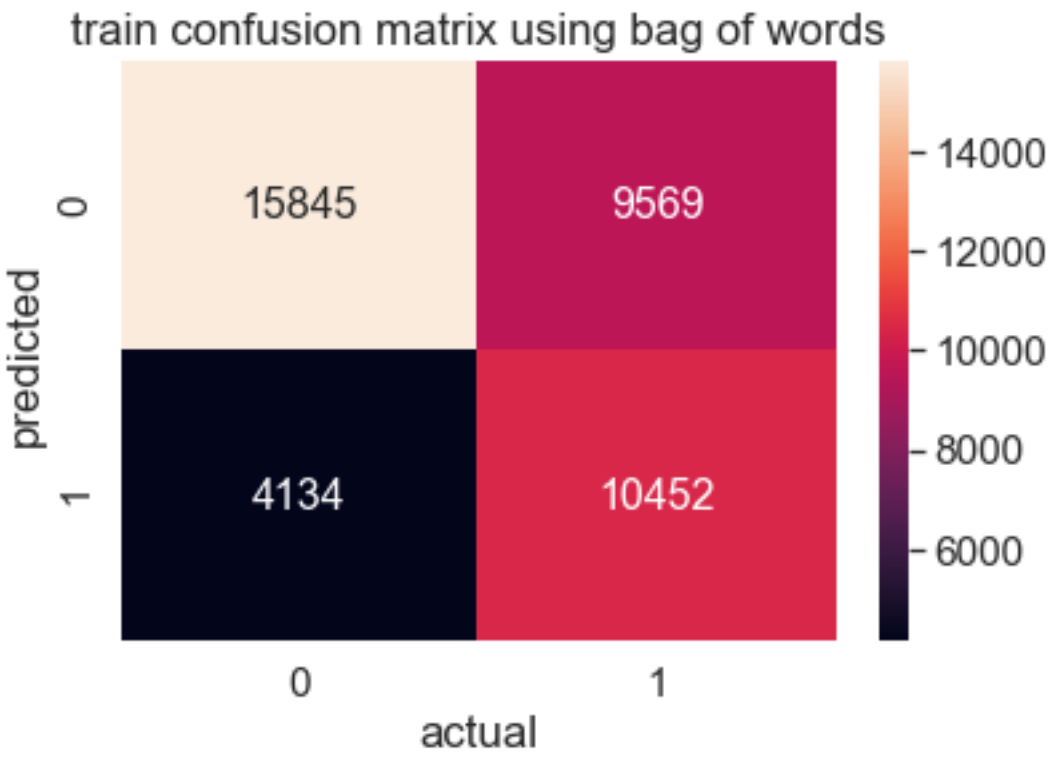
```
3772 1227 2621 2380
true positive rate 0.47590481903619275
true negaitive rate 0.7545509101820365
[[3772, 2621], [1227, 2380]]
```



In [84]:
```
        #https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
        #compute confudion matrix values and plot
        from sklearn.metrics import confusion_matrix
```

```
predicted_bow_test=model.predict(BOW)
tn, fp, fn, tp = confusion_matrix(project_data_Y_train,predicted_bow_test).ravel()
print(tn, fp, fn, tp)
print("true positive rate",(tp/(tp+fn)))
print("true negaitive rate",(tn/(tn+fp)))
matrix=[[tn,fn],[fp,tp]]
print(matrix)
df_cm = pd.DataFrame(matrix, range(2),
                     range(2))
#plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')# font size
plt.title("train confusion matrix using bag of words")
plt.xlabel("actual")
plt.ylabel("predicted")
plt.show()
```

```
15845 4134 9569 10452
true positive rate 0.5220518455621598
true negaitive rate 0.7930827368737174
[[15845, 9569], [4134, 10452]]
```



2.0.2 2.4.2 Applying KNN brute force on TFIDF, SET 2

```
In [85]: # 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 mati
         TFIDF = hstack((categories_one_hot_train, sub_categories_one_hot_train,school_state_o
         print(TFIDF.shape)
         TFIDF_test = hstack((categories_one_hot_test, sub_categories_one_hot_test,school_stat
         print(TFIDF_test.shape)
```

```
(40000, 12926)
(10000, 12926)
```

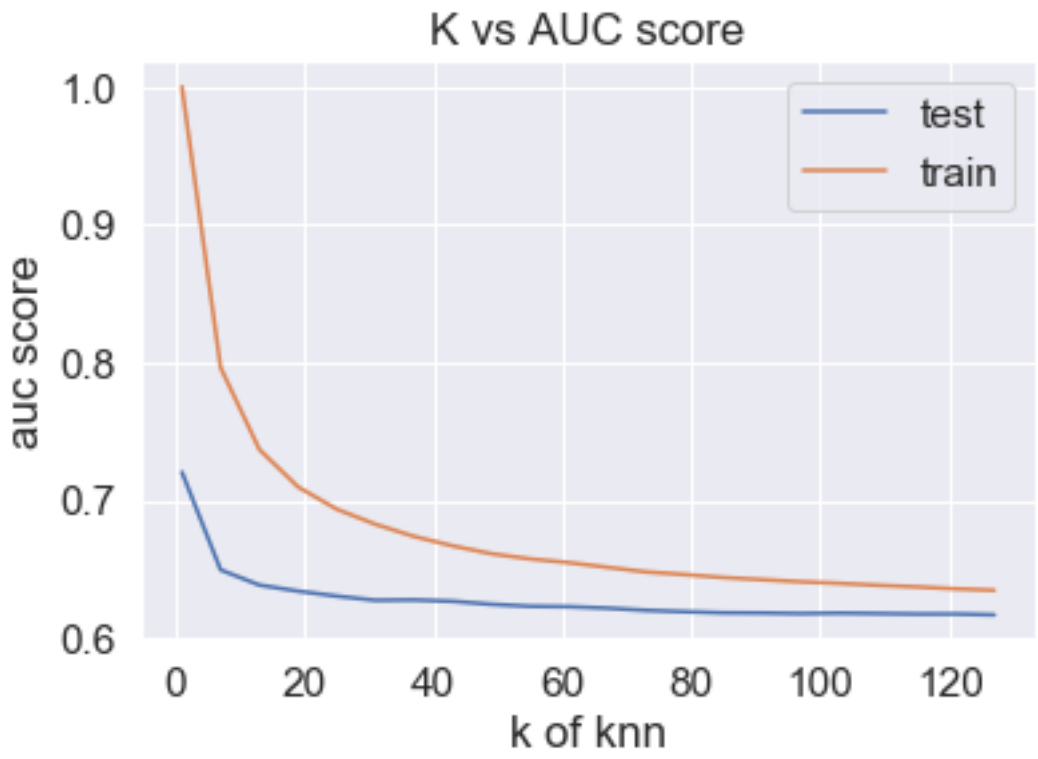
```
In [86]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.model_selection import GridSearchCV
         model=KNeighborsClassifier(algorithm='brute')
         a=np.arange(1,132,6)
         print(a)
         parameters = {'n_neighbors': a }
         clf = GridSearchCV(model, parameters, scoring='roc_auc',n_jobs=4,verbose=10)
         clf.fit(TFIDF,project_data_Y_train)
```

```
[  1   7  13  19  25  31  37  43  49  55  61  67  73  79  85  91  97 103
 109 115 121 127]
Fitting 3 folds for each of 22 candidates, totalling 66 fits


[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:  4.7min
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:  7.2min
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed: 11.9min
[Parallel(n_jobs=4)]: Done  24 tasks      | elapsed: 14.2min
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed: 20.7min
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed: 25.2min
[Parallel(n_jobs=4)]: Done  53 tasks      | elapsed: 31.9min
[Parallel(n_jobs=4)]: Done  66 out of  66 | elapsed: 38.0min remaining:    0.0s
[Parallel(n_jobs=4)]: Done  66 out of  66 | elapsed: 38.0min finished
```

```
Out[86]: GridSearchCV(cv='warn', error_score='raise-deprecating',
                estimator=KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkows
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform'),
                fit_params=None, iid='warn', n_jobs=4,
                param_grid={'n_neighbors': array([  1,   7,  13,  19,  25,  31,  37,  43,  49,
                 79,  85,  91,  97, 103, 109, 115, 121, 127])},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='roc_auc', verbose=10)
```

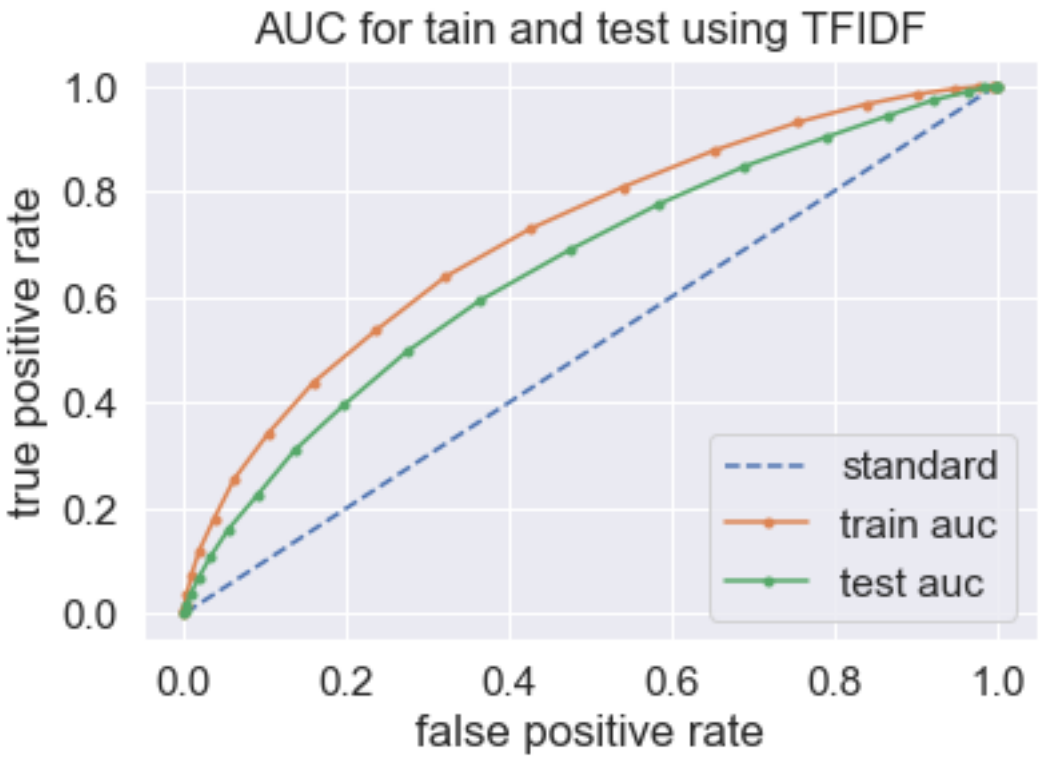
```
In [87]: clf.best_score_
         k=a
         auc_cv=clf.cv_results_['mean_test_score']
         auc_train=clf.cv_results_['mean_train_score']
         x=np.argsort(auc_cv)
         optimal_value=k[x[-1]]
         print("optimal value is: ",optimal_value)

optimal value is:  1
```

```
In [88]: plt.plot(k,auc_cv)
         plt.plot(k,auc_train)
         plt.title('K vs AUC score')
         plt.xlabel('k of knn')
         plt.ylabel('auc score')
         plt.legend({"test":"","train":""})

Out[88]: <matplotlib.legend.Legend at 0x1fb0ec2e898>
```




```
In [89]: from sklearn.neighbors import KNeighborsClassifier
         model=KNeighborsClassifier(n_neighbors=21,algorithm='brute',n_jobs=4)
         model.fit(TFIDF,project_data_Y_train)
```

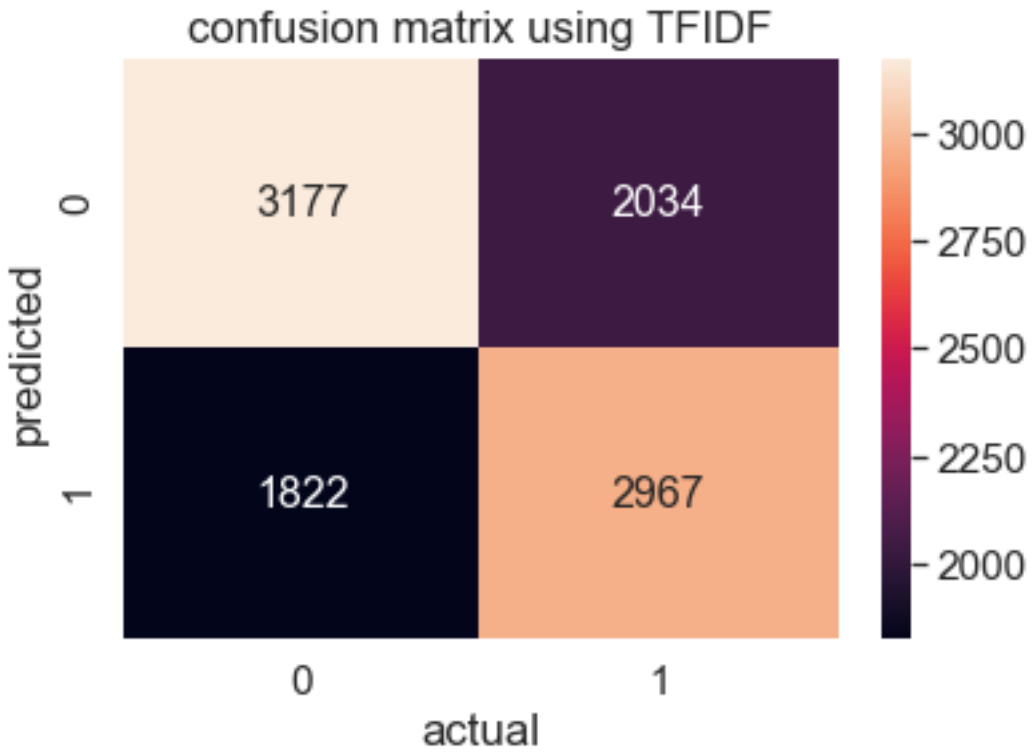
```
Out[89]: KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=4, n_neighbors=21, p=2,
                              weights='uniform')

In [90]: #https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classi
         from sklearn.metrics import roc_curve
         from sklearn.metrics import roc_auc_score

         probs_test = model.predict_proba(TFIDF_test)
         # keep probabilities for the positive outcome only
         probs_test = probs_test[:, 1]
         auc_test = roc_auc_score(project_data_Y_test, probs_test)
         print('AUC: %.3f' % auc_test)
         fpr, tpr, thresholds = roc_curve(project_data_Y_test, probs_test)

         probs_train = model.predict_proba(TFIDF)
         # keep probabilities for the positive outcome only
         probs_train = probs_train[:, 1]
         auc_train = roc_auc_score(project_data_Y_train, probs_train)
         print('AUC: %.3f' % auc_train)
         fpr1, tpr1, thresholds1 = roc_curve(project_data_Y_train, probs_train)


         plt.plot([0, 1], [0, 1], linestyle='--')
         plt.plot(fpr1, tpr1, marker='.')
         plt.plot(fpr, tpr, marker='.')

         plt.legend({"standard":"","train auc":"","test auc":""})
         plt.title("AUC for tain and test using TFIDF")
         plt.xlabel("false positive rate")
         plt.ylabel("true positive rate")
         plt.show()

AUC: 0.654
AUC: 0.716
```



In [91]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#compute confudion matrix values and plot
from sklearn.metrics import confusion_matrix
predicted_bow_test=model.predict(TFIDF_test)
tn, fp, fn, tp = confusion_matrix(project_data_Y_test,predicted_bow_test).ravel()
print(tn, fp, fn, tp)
print("true positive rate",(tp/(tp+fn)))
print("true negaitive rate",(tn/(tn+fp)))

matrix=[[tn,fn],[fp,tp]]
print(matrix)
df_cm = pd.DataFrame(matrix, range(2),
                     range(2))
#plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')# font size
plt.title("confusion matrix using TFIDF")
plt.xlabel("actual")
plt.ylabel("predicted")
plt.show()
```

```
3177 1822 2034 2967
true positive rate 0.5932813437312537
```

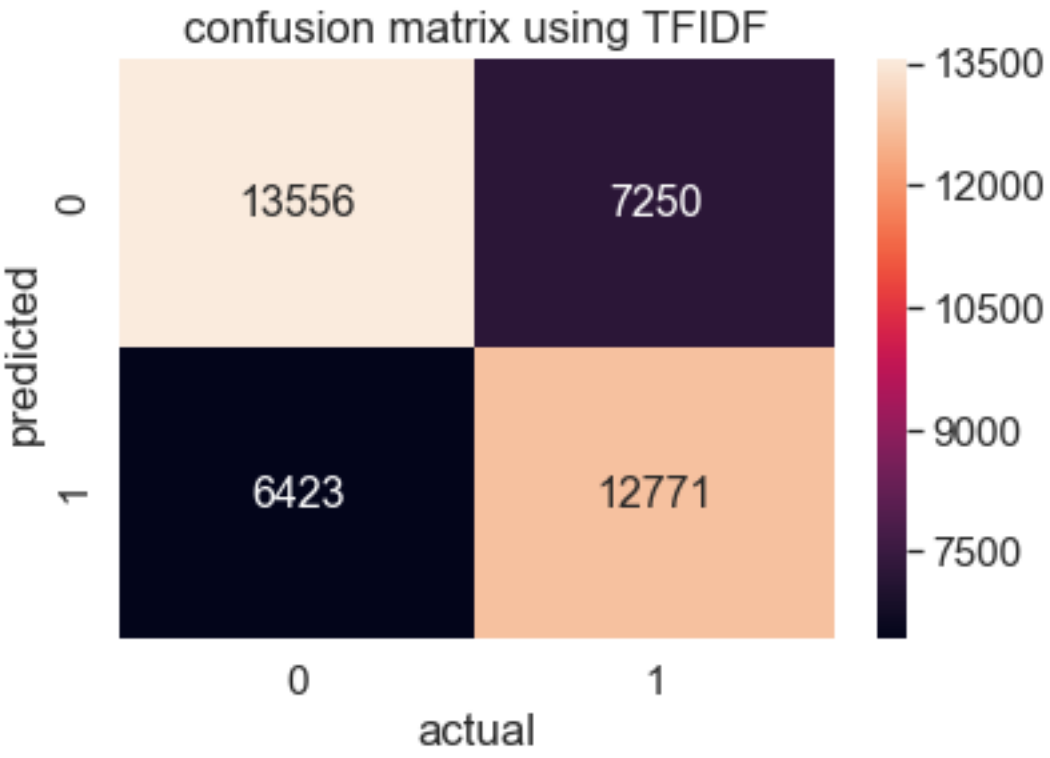
```
true negaitive rate 0.6355271054210843
[[3177, 2034], [1822, 2967]]
```



In [92]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#compute confudion matrix values and plot
from sklearn.metrics import confusion_matrix
predicted_bow_test=model.predict(TFIDF)
tn, fp, fn, tp = confusion_matrix(project_data_Y_train,predicted_bow_test).ravel()
print(tn, fp, fn, tp)
print("true positive rate",(tp/(tp+fn)))
print("true negaitive rate",(tn/(tn+fp)))

matrix=[[tn,fn],[fp,tp]]
print(matrix)
df_cm = pd.DataFrame(matrix, range(2),
                     range(2))
#plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')# font size
plt.title("confusion matrix using TFIDF")
plt.xlabel("actual")
```

```
        plt.ylabel("predicted")
        plt.show()
```

```
13556 6423 7250 12771
true positive rate 0.6378802257629489
true negaitive rate 0.678512438059963
[[13556, 7250], [6423, 12771]]
```



2.0.3 2.4.3 Applying KNN brute force on AVG W2V, SET 3

```
In [93]: # 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 mati
         AVG_W2V = hstack((categories_one_hot_train, sub_categories_one_hot_train,school_state_
         print(AVG_W2V.shape)
         AVG_W2V_test = hstack((categories_one_hot_test, sub_categories_one_hot_test,school_sta
         print(AVG_W2V_test.shape)
```

```
(40000, 697)
(10000, 697)
```

```
In [94]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.model_selection import GridSearchCV
         model=KNeighborsClassifier(algorithm='brute')
         a=np.arange(10,200,6)
         print(a)
         parameters = {'n_neighbors': a }
         clf = GridSearchCV(model, parameters, scoring='roc_auc',n_jobs=4,verbose=30)
         clf.fit(AVG_W2V.todense()[:2500,:],project_data_Y_train.values[:2500])

[ 10  16  22  28  34  40  46  52  58  64  70  76  82  88  94 100 106 112
 118 124 130 136 142 148 154 160 166 172 178 184 190 196]
Fitting 3 folds for each of 32 candidates, totalling 96 fits


[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   1 tasks      | elapsed:    0.5s
[Parallel(n_jobs=4)]: Done   2 tasks      | elapsed:    0.5s
[Parallel(n_jobs=4)]: Done   3 tasks      | elapsed:    0.5s
[Parallel(n_jobs=4)]: Done   4 tasks      | elapsed:    0.5s
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:    0.9s
[Parallel(n_jobs=4)]: Done   6 tasks      | elapsed:    1.1s
[Parallel(n_jobs=4)]: Done   7 tasks      | elapsed:    1.2s
[Parallel(n_jobs=4)]: Done   8 tasks      | elapsed:    1.3s
[Parallel(n_jobs=4)]: Done   9 tasks      | elapsed:    1.4s
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:    1.5s
[Parallel(n_jobs=4)]: Done  11 tasks      | elapsed:    1.7s
[Parallel(n_jobs=4)]: Done  12 tasks      | elapsed:    1.8s
[Parallel(n_jobs=4)]: Done  13 tasks      | elapsed:    1.9s
[Parallel(n_jobs=4)]: Done  14 tasks      | elapsed:    2.0s
[Parallel(n_jobs=4)]: Done  15 tasks      | elapsed:    2.1s
[Parallel(n_jobs=4)]: Done  16 tasks      | elapsed:    2.3s
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed:    2.4s
[Parallel(n_jobs=4)]: Done  18 tasks      | elapsed:    2.5s
[Parallel(n_jobs=4)]: Done  19 tasks      | elapsed:    2.6s
[Parallel(n_jobs=4)]: Done  20 tasks      | elapsed:    2.8s
[Parallel(n_jobs=4)]: Done  21 tasks      | elapsed:    2.9s
[Parallel(n_jobs=4)]: Done  22 tasks      | elapsed:    3.1s
[Parallel(n_jobs=4)]: Done  23 tasks      | elapsed:    3.2s
[Parallel(n_jobs=4)]: Done  24 tasks      | elapsed:    3.3s
[Parallel(n_jobs=4)]: Done  25 tasks      | elapsed:    3.4s
[Parallel(n_jobs=4)]: Done  26 tasks      | elapsed:    3.6s
[Parallel(n_jobs=4)]: Done  27 tasks      | elapsed:    3.7s
[Parallel(n_jobs=4)]: Done  28 tasks      | elapsed:    3.8s
[Parallel(n_jobs=4)]: Done  29 tasks      | elapsed:    3.9s
[Parallel(n_jobs=4)]: Done  30 tasks      | elapsed:    4.1s
[Parallel(n_jobs=4)]: Done  31 tasks      | elapsed:    4.2s
[Parallel(n_jobs=4)]: Done  32 tasks      | elapsed:    4.3s
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:    4.5s
```

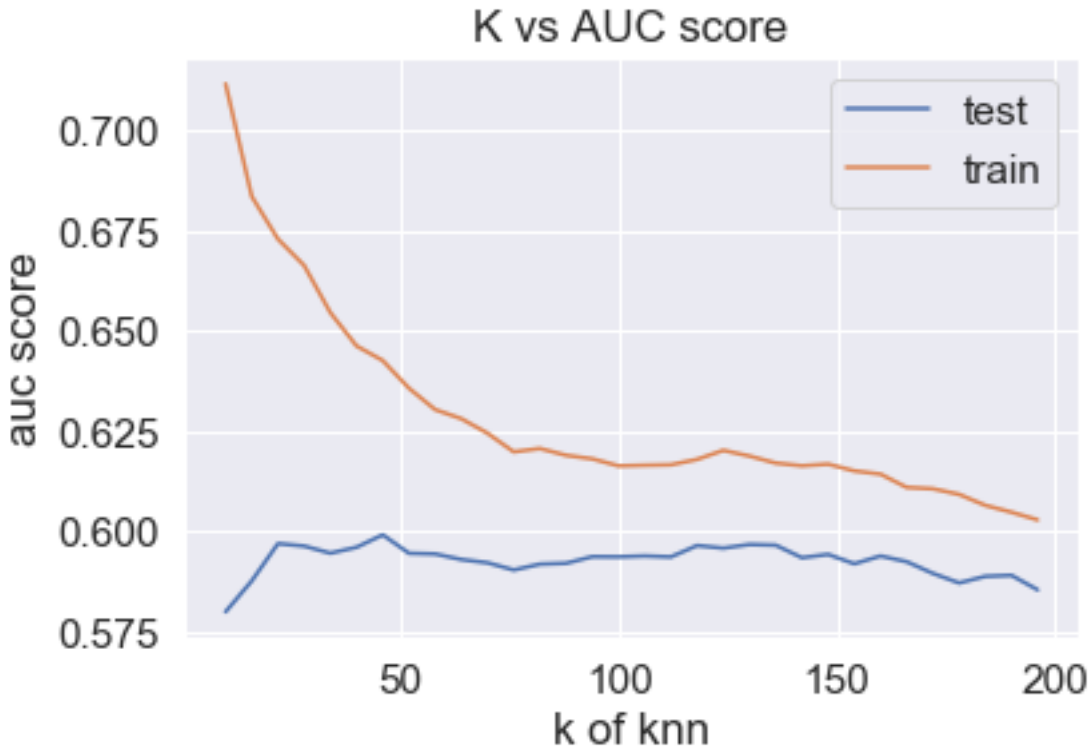
```
[Parallel(n_jobs=4)]: Done  34 tasks      | elapsed:    4.6s
[Parallel(n_jobs=4)]: Done  35 tasks      | elapsed:    4.7s
[Parallel(n_jobs=4)]: Done  36 tasks      | elapsed:    4.8s
[Parallel(n_jobs=4)]: Done  37 tasks      | elapsed:    4.9s
[Parallel(n_jobs=4)]: Done  38 tasks      | elapsed:    5.1s
[Parallel(n_jobs=4)]: Done  39 tasks      | elapsed:    5.2s
[Parallel(n_jobs=4)]: Done  40 tasks      | elapsed:    5.4s
[Parallel(n_jobs=4)]: Done  41 tasks      | elapsed:    5.5s
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    5.6s
[Parallel(n_jobs=4)]: Done  43 tasks      | elapsed:    5.8s
[Parallel(n_jobs=4)]: Done  44 tasks      | elapsed:    5.9s
[Parallel(n_jobs=4)]: Done  45 tasks      | elapsed:    6.0s
[Parallel(n_jobs=4)]: Done  46 tasks      | elapsed:    6.1s
[Parallel(n_jobs=4)]: Done  47 tasks      | elapsed:    6.3s
[Parallel(n_jobs=4)]: Done  48 tasks      | elapsed:    6.4s
[Parallel(n_jobs=4)]: Done  49 tasks      | elapsed:    6.5s
[Parallel(n_jobs=4)]: Done  50 tasks      | elapsed:    6.7s
[Parallel(n_jobs=4)]: Done  51 tasks      | elapsed:    6.8s
[Parallel(n_jobs=4)]: Done  52 tasks      | elapsed:    6.9s
[Parallel(n_jobs=4)]: Done  53 tasks      | elapsed:    7.0s
[Parallel(n_jobs=4)]: Done  54 tasks      | elapsed:    7.2s
[Parallel(n_jobs=4)]: Done  55 tasks      | elapsed:    7.3s
[Parallel(n_jobs=4)]: Done  56 tasks      | elapsed:    7.5s
[Parallel(n_jobs=4)]: Done  57 tasks      | elapsed:    7.6s
[Parallel(n_jobs=4)]: Done  58 tasks      | elapsed:    7.7s
[Parallel(n_jobs=4)]: Done  59 tasks      | elapsed:    7.8s
[Parallel(n_jobs=4)]: Done  60 tasks      | elapsed:    8.0s
[Parallel(n_jobs=4)]: Done  61 tasks      | elapsed:    8.1s
[Parallel(n_jobs=4)]: Done  62 tasks      | elapsed:    8.2s
[Parallel(n_jobs=4)]: Done  63 tasks      | elapsed:    8.3s
[Parallel(n_jobs=4)]: Done  64 tasks      | elapsed:    8.6s
[Parallel(n_jobs=4)]: Done  65 tasks      | elapsed:    8.6s
[Parallel(n_jobs=4)]: Done  66 tasks      | elapsed:    8.8s
[Parallel(n_jobs=4)]: Done  67 tasks      | elapsed:    8.8s
[Parallel(n_jobs=4)]: Done  68 tasks      | elapsed:    9.1s
[Parallel(n_jobs=4)]: Done  69 tasks      | elapsed:    9.3s
[Parallel(n_jobs=4)]: Done  70 tasks      | elapsed:    9.4s
[Parallel(n_jobs=4)]: Done  71 tasks      | elapsed:    9.5s
[Parallel(n_jobs=4)]: Done  72 tasks      | elapsed:    9.6s
[Parallel(n_jobs=4)]: Done  73 tasks      | elapsed:    9.8s
[Parallel(n_jobs=4)]: Done  74 tasks      | elapsed:    9.9s
[Parallel(n_jobs=4)]: Done  75 tasks      | elapsed:   10.1s
[Parallel(n_jobs=4)]: Done  76 tasks      | elapsed:   10.2s
[Parallel(n_jobs=4)]: Done  77 tasks      | elapsed:   10.3s
[Parallel(n_jobs=4)]: Done  78 tasks      | elapsed:   10.5s
[Parallel(n_jobs=4)]: Done  79 tasks      | elapsed:   10.6s
[Parallel(n_jobs=4)]: Done  80 tasks      | elapsed:   10.7s
[Parallel(n_jobs=4)]: Done  81 tasks      | elapsed:   10.9s
```

```
[Parallel(n_jobs=4)]: Done  82 tasks      | elapsed:   11.0s
[Parallel(n_jobs=4)]: Done  83 tasks      | elapsed:   11.2s
[Parallel(n_jobs=4)]: Done  84 tasks      | elapsed:   11.3s
[Parallel(n_jobs=4)]: Done  85 tasks      | elapsed:   11.4s
[Parallel(n_jobs=4)]: Done  86 tasks      | elapsed:   11.5s
[Parallel(n_jobs=4)]: Done  87 tasks      | elapsed:   11.7s
[Parallel(n_jobs=4)]: Done  88 tasks      | elapsed:   11.9s
[Parallel(n_jobs=4)]: Done  89 tasks      | elapsed:   12.0s
[Parallel(n_jobs=4)]: Done  93 out of  96 | elapsed:   12.5s remaining:    0.3s
[Parallel(n_jobs=4)]: Done  96 out of  96 | elapsed:   12.8s finished
```

```
Out[94]: GridSearchCV(cv='warn', error_score='raise-deprecating',
                estimator=KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkows
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform'),
                fit_params=None, iid='warn', n_jobs=4,
                param_grid={'n_neighbors': array([ 10,  16,  22,  28,  34,  40,  46,  52,  58,
                 88,  94, 100, 106, 112, 118, 124, 130, 136, 142, 148, 154, 160,
                166, 172, 178, 184, 190, 196])},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='roc_auc', verbose=30)
```

```
In [95]: clf.best_score_
         k=a
         auc_cv=clf.cv_results_['mean_test_score']
         auc_train=clf.cv_results_['mean_train_score']
         x=np.argsort(auc_cv)
         optimal_value=k[x[-1]]
         print("optimal value is: ",optimal_value)

optimal value is:  46
```

```
In [96]: plt.plot(k,auc_cv)
         plt.plot(k,auc_train)
         plt.title('K vs AUC score')
         plt.xlabel('k of knn')
         plt.ylabel('auc score')
         plt.legend({"test":"","train":""})

Out[96]: <matplotlib.legend.Legend at 0x1fb0e5131d0>
```



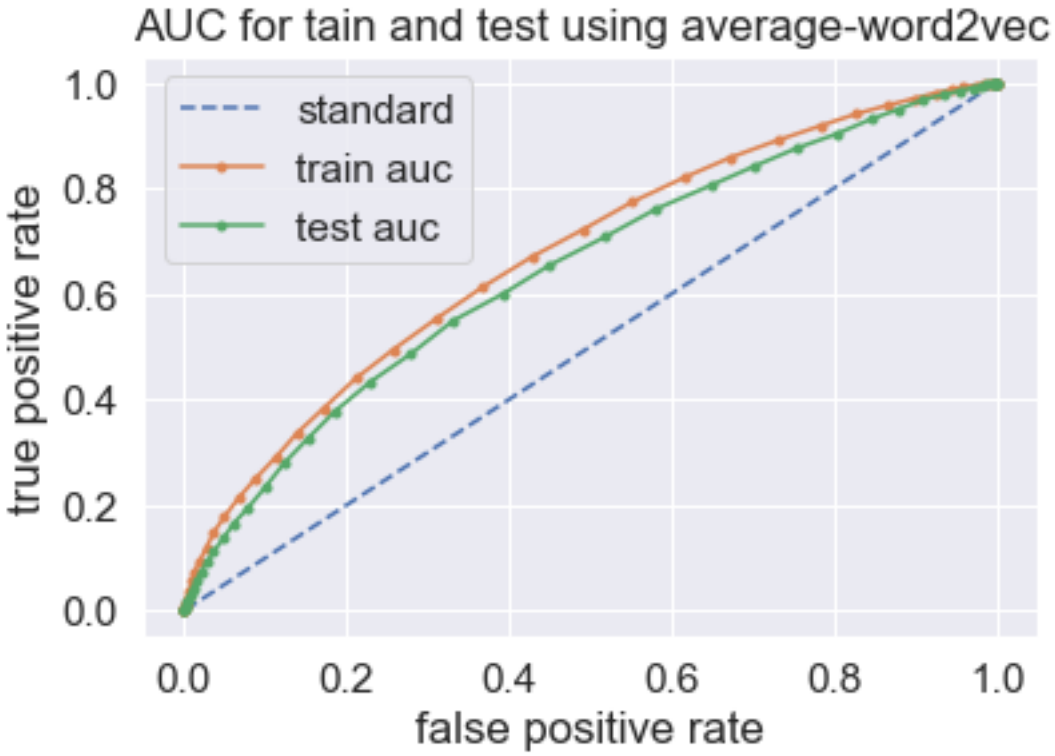
In [97]:
```
from sklearn.neighbors import KNeighborsClassifier
print()
model=KNeighborsClassifier(n_neighbors=49,algorithm='brute',n_jobs=4)
model.fit(AVG_W2V,project_data_Y_train)
```

Out[97]:
```
KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=4, n_neighbors=49, p=2,
                     weights='uniform')
```

In [98]:
```
#https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classi
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

probs_test = model.predict_proba(AVG_W2V_test)
# keep probabilities for the positive outcome only
probs_test = probs_test[:, 1]
auc_test = roc_auc_score(project_data_Y_test, probs_test)
print('AUC: %.3f' % auc_test)
fpr, tpr, thresholds = roc_curve(project_data_Y_test, probs_test)
```

```
probs_train = model.predict_proba(AVG_W2V)
# keep probabilities for the positive outcome only
probs_train = probs_train[:, 1]
auc_train = roc_auc_score(project_data_Y_train, probs_train)
print('AUC: %.3f' % auc_train)
fpr1, tpr1, thresholds1 = roc_curve(project_data_Y_train, probs_train)


plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr1, tpr1, marker='.')
plt.plot(fpr, tpr, marker='.')

plt.legend({"standard":"","train auc":"","test auc":""})
plt.title("AUC for tain and test using average-word2vec")
plt.xlabel("false positive rate")
plt.ylabel("true positive rate")
plt.show()
```

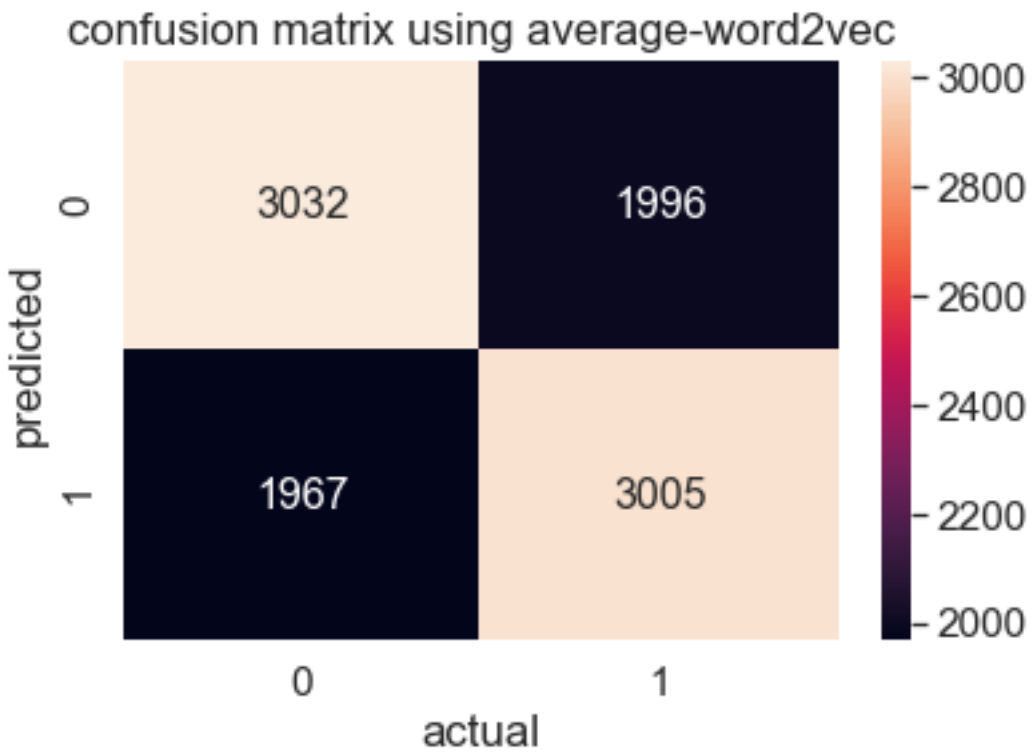
```
AUC: 0.645
AUC: 0.673
```



In [99]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#compute confudion matrix values and plot
```

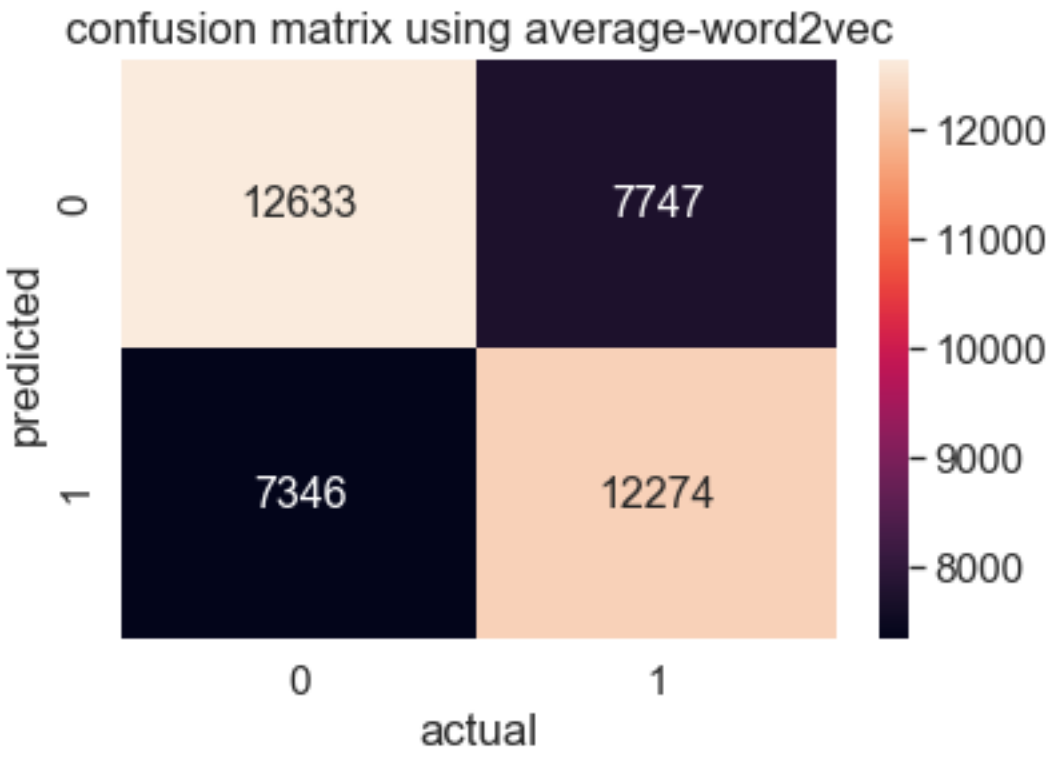
```
from sklearn.metrics import confusion_matrix
predicted_bow_test=model.predict(AVG_W2V_test)
tn, fp, fn, tp = confusion_matrix(project_data_Y_test,predicted_bow_test).ravel()
print(tn, fp, fn, tp)
print("true positive rate",(tp/(tp+fn)))
print("true negaitive rate",(tn/(tn+fp)))

matrix=[[tn,fn],[fp,tp]]
print(matrix)
df_cm = pd.DataFrame(matrix, range(2),
                     range(2))
#plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')# font size
plt.title("confusion matrix using average-word2vec")
plt.xlabel("actual")
plt.ylabel("predicted")
plt.show()
```

```
3032 1967 1996 3005
true positive rate 0.6008798240351929
true negaitive rate 0.6065213042608522
[[3032, 1996], [1967, 3005]]
```



In [100]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#compute confudion matrix values and plot
from sklearn.metrics import confusion_matrix
predicted_bow_test=model.predict(AVG_W2V)
tn, fp, fn, tp = confusion_matrix(project_data_Y_train,predicted_bow_test).ravel()
print(tn, fp, fn, tp)
print("true positive rate",(tp/(tp+fn)))
print("true negaitive rate",(tn/(tn+fp)))

matrix=[[tn,fn],[fp,tp]]
print(matrix)
df_cm = pd.DataFrame(matrix, range(2),
                     range(2))
#plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')# font size
plt.title("confusion matrix using average-word2vec")
plt.xlabel("actual")
plt.ylabel("predicted")
plt.show()
```

```
12633 7346 7747 12274
true positive rate 0.6130562908945607
true negaitive rate 0.6323139296261074
[[12633, 7747], [7346, 12274]]
```



2.0.4 2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

```
In [101]: # 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 mat
          TFIDF_W2V = hstack((categories_one_hot_train, sub_categories_one_hot_train,school_sta
          print(TFIDF_W2V.shape)
          TFIDF_W2V_test = hstack((categories_one_hot_test, sub_categories_one_hot_test,school_
          print(TFIDF_W2V_test.shape)
```

```
(40000, 697)
(10000, 697)
```

```
In [102]: from sklearn.neighbors import KNeighborsClassifier
          from sklearn.model_selection import GridSearchCV
          model=KNeighborsClassifier(algorithm='brute')
          a=np.arange(1,200,6)
          print(a)
          parameters = {'n_neighbors': a }
          clf = GridSearchCV(model, parameters, scoring='roc_auc',n_jobs=4,verbose=10)
          clf.fit(TFIDF_W2V.todense()[:2500,:],project_data_Y_train.values[:2500])
```

```
[  1   7  13  19  25  31  37  43  49  55  61  67  73  79  85  91  97 103
 109 115 121 127 133 139 145 151 157 163 169 175 181 187 193 199]
Fitting 3 folds for each of 34 candidates, totalling 102 fits


[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:    0.8s
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:    1.4s
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed:    2.2s
[Parallel(n_jobs=4)]: Done  24 tasks      | elapsed:    3.1s
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:    4.2s
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    5.3s
[Parallel(n_jobs=4)]: Done  53 tasks      | elapsed:    6.7s
[Parallel(n_jobs=4)]: Done  64 tasks      | elapsed:    8.1s
[Parallel(n_jobs=4)]: Done  77 tasks      | elapsed:    9.8s
[Parallel(n_jobs=4)]: Done  90 tasks      | elapsed:   11.5s
[Parallel(n_jobs=4)]: Done 102 out of 102 | elapsed:   12.9s finished
```

```
Out[102]: GridSearchCV(cv='warn', error_score='raise-deprecating',
                  estimator=KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkow
                      metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                      weights='uniform'),
```

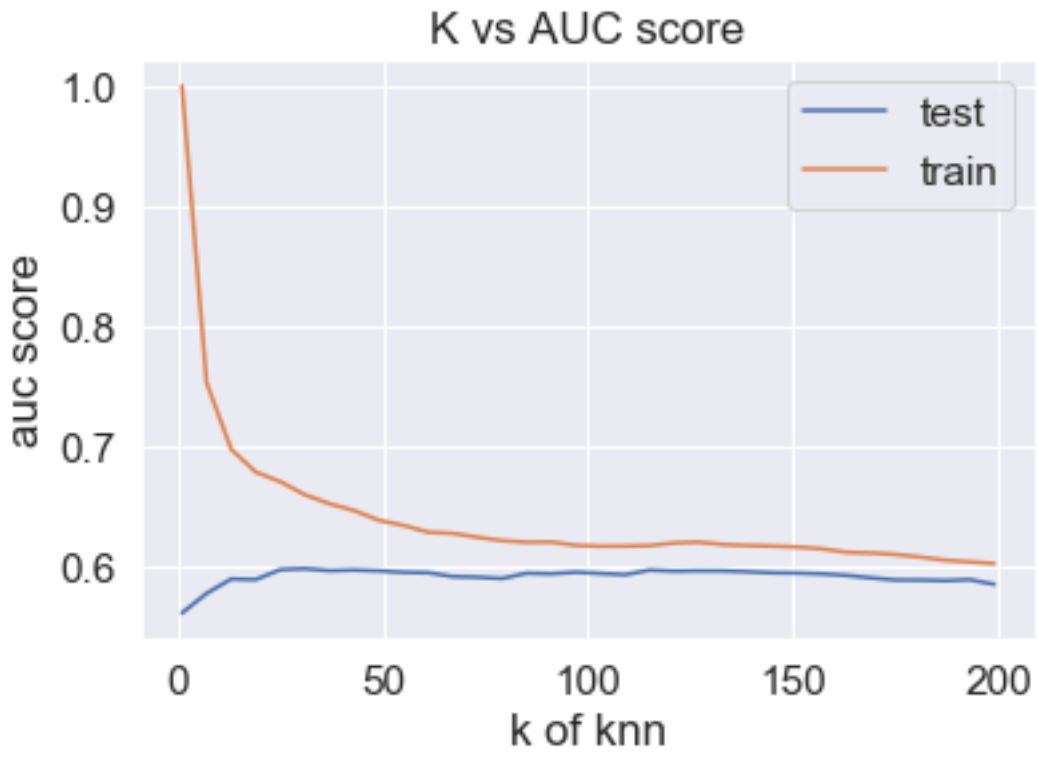
```
                fit_params=None, iid='warn', n_jobs=4,
                param_grid={'n_neighbors': array([  1,   7,  13,  19,  25,  31,  37,  43,  49
                 79,  85,  91,  97, 103, 109, 115, 121, 127, 133, 139, 145, 151,
                157, 163, 169, 175, 181, 187, 193, 199])},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='roc_auc', verbose=10)
```

In [103]:
```
clf.best_score_
k=a
auc_cv=clf.cv_results_['mean_test_score']
auc_train=clf.cv_results_['mean_train_score']
x=np.argsort(auc_cv)
optimal_value=k[x[-1]]
print("optimal value is: ",optimal_value)
```

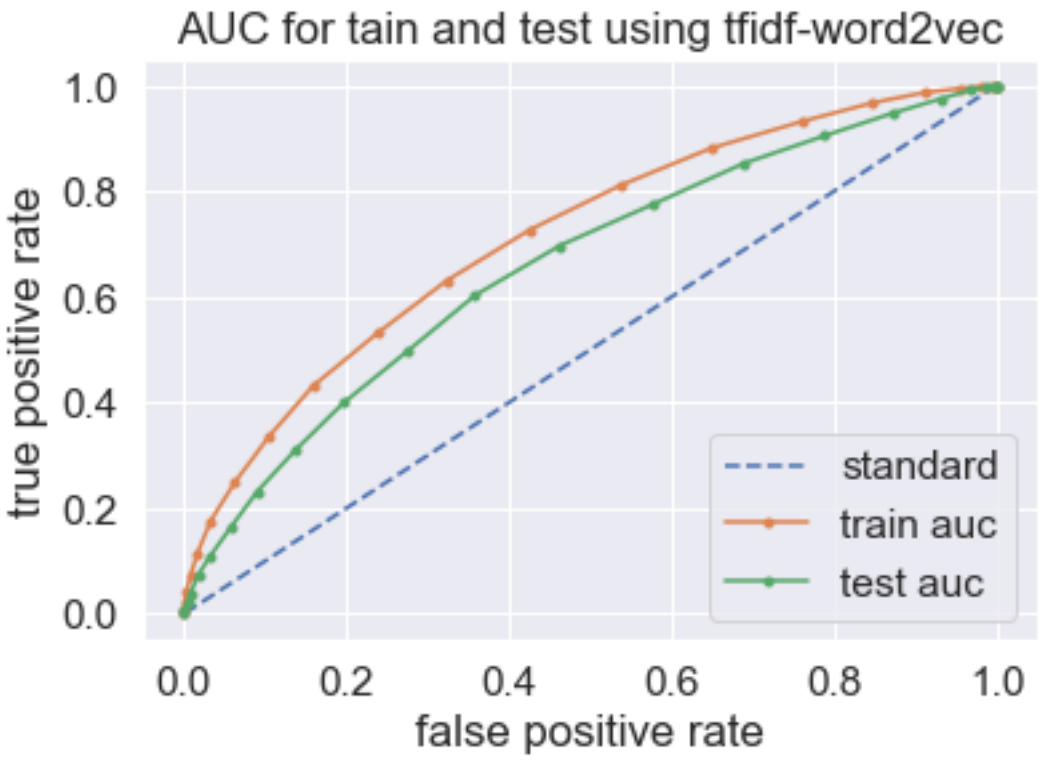
```
optimal value is:  31
```

In [104]:
```
plt.plot(k,auc_cv)
plt.plot(k,auc_train)
plt.title('K vs AUC score')
plt.xlabel('k of knn')
plt.ylabel('auc score')
plt.legend({"test":"","train":""})
```

Out[104]: <matplotlib.legend.Legend at 0x1fb0eb9c080>



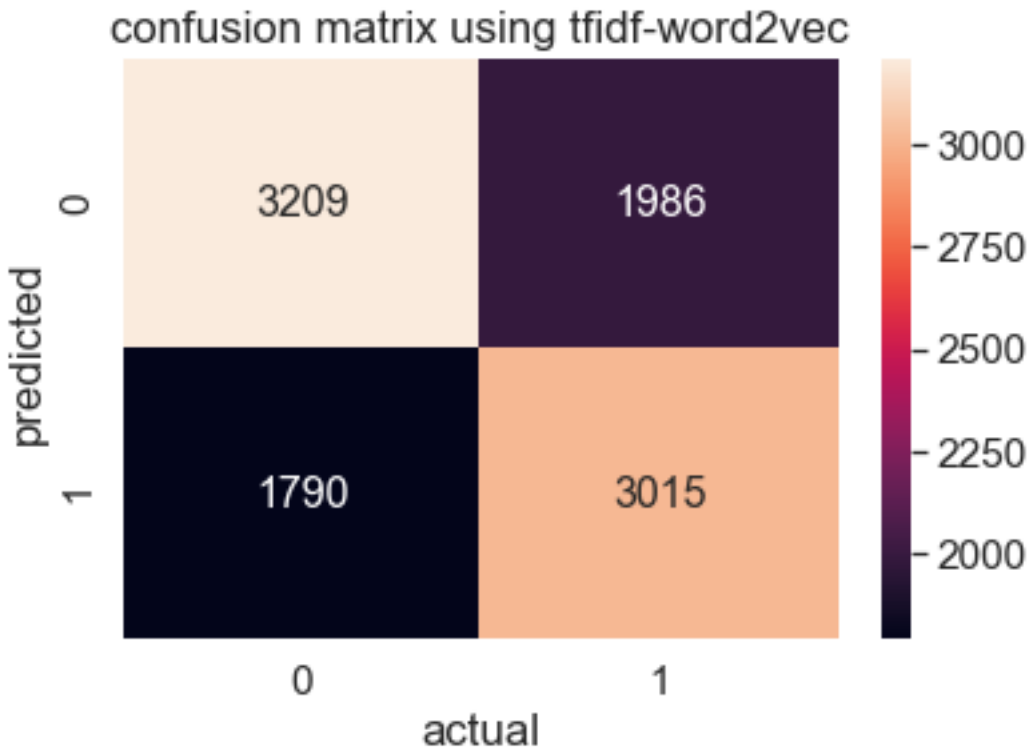
```
In [105]: from sklearn.neighbors import KNeighborsClassifier
          model=KNeighborsClassifier(n_neighbors=21,algorithm='brute',n_jobs=4)
          model.fit(TFIDF_W2V,project_data_Y_train)

Out[105]: KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=4, n_neighbors=21, p=2,
                     weights='uniform')

In [106]: #https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-class
          from sklearn.metrics import roc_curve
          from sklearn.metrics import roc_auc_score

          probs_test = model.predict_proba(TFIDF_W2V_test)
          # keep probabilities for the positive outcome only
          probs_test = probs_test[:, 1]
          auc_test = roc_auc_score(project_data_Y_test, probs_test)
          print('AUC: %.3f' % auc_test)
          fpr, tpr, thresholds = roc_curve(project_data_Y_test, probs_test)

          probs_train = model.predict_proba(TFIDF_W2V)
          # keep probabilities for the positive outcome only
          probs_train = probs_train[:, 1]
          auc_train = roc_auc_score(project_data_Y_train, probs_train)
          print('AUC: %.3f' % auc_train)
          fpr1, tpr1, thresholds1 = roc_curve(project_data_Y_train, probs_train)


          plt.plot([0, 1], [0, 1], linestyle='--')
          plt.plot(fpr1, tpr1, marker='.')
          plt.plot(fpr, tpr, marker='.')

          plt.legend({"standard":"","train auc":"","test auc":""})
          plt.title("AUC for tain and test using tfidf-word2vec")
          plt.xlabel("false positive rate")
          plt.ylabel("true positive rate")
          plt.show()

AUC: 0.660
AUC: 0.715
```



In [107]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#compute confudion matrix values and plot
from sklearn.metrics import confusion_matrix
predicted_bow_test=model.predict(TFIDF_W2V_test)
tn, fp, fn, tp = confusion_matrix(project_data_Y_test,predicted_bow_test).ravel()
print(tn, fp, fn, tp)
print("true positive rate",(tp/(tp+fn)))
print("true negaitive rate",(tn/(tn+fp)))

matrix=[[tn,fn],[fp,tp]]
print(matrix)
df_cm = pd.DataFrame(matrix, range(2),
                     range(2))
#plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')# font size
plt.title("confusion matrix using tfidf-word2vec")
plt.xlabel("actual")
plt.ylabel("predicted")
plt.show()
```

```
3209 1790 1986 3015
true positive rate 0.6028794241151769
```

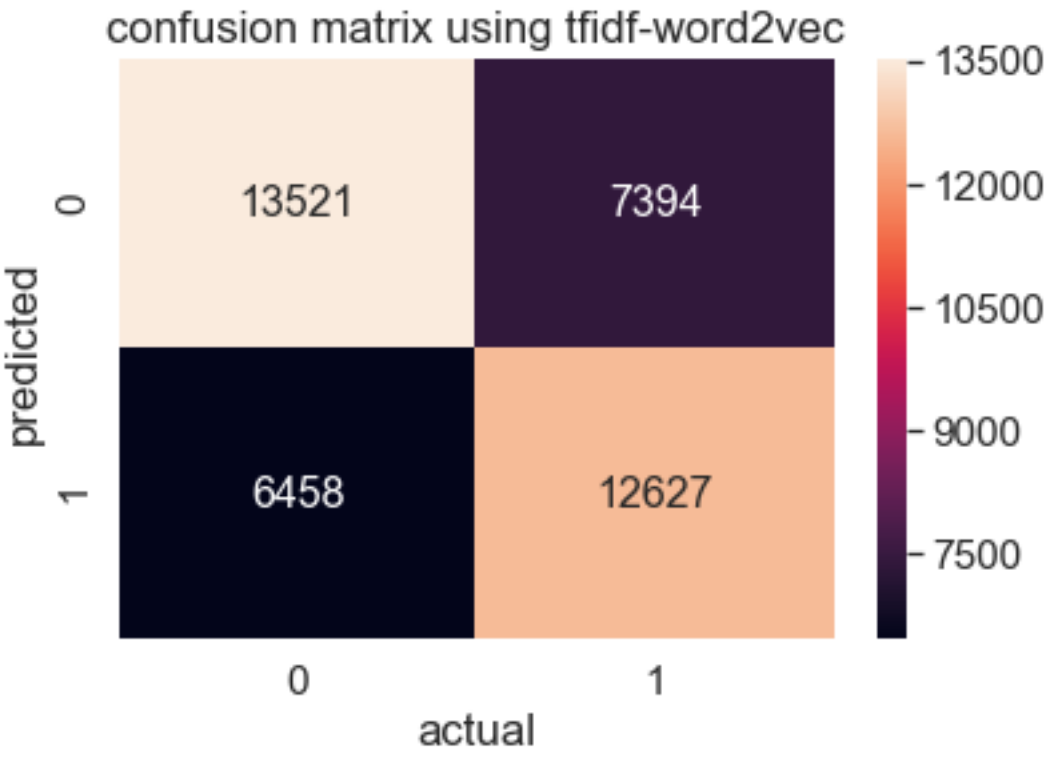
```
true negaitive rate 0.6419283856771354
[[3209, 1986], [1790, 3015]]
```



In [108]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#compute confudion matrix values and plot
from sklearn.metrics import confusion_matrix
predicted_bow_test=model.predict(TFIDF_W2V)
tn, fp, fn, tp = confusion_matrix(project_data_Y_train,predicted_bow_test).ravel()
print(tn, fp, fn, tp)
print("true positive rate",(tp/(tp+fn)))
print("true negaitive rate",(tn/(tn+fp)))

matrix=[[tn,fn],[fp,tp]]
print(matrix)
df_cm = pd.DataFrame(matrix, range(2),
                     range(2))
#plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')# font size
plt.title("confusion matrix using tfidf-word2vec")
plt.xlabel("actual")
```

```
        plt.ylabel("predicted")
        plt.show()
```

```
13521 6458 7394 12627
true positive rate 0.630687777833275
true negaitive rate 0.67676059862856
[[13521, 7394], [6458, 12627]]
```



2.5 Feature selection with `SelectKBest`

```
In [109]: # please write all the code with proper documentation, and proper titles for each su
          # go through documentations and blogs before you start coding
          # first figure out what to do, and then think about how to do.
          # reading and understanding error messages will be very much helpfull in debugging y

          # when you plot any graph make sure you use
              # a. Title, that describes your plot, this will be very helpful to the reader
              # b. Legends if needed
              # c. X-axis label
              # d. Y-axis label
```

```
In [110]: from sklearn.feature_selection import SelectKBest,chi2
          from scipy.sparse import vstack
```

```
          tfidf_total=vstack([TFIDF,TFIDF_test])
          tfidf_total=SelectKBest(chi2,k=20).fit_transform(tfidf_total,project_data_Y)
          print(tfidf_total.shape)
          print(type(tfidf_total))
```

```
(50000, 20)
<class 'scipy.sparse.csr.csr_matrix'>
```

```
In [111]: X_best_train,X_best_test,y_best_train,y_best_test=train_test_split(tfidf_total,proje
          print(X_best_train.shape)
          print(X_best_test.shape)
          print(y_best_train.shape)
          print(y_best_test.shape)
```

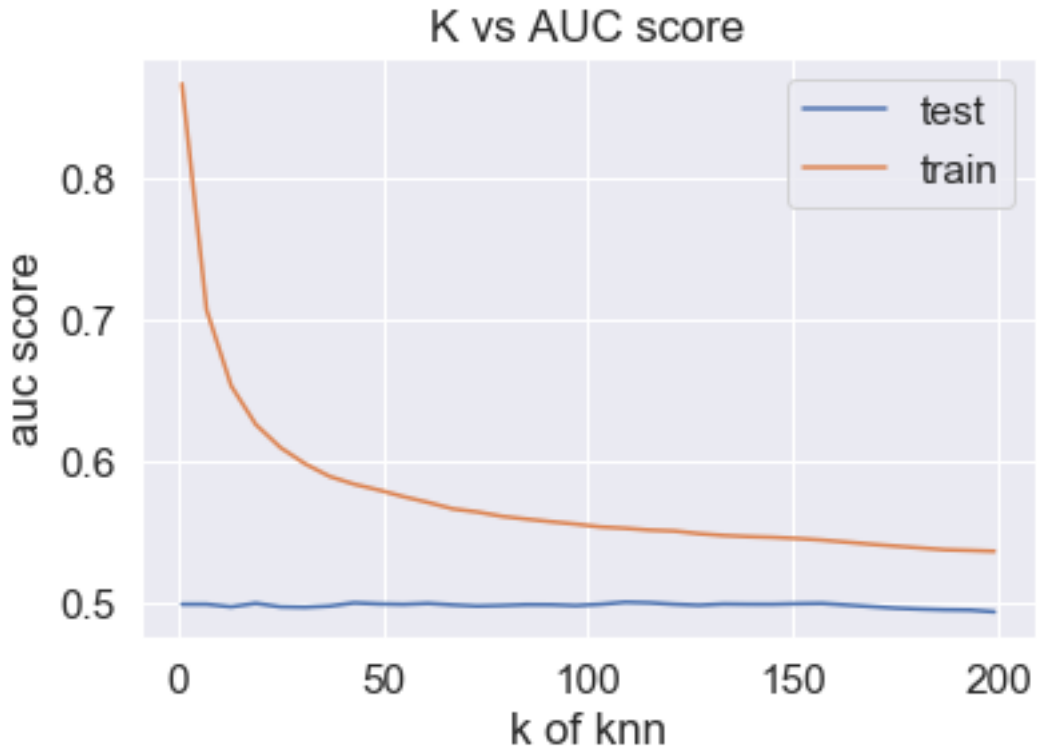
```
(37500, 20)
(12500, 20)
(37500,)
(12500,)
```

```
In [112]: from sklearn.neighbors import KNeighborsClassifier
          from sklearn.model_selection import GridSearchCV
          model=KNeighborsClassifier(algorithm='brute')
          a=np.arange(1,200,6)
          print(a)
          parameters = {'n_neighbors': a }
          clf = GridSearchCV(model, parameters, scoring='roc_auc',n_jobs=4,verbose=10)
          clf.fit(X_best_train,y_best_train)
```

```
[  1   7  13  19  25  31  37  43  49  55  61  67  73  79  85  91  97 103
 109 115 121 127 133 139 145 151 157 163 169 175 181 187 193 199]
Fitting 3 folds for each of 34 candidates, totalling 102 fits
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:  1.7min
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:  2.5min
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed:  4.2min
[Parallel(n_jobs=4)]: Done  24 tasks      | elapsed:  5.1min
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:  7.5min
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:  9.4min
[Parallel(n_jobs=4)]: Done  53 tasks      | elapsed: 11.6min
[Parallel(n_jobs=4)]: Done  64 tasks      | elapsed: 13.8min
[Parallel(n_jobs=4)]: Done  77 tasks      | elapsed: 16.5min
[Parallel(n_jobs=4)]: Done  90 tasks      | elapsed: 19.9min
[Parallel(n_jobs=4)]: Done 102 out of 102 | elapsed: 22.1min finished
```

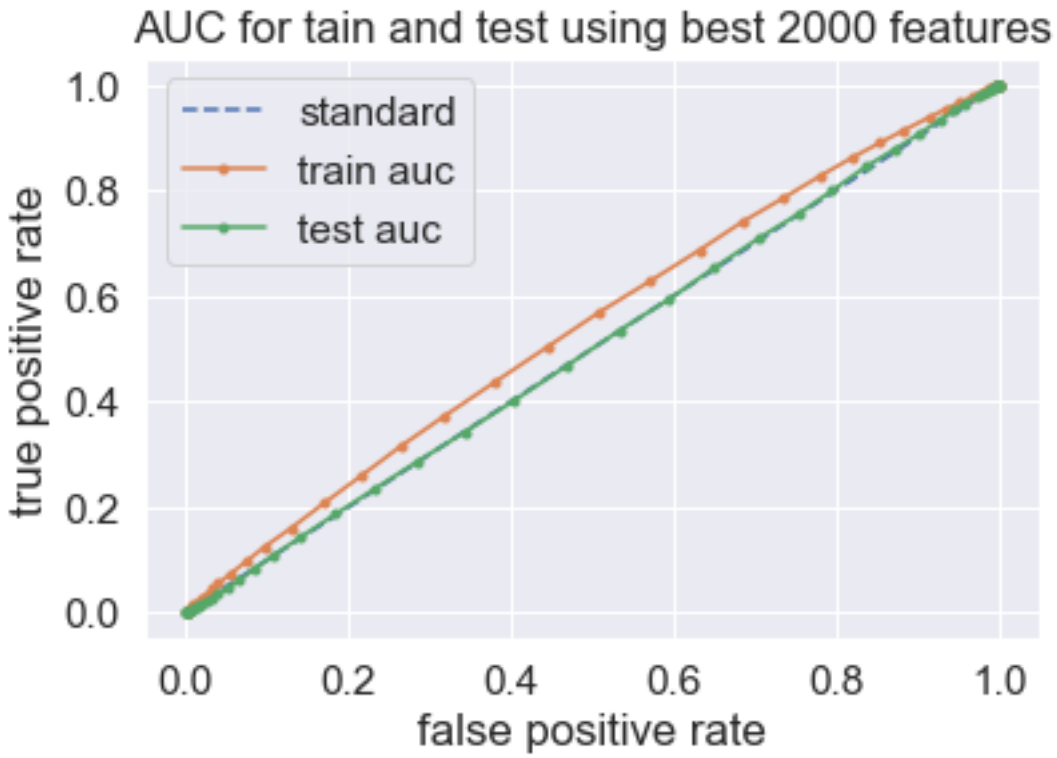
```
Out[112]: GridSearchCV(cv='warn', error_score='raise-deprecating',
                  estimator=KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkow
                      metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                      weights='uniform'),
                  fit_params=None, iid='warn', n_jobs=4,
                  param_grid={'n_neighbors': array([  1,   7,  13,  19,  25,  31,  37,  43,  49
                   79,  85,  91,  97, 103, 109, 115, 121, 127, 133, 139, 145, 151,
                  157, 163, 169, 175, 181, 187, 193, 199])},
                  pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                  scoring='roc_auc', verbose=10)

In [113]: clf.best_score_
          k=a
          auc_cv=clf.cv_results_['mean_test_score']
          auc_train=clf.cv_results_['mean_train_score']
          x=np.argsort(auc_cv)
          optimal_value=k[x[-1]]
          print("optimal value is: ",optimal_value)

optimal value is:  109


In [114]: plt.plot(k,auc_cv)
          plt.plot(k,auc_train)
          plt.title('K vs AUC score')
          plt.xlabel('k of knn')
          plt.ylabel('auc score')
          plt.legend({"test":"","train":""})

Out[114]: <matplotlib.legend.Legend at 0x1fb0ef1ef98>
```




```
In [115]: from sklearn.neighbors import KNeighborsClassifier
          model=KNeighborsClassifier(n_neighbors=175,algorithm='brute',n_jobs=4)
          model.fit(X_best_train,y_best_train)

Out[115]: KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=4, n_neighbors=175, p=2,
                     weights='uniform')

In [116]: #https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-class
          from sklearn.metrics import roc_curve
          from sklearn.metrics import roc_auc_score

          probs_test = model.predict_proba(X_best_test)
          # keep probabilities for the positive outcome only
          probs_test = probs_test[:, 1]
          auc_test = roc_auc_score(y_best_test, probs_test)
          print('AUC: %.3f' % auc_test)
          fpr, tpr, thresholds = roc_curve(y_best_test, probs_test)

          probs_train = model.predict_proba(X_best_train)
          # keep probabilities for the positive outcome only
          probs_train = probs_train[:, 1]
          auc_train = roc_auc_score(y_best_train, probs_train)
```

```
print('AUC: %.3f' % auc_train)
fpr1, tpr1, thresholds1 = roc_curve(y_best_train, probs_train)


plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr1, tpr1, marker='.')
plt.plot(fpr, tpr, marker='.')

plt.legend({"standard":"","train auc":"","test auc":""})
plt.title("AUC for tain and test using best 2000 features")
plt.xlabel("false positive rate")
plt.ylabel("true positive rate")
plt.show()
```

```
AUC: 0.501
AUC: 0.543
```

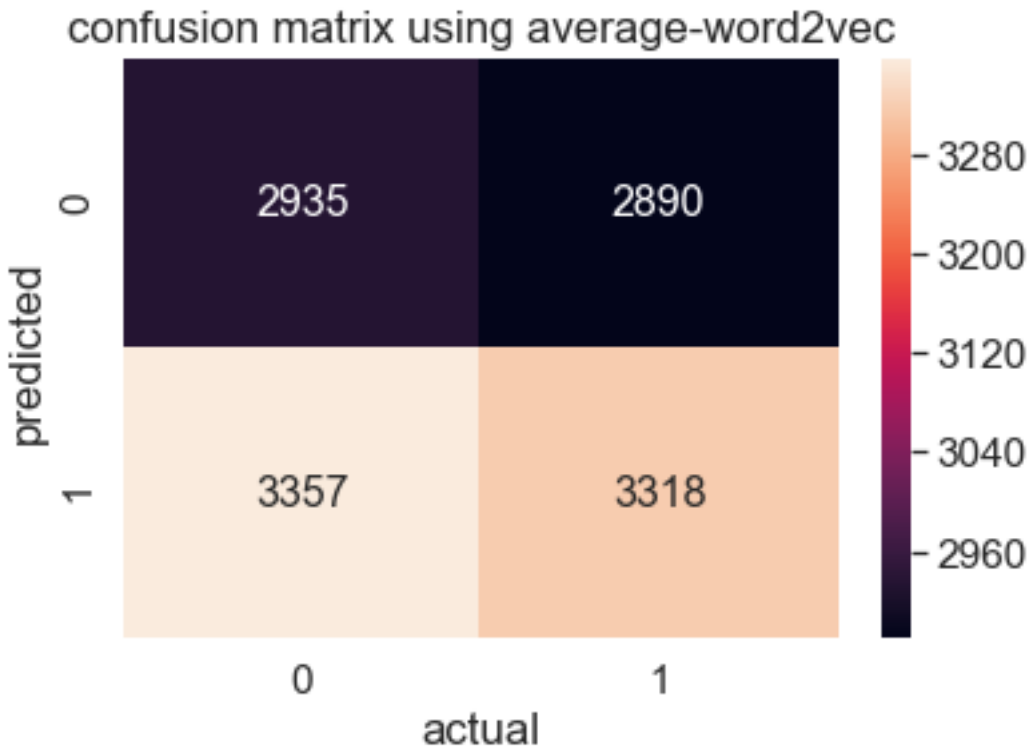


2.0.5 Note: This is worst than standard. Not acceptable

```
In [117]: #https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
          #compute confudion matrix values and plot
          from sklearn.metrics import confusion_matrix
```

```
predicted_bow_test=model.predict(X_best_test)
tn, fp, fn, tp = confusion_matrix(y_best_test,predicted_bow_test).ravel()
print(tn, fp, fn, tp)
print("true positive rate",(tp/(tp+fn)))
print("true negaitive rate",(tn/(tn+fp)))


matrix=[[tn,fn],[fp,tp]]
print(matrix)
df_cm = pd.DataFrame(matrix, range(2),
                     range(2))
#plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')# font size
plt.title("confusion matrix using average-word2vec")
plt.xlabel("actual")
plt.ylabel("predicted")
plt.show()
```

```
2935 3357 2890 3318
true positive rate 0.5344716494845361
true negaitive rate 0.4664653528289892
[[2935, 2890], [3357, 3318]]
```



3. Conclusions

```
In [119]: from prettytable import PrettyTable
          x = PrettyTable()
          x.field_names = ["Vectorizer", "Model","Under Sampling","k", "AUC"]
          x.add_row(["BAG of words", "KNN",  True, 21, 0.667])
          x.add_row(["TFIDF", "KNN" , True , 21, 0.654])
          x.add_row(["Average W2V", "KNN",  True,49, 0.645])
          x.add_row(["TFIDF W2V", "KNN" , True, 21, 0.660])
          x.add_row(["2000 best features of TFIDF", "KNN" , True, 175, 0.501])
          x.border=True
          print(x)
```

```
+-----------------------------+-------+----------------+-----+-------+
|          Vectorizer         | Model | Under Sampling |  k  |  AUC  |
+-----------------------------+-------+----------------+-----+-------+
|         BAG of words        |  KNN  |      True      |  21 | 0.667 |
|            TFIDF            |  KNN  |      True      |  21 | 0.654 |
|         Average W2V         |  KNN  |      True      |  49 | 0.645 |
|          TFIDF W2V          |  KNN  |      True      |  21 |  0.66 |
| 2000 best features of TFIDF |  KNN  |      True      | 175 | 0.501 |
+-----------------------------+-------+----------------+-----+-------+
```

```
In [ ]:
```