prabhudayala@gmail.com_7

April 29, 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 [1]: %matplotlib inline
        import warnings
```

```
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

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

```
In [3]: 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 [4]: 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[4]:         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 [5]: # 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 [6]: project_data.columns
```

```
Out[6]: Index(['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',
               'price', 'quantity'],
              dtype='object')
```

1.3 1.2 preprocessing of `project_subject_categories`

```
In [7]: catogories = list(project_data['project_subject_categories'].values)
        # remove special characters from list of strings python: https://stackoverflow.com/a/4

        # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
        # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-str
        # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyt
        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", "Warmt
        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 replace
        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 [8]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/4

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

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", "Warmt
        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 replace
        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/22898595/408403
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 [9]: # 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 [10]: project_data.head(2)

Out[10]:    Unnamed: 0       id                         teacher_id teacher_prefix  \
         0      160221  p253737  c90749f5d961ff158d4b4d1e7dc665fc           Mrs.
         1      140945  p258326  897464ce9ddc600bced1151f324dd63a            Mr.

           school_state project_submitted_datetime project_grade_category  \
         0           IN        2016-12-05 13:43:57          Grades PreK-2
         1           FL        2016-10-25 09:22:10             Grades 6-8

                                          project_title  \
         0  Educational Support for English Learners at Home
         1            Wanted: Projector for Hungry Learners

                                         project_essay_1  \
         0  My students are English learners that are work...
         1  Our students arrive to our school eager to lea...

                                         project_essay_2 project_essay_3  \
         0  \"The limits of your language are the limits o...            NaN
         1  The projector we need for our school is very c...            NaN

           project_essay_4                         project_resource_summary  \
         0             NaN  My students need opportunities to practice beg...
         1             NaN  My students need a projector to help with view...

            teacher_number_of_previously_posted_projects  project_is_approved  price  \
         0                                             0                   0  154.6
         1                                             7                   1  299.0

            quantity             clean_categories           clean_subcategories  \
         0        23            Literacy_Language                   ESL Literacy
         1         1  History_Civics Health_Sports  Civics_Government TeamSports
```

```
                                                   essay
0  My students are English learners that are work...
1  Our students arrive to our school eager to lea...
```

In [11]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [12]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

```
My students are English learners that are working on English as their second or third languages
==================================================
The 51 fifth grade students that will cycle through my classroom this year all love learning, a
==================================================
How do you remember your days of school? Was it in a sterile environment with plain walls, rows
==================================================
My kindergarten students have varied disabilities ranging from speech and language delays, cogn
==================================================
The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The g
==================================================
```

In [13]:
```
# 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 [14]: sent = decontracted(project_data['essay'].values[20000])
         print(sent)
         print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogn
==================================================
```

```
In [15]: # \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-p
         sent = sent.replace('\\r', ' ')
         sent = sent.replace('\\"', ' ')
         sent = sent.replace('\\n', ' ')
         print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogn
```

```
In [16]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
         print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays cogni
```

```
In [17]: # 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'r
                     "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', "t
                     'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'h
                     'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                     'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
                     'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
                     'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
                     'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
                     's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
                     've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
                     "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
                     "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                     'won', "won't", 'wouldn', "wouldn't"]
```

```
In [18]: # 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 not in stopwords)
            preprocessed_essays.append(sent.lower().strip())
```

```
100%|| 109248/109248 [00:45<00:00, 2406.37it/s]
```

```
In [19]: # after preprocesing
         preprocessed_essays[20000]
```

```
Out[19]: 'my kindergarten students varied disabilities ranging speech language delays cognitive
```

```
In [20]: project_data["essay"]=preprocessed_essays
```

1.4 Preprocessing of project_title

```
In [21]: 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%|| 109248/109248 [00:01<00:00, 55059.77it/s]
```

```
In [22]: print(project_data['project_title'].values[20000])
         project_data['project_title']=preprocessed_project_title
         print(project_data['project_title'].values[20000])
```

```
We Need To Move It While We Input It!
need move input
```

__ Computing Sentiment Scores__

```
In [23]: from nltk.sentiment import SentimentIntensityAnalyzer as SID
         #nltk.download('vader_lexicon')
         new_df_as_dictinary=[]
         sid=SID()
         for i in tqdm(project_data.essay.values):
             new_df_as_dictinary.append(sid.polarity_scores(i))
```

```
100%|| 109248/109248 [02:32<00:00, 715.54it/s]
```

```
In [24]: print(project_data.columns)
         print(project_data.shape)
         sentiment_score=pd.DataFrame(new_df_as_dictinary)
         print(sentiment_score.columns)
         print(sentiment_score.shape)
```

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity', 'clean_categories', 'clean_subcategories',
       'essay'],
      dtype='object')
(109248, 20)
Index(['compound', 'neg', 'neu', 'pos'], dtype='object')
(109248, 4)
```

```
In [25]: sentiment_score=pd.DataFrame(new_df_as_dictinary)
         project_data=pd.concat((project_data,sentiment_score),axis=1,ignore_index=True)
         print(project_data.shape)
```

```
(109248, 24)
```

```
In [26]: project_data.columns=['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_sta
                'project_submitted_datetime', 'project_grade_category', 'project_title',
                'project_essay_1', 'project_essay_2', 'project_essay_3',
                'project_essay_4', 'project_resource_summary',
                'teacher_number_of_previously_posted_projects', 'project_is_approved',
                'price', 'quantity', 'clean_categories', 'clean_subcategories',
                'essay','compound', 'neg', 'neu', 'pos']
```

2 Assignment 7: SVM

```
<li><strong>[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) o
    <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
</li>
<br>
<li><strong>The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalt
```

```
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </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 after you found the best hyper parameter, you need to train your model with it, and fi
<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.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>[Task-2] Apply the Support Vector Machines on these features by finding the best hy
    <ul>
<li> Consider these set of features <font color='red'> Set 5 :</font>
        <ul>
            <li><strong>school_state</strong> : categorical data</li>
            <li><strong>clean_categories</strong> : categorical data</li>
            <li><strong>clean_subcategories</strong> : categorical data</li>
            <li><strong>project_grade_category</strong> :categorical data</li>
            <li><strong>teacher_prefix</strong> : categorical data</li>
            <li><strong>quantity</strong> : numerical data</li>
            <li><strong>teacher_number_of_previously_posted_projects</strong> : numerical data
            <li><strong>price</strong> : numerical data</li>
            <li><strong>sentiment score's of each of the essay</strong> : numerical data</li>
            <li><strong>number of words in the title</strong> : numerical data</li>
            <li><strong>number of words in the combine essays</strong> : numerical data</li>
            <li><strong>Apply <a href='http://scikit-learn.org/stable/modules/generated/sklearn
        </ul>
     <br>
</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 for
    <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-leakage, 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. Support Vector Machines

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

In [27]:
```
project_data.columns
```

Out[27]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity', 'clean_categories', 'clean_subcategories', 'essay',
       'compound', 'neg', 'neu', 'pos'],
      dtype='object')
```

In [28]:
```
sampling=False
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  (109248, 24)
positive and negative counts
1    92706
0    16542
Name: project_is_approved, dtype: int64
After sampling:  (109248, 23)
```

```
In [29]: from sklearn.model_selection import train_test_split
         project_data_X_train,project_data_X_test,project_data_Y_train,project_data_Y_test=trai
```

2.2 Make Data Model Ready: encoding numerical, categorical features

2.2.1 Categorical features

```
In [30]: 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_categories
         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  (87398, 9)
Shape of matrix after one hot encodig  (21850, 9)
```

```
In [31]: 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  (87398, 30)
Shape of matrix after one hot encodig  (21850, 30)
```

In [32]:
```
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.       45800
Ms.        31168
Mr.         8519
Teacher     1898
Dr.           11
               2
Name: teacher_prefix, dtype: int64
Mrs.       11469
Ms.         7787
Mr.         2129
Teacher      462
Dr.            2
               1
Name: teacher_prefix, dtype: int64
```

In [33]:
```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=['Mrs.','Ms.','Mr.','Teacher','Dr.'], lowercas
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)
```

```
['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.']
Shape of matrix after one hot encodig  (87398, 5)
Shape of matrix after one hot encodig  (21850, 5)
```

In [34]:
```
# 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 PreK-2', 'Grades 3-5', 'Grades 6-8', 'Grades 9-12']
Shape of matrix after one hot encodig  (87398, 4)
```

```
Shape of matrix after one hot encodig  (21850, 4)
```

```
In [35]: # we use count vectorizer to convert the values into one hot encoded features
         vectorizer = CountVectorizer(vocabulary=list(project_data_X_train['school_state'].uni
         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'].
         print("Shape of matrix after one hot encodig ",school_state_one_hot_test.shape)
```

```
['NY', 'MD', 'OK', 'MA', 'CA', 'AR', 'FL', 'PA', 'SC', 'NC', 'AZ', 'MI', 'AL', 'WI', 'NV', 'UT
Shape of matrix after one hot encodig  (87398, 51)
Shape of matrix after one hot encodig  (21850, 51)
```

2.2.2 Numerical features

```
In [36]: # 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

         # 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 me
         print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.va

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

```
Mean : 298.64356770177807, Standard deviation : 368.42853396795914
```

```
In [37]: # 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.va

        # Now standardize the data with above maen and variance.
        teacher_number_of_previously_posted_projects_standardized_train = price_scalar.transf

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

```
Mean : 11.102897091466623, Standard deviation : 27.572082372998246
```

```
In [ ]:
```

2.3 Make Data Model Ready: encoding eassay, and project_title

```
In [38]: vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2),max_features=5000)
         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)
```

```
(87398, 5000)
(21850, 5000)
```

```
In [39]: # 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.valu
         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)
```

```
(87398, 2803)
(21850, 2803)
```

```
In [40]: from sklearn.feature_extraction.text import TfidfVectorizer
         vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,2),max_features=5000)
```

```
        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)
```

```
(87398, 5000)
(21850, 5000)
```

In [41]:
```
        # 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.val
        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)
```

```
(87398, 2803)
(21850, 2803)
```

In [42]:
```
        # 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 [43]: with open('glove_vectors', 'rb') as f:
             model = pickle.load(f)
             glove_words =  set(model.keys())
```

```
In [44]: # 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%|| 87398/87398 [00:20<00:00, 4287.50it/s]
```

```
87398
300
```

```
In [45]: # 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%|| 21850/21850 [00:05<00:00, 4307.36it/s]
```

```
21850
300
```

```
In [46]: # 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%|| 87398/87398 [00:00<00:00, 89207.26it/s]


87398
300
```

```
In [47]: # 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%|| 21850/21850 [00:00<00:00, 88699.31it/s]


21850
300
```

```
In [48]: # 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 [49]: # 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%|| 87398/87398 [02:27<00:00, 591.95it/s]
```

```
87398
300
```

```
In [50]: # 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%|| 21850/21850 [00:36<00:00, 598.17it/s]
```

```
21850
300
```

In [51]:
```
# 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 [52]:
```
# 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%|| 87398/87398 [00:02<00:00, 42115.01it/s]


87398
300
```

In [53]:
```
# 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%|| 21850/21850 [00:00<00:00, 41569.79it/s]


21850
300
```

2.4 Appling Support Vector Machines on different kind of featurization as mentioned in the instructions

Apply Support Vector Machines 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 instrucations

2.0.1 2.4.1 Applying LR on BOW, SET 1

In [54]:
```
# 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)
```

```
(87398, 7900)
(21850, 7900)
```

In [55]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
model1=SGDClassifier(class_weight='balanced',tol=10**-7)
a=[10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]
print(a)
b=['l1','l2']
parameters = {'alpha': a, 'penalty':b}
clf = GridSearchCV(model1, parameters,scoring='roc_auc',n_jobs=2,verbose=10)
clf.fit(BOW,project_data_Y_train)
```

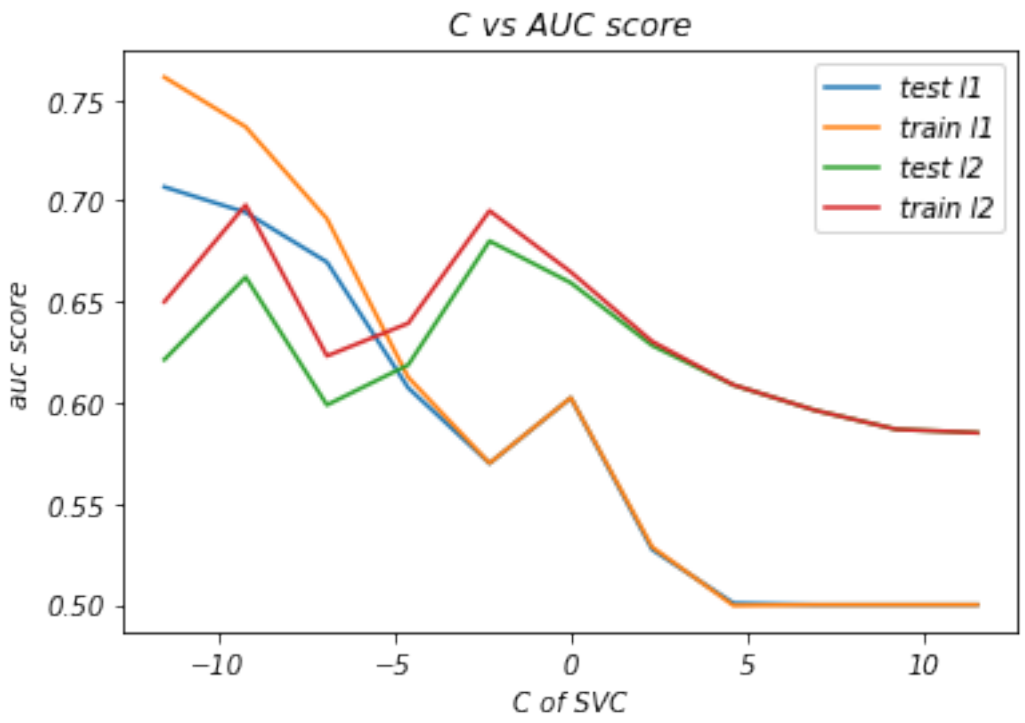
```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
Fitting 3 folds for each of 22 candidates, totalling 66 fits
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   1 tasks      | elapsed:    5.5s
[Parallel(n_jobs=2)]: Done   4 tasks      | elapsed:   11.7s
[Parallel(n_jobs=2)]: Done   9 tasks      | elapsed:   23.3s
[Parallel(n_jobs=2)]: Done  14 tasks      | elapsed:   34.4s
[Parallel(n_jobs=2)]: Done  21 tasks      | elapsed:   43.8s
[Parallel(n_jobs=2)]: Done  28 tasks      | elapsed:   51.5s
[Parallel(n_jobs=2)]: Done  37 tasks      | elapsed:  1.0min
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:  2.0min
[Parallel(n_jobs=2)]: Done  57 tasks      | elapsed:  2.1min
[Parallel(n_jobs=2)]: Done  66 out of  66 | elapsed:  2.6min finished
```

```
Out[55]: GridSearchCV(cv='warn', error_score='raise-deprecating',
                estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
                early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
                power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
                validation_fraction=0.1, verbose=0, warm_start=False),
                fit_params=None, iid='warn', n_jobs=2,
                param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='roc_auc', verbose=10)
```

```
In [56]: k=a
         auc_cv=clf.cv_results_['mean_test_score']
         auc_train=clf.cv_results_['mean_train_score']
         plt.plot(np.log(k),auc_cv[::2])
         plt.plot(np.log(k),auc_train[::2])
         plt.plot(np.log(k),auc_cv[1::2])
         plt.plot(np.log(k),auc_train[1::2])
         plt.title('C vs AUC score')
         plt.xlabel('C of SVC')
         plt.ylabel('auc score')
         plt.legend({"test l1":"","train l1":"","test l2":"","train l2":""})
```

```
Out[56]: <matplotlib.legend.Legend at 0x289a5e82ef0>
```




```
In [57]: print(k)
         print(np.log(k))
         np.exp(-4.60517019)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
[-11.51292546  -9.21034037  -6.90775528  -4.60517019  -2.30258509
   0.           2.30258509   4.60517019   6.90775528   9.21034037
  11.51292546]
```

```
Out[57]: 0.009999999959880915
```

```
In [58]: model2=SGDClassifier(alpha=0.01,penalty='l2',n_jobs=2,loss='hinge',class_weight='bala
         model2.fit(BOW,project_data_Y_train)
```

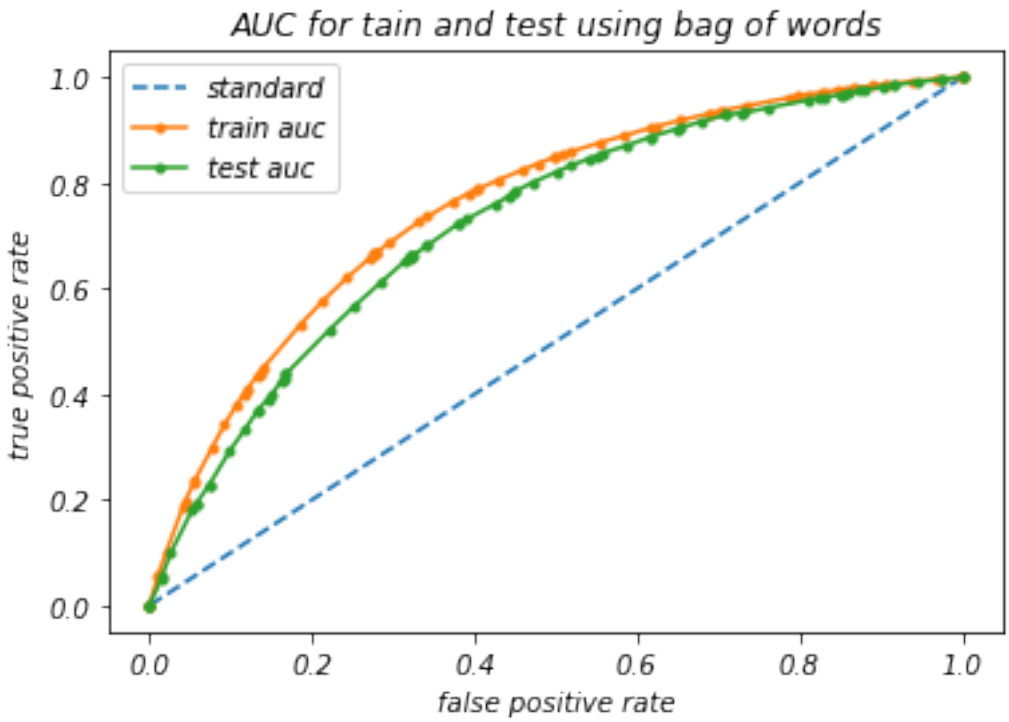
```
Out[58]: SGDClassifier(alpha=0.01, average=False, class_weight='balanced',
                       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                       n_iter=None, n_iter_no_change=5, n_jobs=2, penalty='l2',
                       power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
                       validation_fraction=0.1, verbose=0, warm_start=False)
```

```
In [59]: from sklearn.calibration import CalibratedClassifierCV
         model3 = CalibratedClassifierCV(model2, method='isotonic', cv='prefit')
         model3.fit(BOW,project_data_Y_train)
```

Out[59]:
```
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.01, average=False, class_w
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=2, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
       validation_fraction=0.1, verbose=0, warm_start=False),
            cv='prefit', method='isotonic')
```

In [60]:
```
#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 = model3.predict_proba(BOW_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 = model3.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()
```

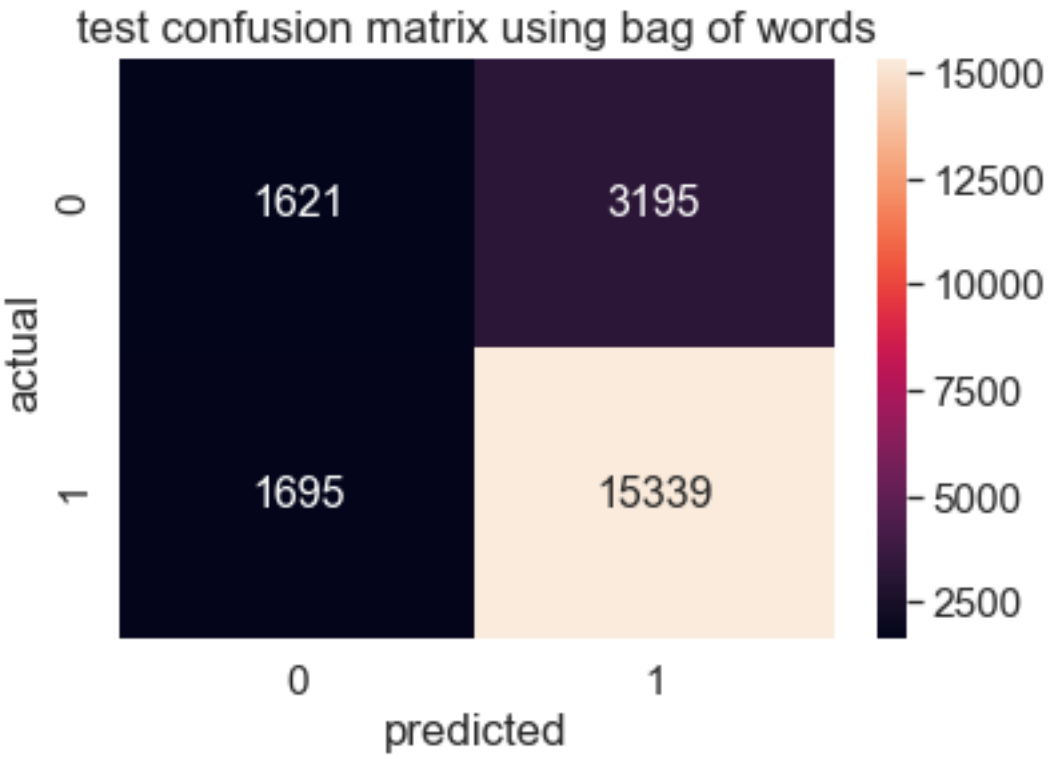
```
AUC: 0.724
AUC: 0.757
```



In [61]:
```
#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=model2.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("predicted")
plt.ylabel("actual")
plt.show()
```

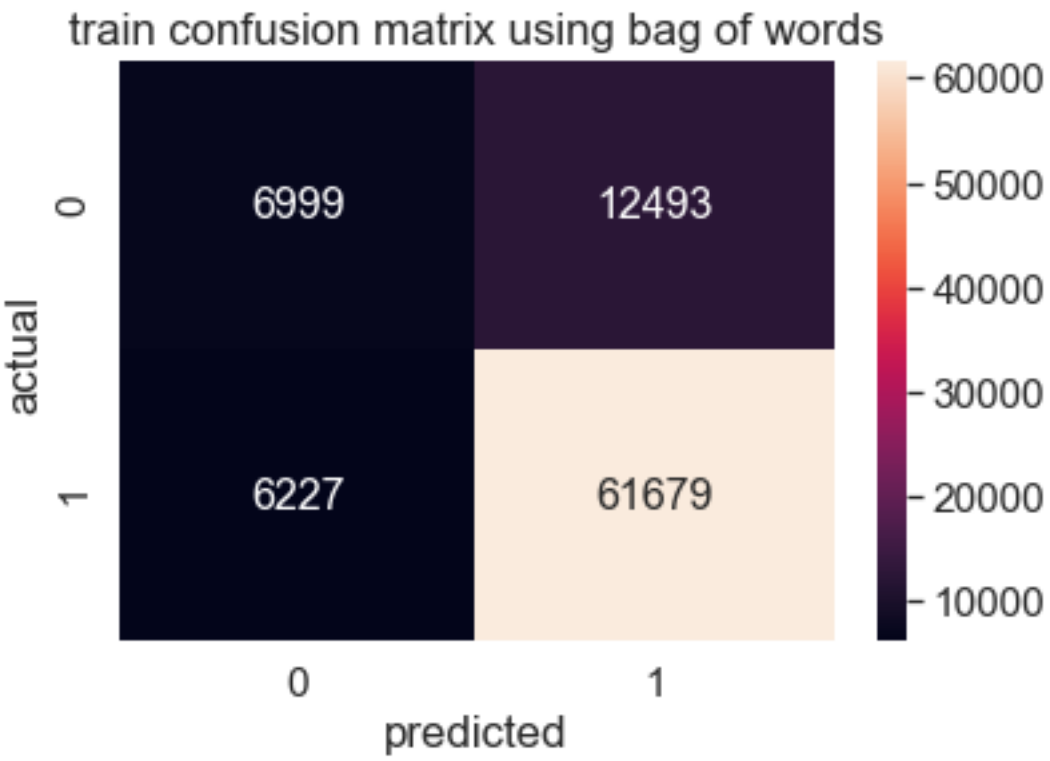
```
1621 1695 3195 15339
true positive rate 0.8276141146001942
true negaitive rate 0.48884197828709286
```

```
[[1621, 3195], [1695, 15339]]
```



In [62]:
```
#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=model2.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("predicted")
plt.ylabel("actual")
plt.show()
```

```
6999 6227 12493 61679
true positive rate 0.8315671682036347
true negaitive rate 0.5291849387569938
[[6999, 12493], [6227, 61679]]
```



2.0.2 2.4.1 Applying LR on TFIDF, SET 2

```
In [63]: # 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)
```

```
(87398, 7900)
(21850, 7900)
```

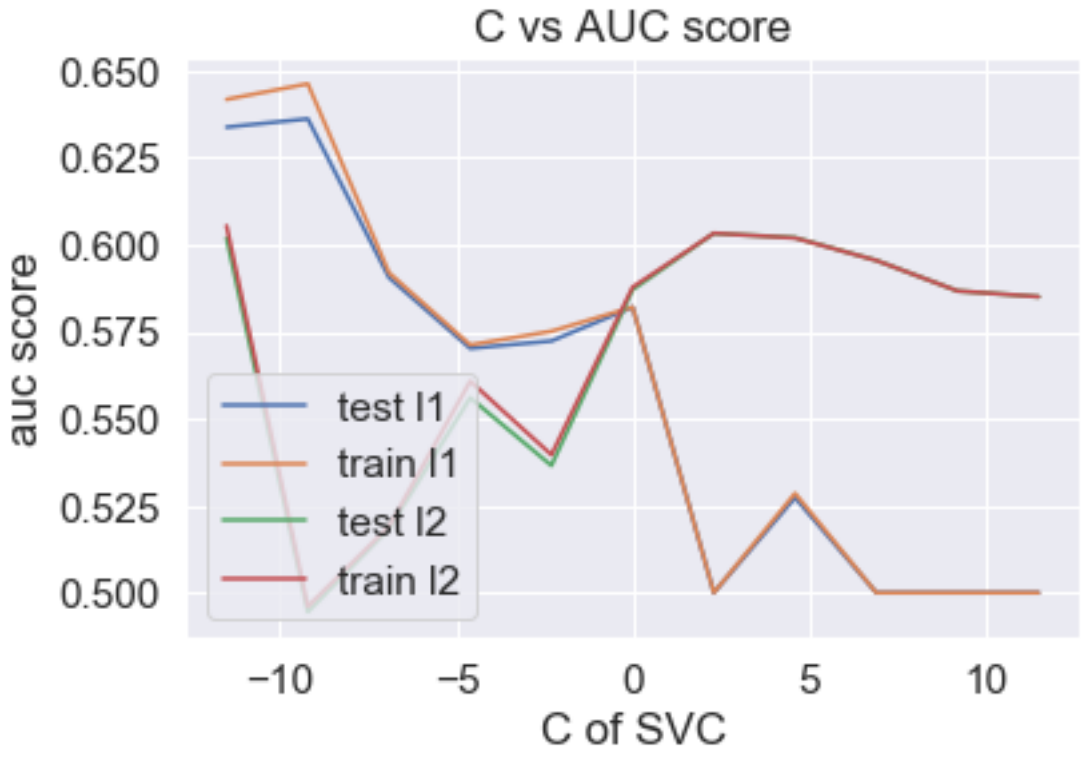
```
In [64]: from sklearn.linear_model import SGDClassifier
         from sklearn.model_selection import GridSearchCV
         model4=SGDClassifier(class_weight='balanced',tol=10**-7)
```

```
a=[10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]
print(a)
b=['l1','l2']
parameters = {'alpha': a, 'penalty':b}
clf = GridSearchCV(model4, parameters,scoring='roc_auc',n_jobs=2,verbose=10)
clf.fit(TFIDF,project_data_Y_train)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
Fitting 3 folds for each of 22 candidates, totalling 66 fits


[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   1 tasks      | elapsed:    4.8s
[Parallel(n_jobs=2)]: Done   4 tasks      | elapsed:   10.2s
[Parallel(n_jobs=2)]: Done   9 tasks      | elapsed:   23.1s
[Parallel(n_jobs=2)]: Done  14 tasks      | elapsed:   29.3s
[Parallel(n_jobs=2)]: Done  21 tasks      | elapsed:   38.3s
[Parallel(n_jobs=2)]: Done  28 tasks      | elapsed:   45.7s
[Parallel(n_jobs=2)]: Done  37 tasks      | elapsed:   56.0s
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:   59.9s
[Parallel(n_jobs=2)]: Done  57 tasks      | elapsed:  1.1min
[Parallel(n_jobs=2)]: Done  66 out of  66 | elapsed:  1.8min finished


Out[64]: GridSearchCV(cv='warn', error_score='raise-deprecating',
               estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
               early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
               l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
               n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
               power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
               validation_fraction=0.1, verbose=0, warm_start=False),
               fit_params=None, iid='warn', n_jobs=2,
               param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000
               pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
               scoring='roc_auc', verbose=10)
```

```
In [65]: k=a
         auc_cv=clf.cv_results_['mean_test_score']
         auc_train=clf.cv_results_['mean_train_score']
         plt.plot(np.log(k),auc_cv[::2])
         plt.plot(np.log(k),auc_train[::2])
         plt.plot(np.log(k),auc_cv[1::2])
         plt.plot(np.log(k),auc_train[1::2])
         plt.title('C vs AUC score')
         plt.xlabel('C of SVC')
         plt.ylabel('auc score')
         plt.legend({"test l1":"","train l1":"","test l2":"","train l2":""})
```

```
Out[65]: <matplotlib.legend.Legend at 0x289a15d3a90>
```




```
In [66]: print(k)
         print(np.log(k))
         np.exp(-9.21034037)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
[-11.51292546  -9.21034037  -6.90775528  -4.60517019  -2.30258509
   0.           2.30258509   4.60517019   6.90775528   9.21034037
  11.51292546]
```

```
Out[66]: 0.00010000000019761819
```

```
In [67]: model5=SGDClassifier(alpha=0.0001,penalty='l1',n_jobs=2,loss='hinge',class_weight='ba
         model5.fit(TFIDF,project_data_Y_train)
```

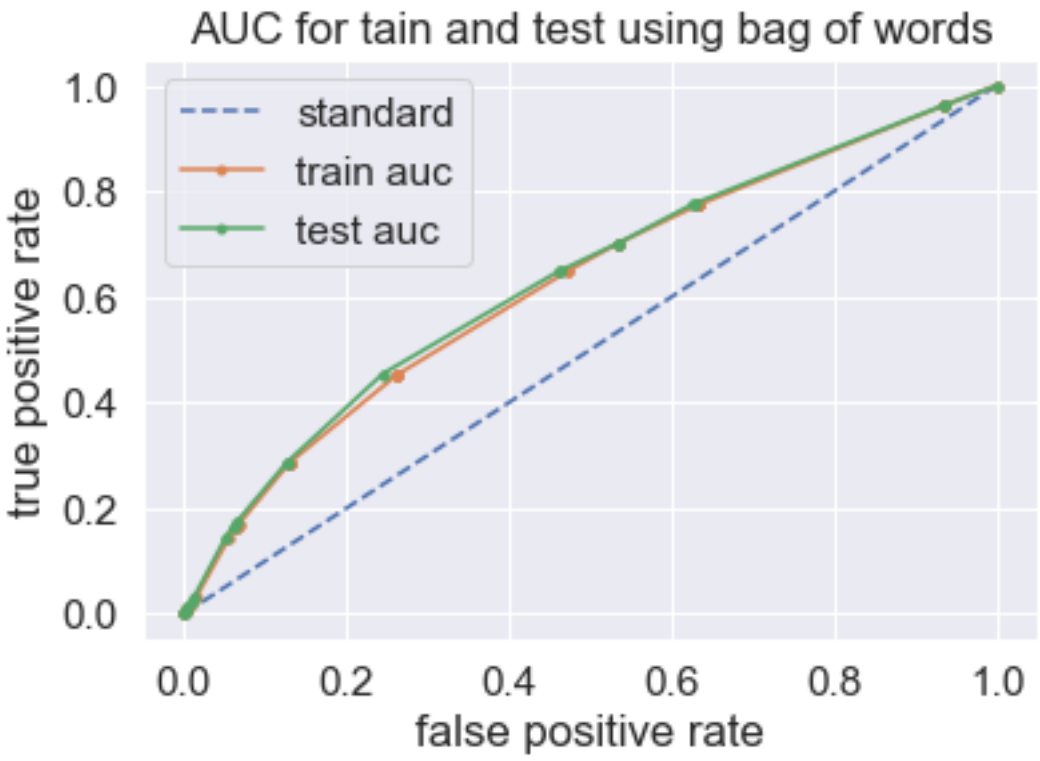
```
Out[67]: SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
                early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                n_iter=None, n_iter_no_change=5, n_jobs=2, penalty='l1',
                power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
                validation_fraction=0.1, verbose=0, warm_start=False)
```

```
In [68]: from sklearn.calibration import CalibratedClassifierCV
         model6 = CalibratedClassifierCV(model5, method='isotonic', cv='prefit')
         model6.fit(BOW,project_data_Y_train)
```

```
Out[68]: CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.0001, average=False, class
                early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                n_iter=None, n_iter_no_change=5, n_jobs=2, penalty='l1',
                power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
                validation_fraction=0.1, verbose=0, warm_start=False),
                     cv='prefit', method='isotonic')
```

```
In [69]: #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 = model6.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 = model6.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 bag of words")
         plt.xlabel("false positive rate")
         plt.ylabel("true positive rate")
         plt.show()
```

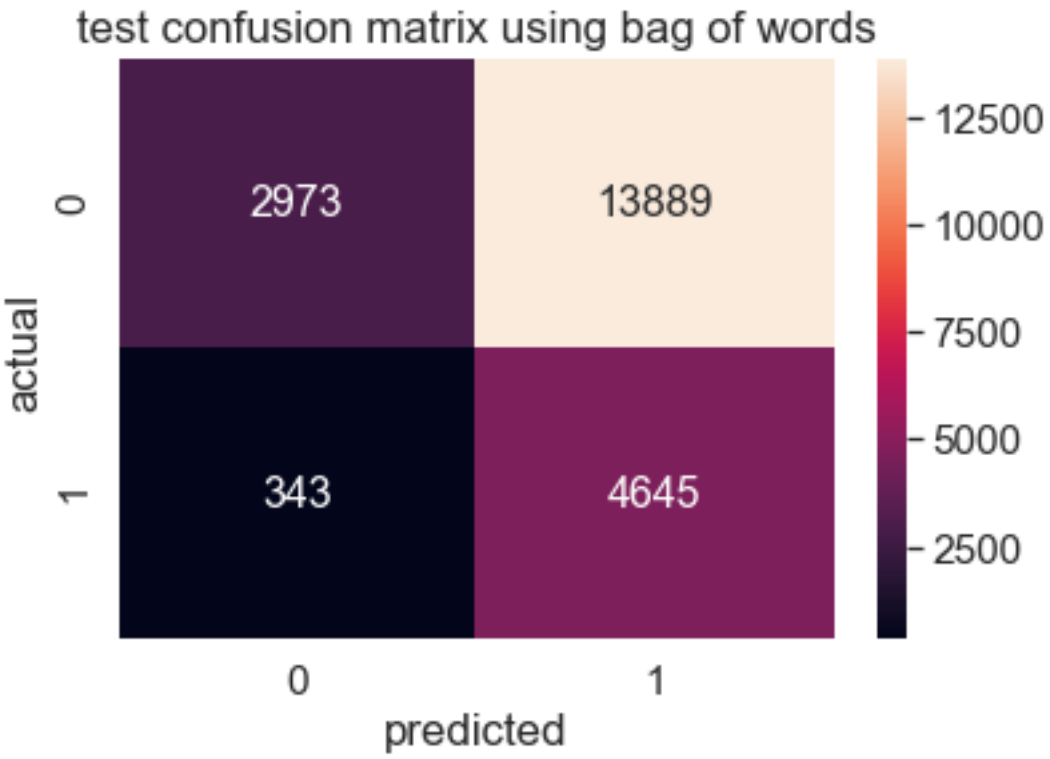
```
AUC: 0.633
AUC: 0.625
```



In [70]:
```
#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=model5.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("test confusion matrix using bag of words")
plt.xlabel("predicted")
plt.ylabel("actual")
plt.show()
```

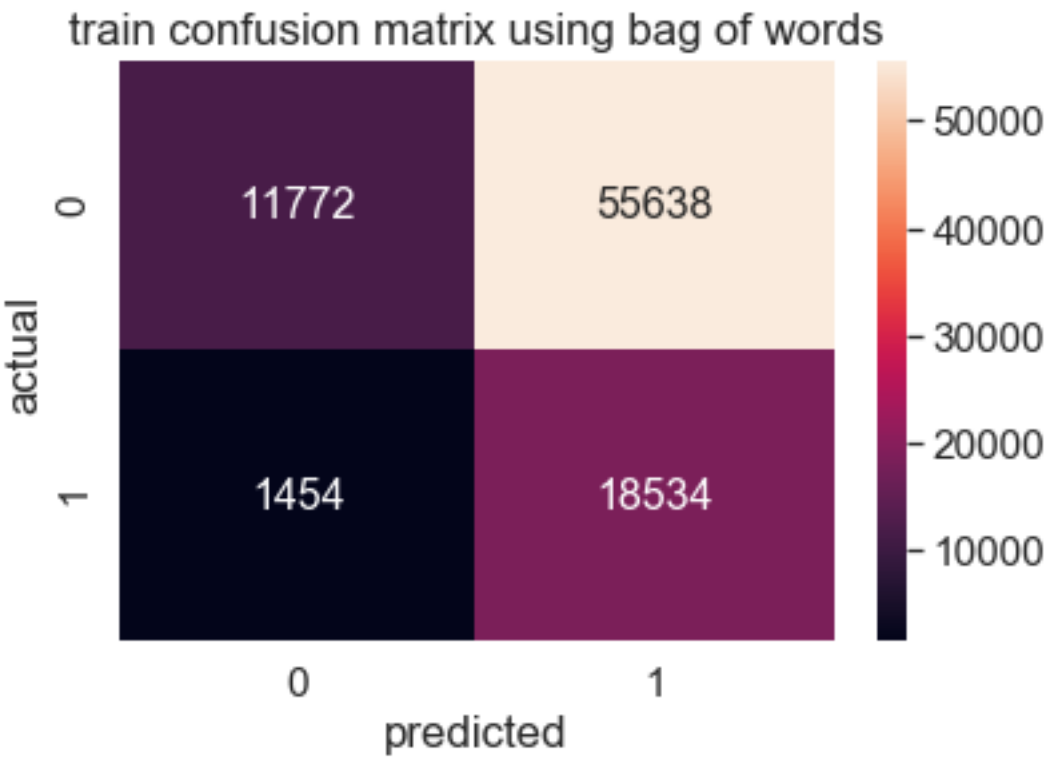
```
2973 343 13889 4645
true positive rate 0.2506204812776519
true negaitive rate 0.896562123039807
```

```
[[2973, 13889], [343, 4645]]
```



In [71]:
```
#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=model5.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("train confusion matrix using bag of words")
plt.xlabel("predicted")
plt.ylabel("actual")
plt.show()
```

```
11772 1454 55638 18534
true positive rate 0.24987866041093673
true negaitive rate 0.8900650234386814
[[11772, 55638], [1454, 18534]]
```



2.0.3 2.4.1 Applying LR on average word to vector, SET 3

```
In [72]: # 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)
```

```
(87398, 697)
(21850, 697)
```

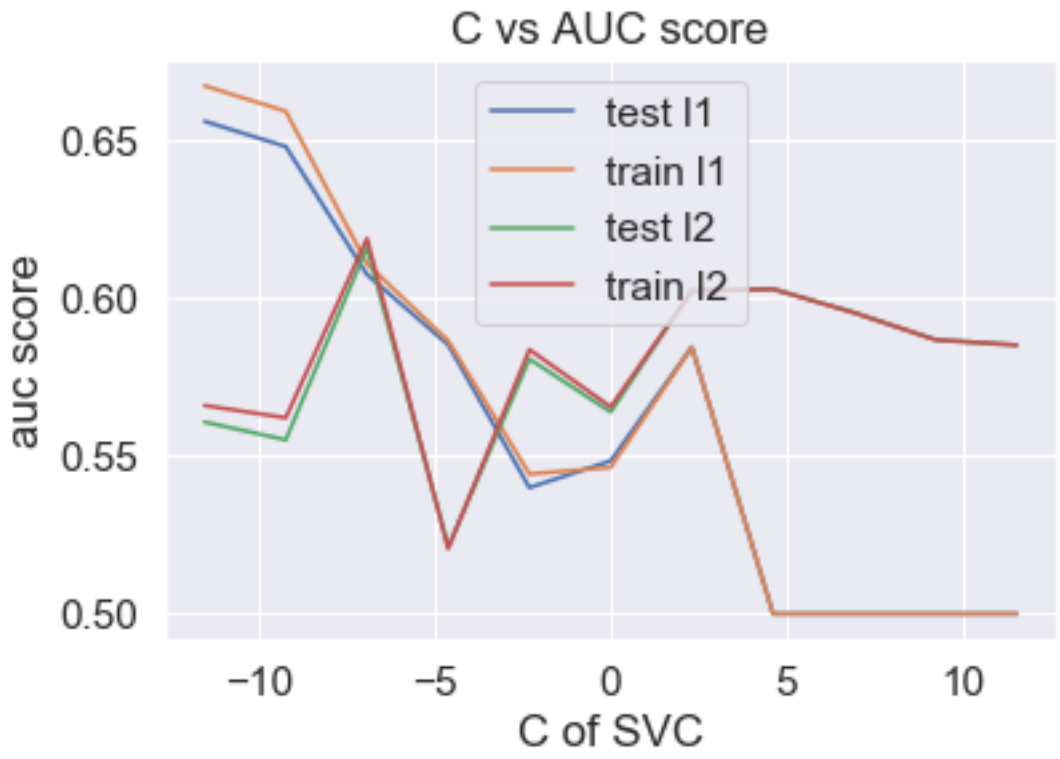
```
In [73]: from sklearn.linear_model import SGDClassifier
         from sklearn.model_selection import GridSearchCV
         model7=SGDClassifier(class_weight='balanced',tol=10**-7)
```

```
        a=[10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]
        print(a)
        b=['l1','l2']
        parameters = {'alpha': a, 'penalty':b}
        clf = GridSearchCV(model7, parameters,scoring='roc_auc',n_jobs=2,verbose=10)
        clf.fit(AVG_W2V,project_data_Y_train)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
Fitting 3 folds for each of 22 candidates, totalling 66 fits


[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   1 tasks      | elapsed:    8.2s
[Parallel(n_jobs=2)]: Done   4 tasks      | elapsed:   24.5s
[Parallel(n_jobs=2)]: Done   9 tasks      | elapsed:   47.2s
[Parallel(n_jobs=2)]: Done  14 tasks      | elapsed:  1.2min
[Parallel(n_jobs=2)]: Done  21 tasks      | elapsed:  1.8min
[Parallel(n_jobs=2)]: Done  28 tasks      | elapsed:  2.3min
[Parallel(n_jobs=2)]: Done  37 tasks      | elapsed:  2.8min
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:  3.2min
[Parallel(n_jobs=2)]: Done  57 tasks      | elapsed:  3.6min
[Parallel(n_jobs=2)]: Done  66 out of  66 | elapsed:  7.6min finished


Out[73]: GridSearchCV(cv='warn', error_score='raise-deprecating',
               estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
               early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
               l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
               n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
               power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
               validation_fraction=0.1, verbose=0, warm_start=False),
               fit_params=None, iid='warn', n_jobs=2,
               param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000
               pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
               scoring='roc_auc', verbose=10)
```

```
In [74]: k=a
         auc_cv=clf.cv_results_['mean_test_score']
         auc_train=clf.cv_results_['mean_train_score']
         plt.plot(np.log(k),auc_cv[::2])
         plt.plot(np.log(k),auc_train[::2])
         plt.plot(np.log(k),auc_cv[1::2])
         plt.plot(np.log(k),auc_train[1::2])
         plt.title('C vs AUC score')
         plt.xlabel('C of SVC')
         plt.ylabel('auc score')
         plt.legend({"test l1":"","train l1":"","test l2":"","train l2":""})
```

```
Out[74]: <matplotlib.legend.Legend at 0x289bcc9d940>
```




```
In [75]: print(k)
         print(np.log(k))
         np.exp(-9.21034037)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
[-11.51292546  -9.21034037  -6.90775528  -4.60517019  -2.30258509
   0.           2.30258509   4.60517019   6.90775528   9.21034037
  11.51292546]
```

```
Out[75]: 0.00010000000019761819
```

```
In [76]: model8=SGDClassifier(alpha=0.0001,penalty='l1',n_jobs=2,loss='hinge',class_weight='ba
         model8.fit(AVG_W2V,project_data_Y_train)
```

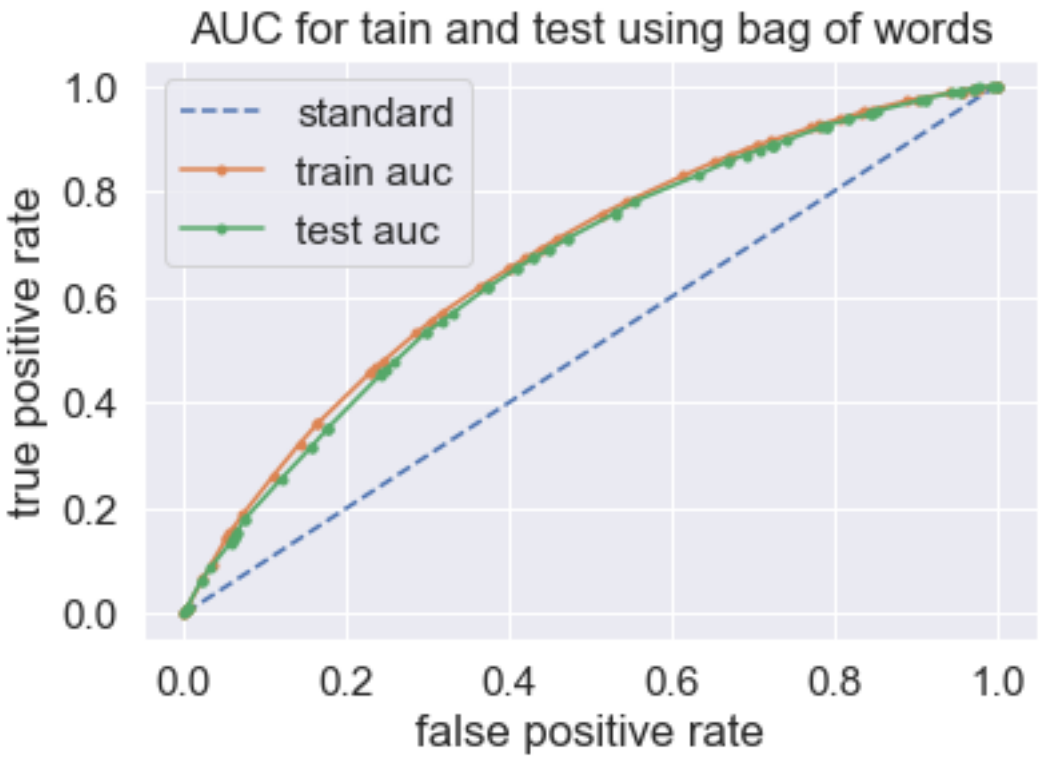
```
Out[76]: SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
                early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                n_iter=None, n_iter_no_change=5, n_jobs=2, penalty='l1',
                power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
                validation_fraction=0.1, verbose=0, warm_start=False)
```

```
In [77]: from sklearn.calibration import CalibratedClassifierCV
         model9 = CalibratedClassifierCV(model8, method='isotonic', cv='prefit')
         model9.fit(AVG_W2V,project_data_Y_train)
```

```
Out[77]: CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.0001, average=False, class
                early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                n_iter=None, n_iter_no_change=5, n_jobs=2, penalty='l1',
                power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
                validation_fraction=0.1, verbose=0, warm_start=False),
                     cv='prefit', method='isotonic')
```

```
In [78]: #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 = model9.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 = model9.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 bag of words")
         plt.xlabel("false positive rate")
         plt.ylabel("true positive rate")
         plt.show()
```

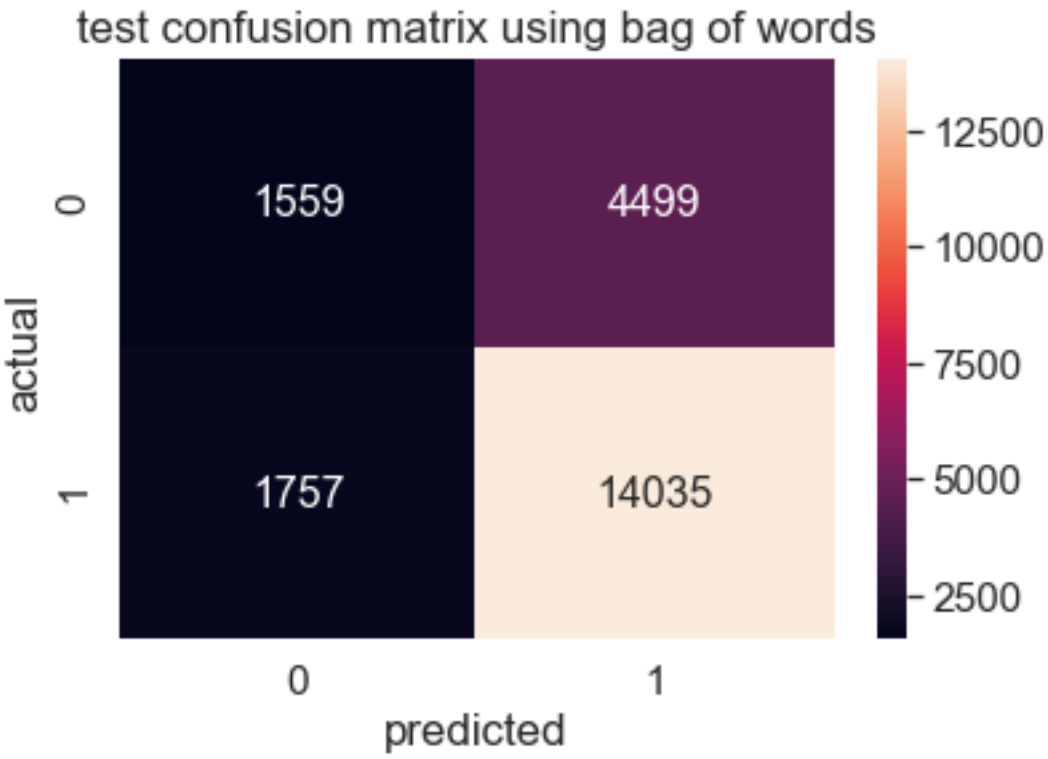
```
AUC: 0.663
AUC: 0.674
```



In [79]:
```
#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=model8.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("test confusion matrix using bag of words")
plt.xlabel("predicted")
plt.ylabel("actual")
plt.show()
```

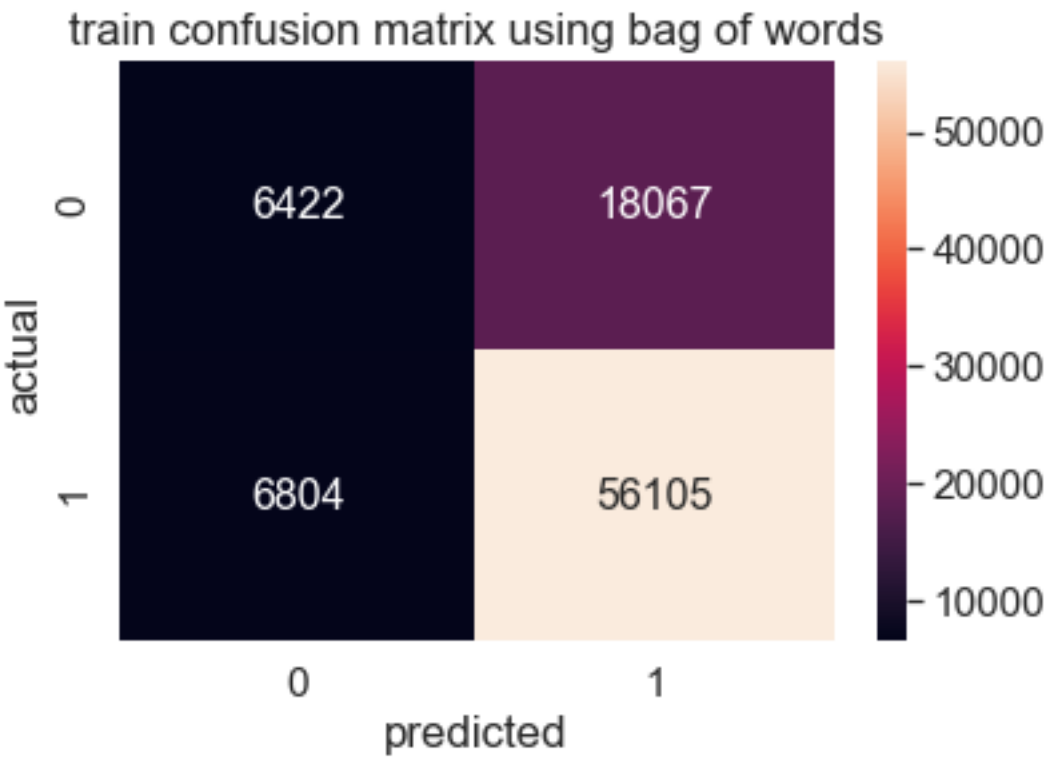
```
1559 1757 4499 14035
true positive rate 0.7572569332038416
true negaitive rate 0.4701447527141134
```

```
[[1559, 4499], [1757, 14035]]
```



In [80]:
```
#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=model8.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("train confusion matrix using bag of words")
plt.xlabel("predicted")
plt.ylabel("actual")
plt.show()
```

```
6422 6804 18067 56105
true positive rate 0.7564175160437902
true negaitive rate 0.48555874792076215
[[6422, 18067], [6804, 56105]]
```



2.0.4 2.4.1 Applying LR on tfidf word to vector, SET 4

```
In [81]: # 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_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)
```

```
(87398, 697)
(21850, 697)
```

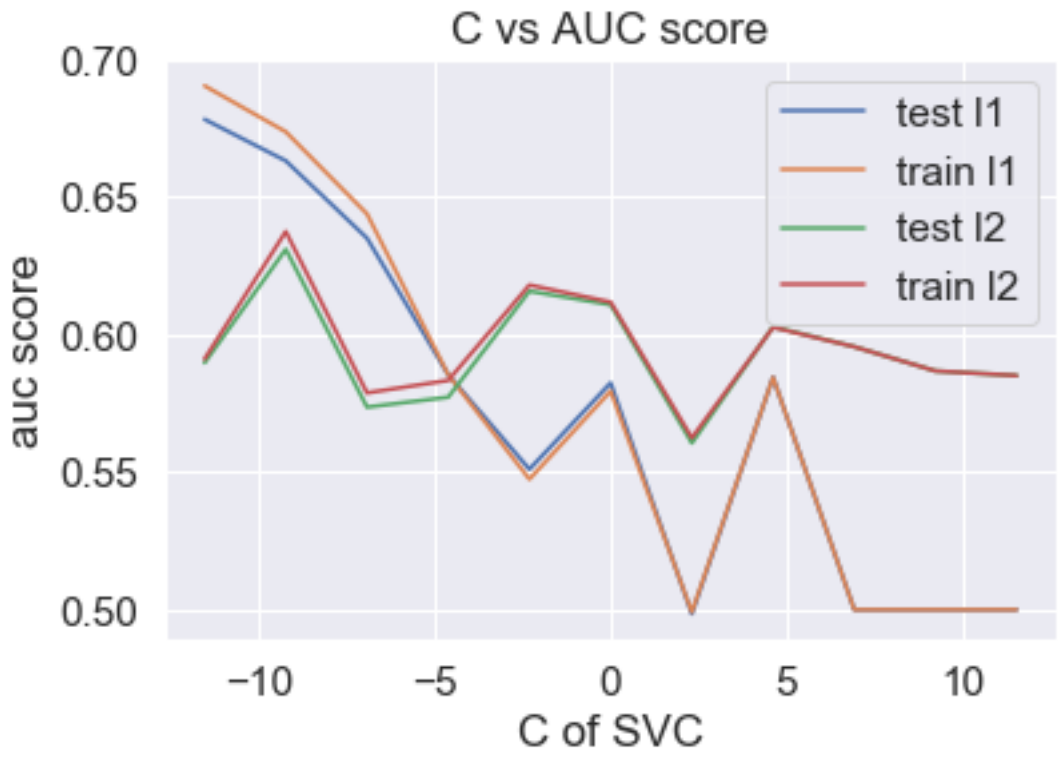
```
In [82]: from sklearn.linear_model import SGDClassifier
         from sklearn.model_selection import GridSearchCV
         model10=SGDClassifier(class_weight='balanced',tol=10**-7)
```

```
        a=[10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]
        print(a)
        b=['l1','l2']
        parameters = {'alpha': a, 'penalty':b}
        clf = GridSearchCV(model10, parameters,scoring='roc_auc',n_jobs=2,verbose=10)
        clf.fit(TFIDF_W2V,project_data_Y_train)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
Fitting 3 folds for each of 22 candidates, totalling 66 fits


[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   1 tasks      | elapsed:    9.3s
[Parallel(n_jobs=2)]: Done   4 tasks      | elapsed:   24.0s
[Parallel(n_jobs=2)]: Done   9 tasks      | elapsed:   46.5s
[Parallel(n_jobs=2)]: Done  14 tasks      | elapsed:  1.2min
[Parallel(n_jobs=2)]: Done  21 tasks      | elapsed:  1.7min
[Parallel(n_jobs=2)]: Done  28 tasks      | elapsed:  2.1min
[Parallel(n_jobs=2)]: Done  37 tasks      | elapsed:  3.6min
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed: 14.0min
[Parallel(n_jobs=2)]: Done  57 tasks      | elapsed: 15.0min
[Parallel(n_jobs=2)]: Done  66 out of  66 | elapsed: 21.6min finished


Out[82]: GridSearchCV(cv='warn', error_score='raise-deprecating',
               estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
               early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
               l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
               n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
               power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
               validation_fraction=0.1, verbose=0, warm_start=False),
               fit_params=None, iid='warn', n_jobs=2,
               param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000
               pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
               scoring='roc_auc', verbose=10)
```

```
In [83]: k=a
         auc_cv=clf.cv_results_['mean_test_score']
         auc_train=clf.cv_results_['mean_train_score']
         plt.plot(np.log(k),auc_cv[::2])
         plt.plot(np.log(k),auc_train[::2])
         plt.plot(np.log(k),auc_cv[1::2])
         plt.plot(np.log(k),auc_train[1::2])
         plt.title('C vs AUC score')
         plt.xlabel('C of SVC')
         plt.ylabel('auc score')
         plt.legend({"test l1":"","train l1":"","test l2":"","train l2":""})
```

```
Out[83]: <matplotlib.legend.Legend at 0x28983d26b00>
```




```
In [84]: print(k)
         print(np.log(k))
         np.exp(-9.21034037)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
[-11.51292546  -9.21034037  -6.90775528  -4.60517019  -2.30258509
   0.           2.30258509   4.60517019   6.90775528   9.21034037
  11.51292546]
```

```
Out[84]: 0.00010000000019761819
```

```
In [85]: model11=SGDClassifier(alpha=0.0001,penalty='l2',n_jobs=2,loss='hinge',class_weight='ba
         model11.fit(TFIDF_W2V,project_data_Y_train)
```

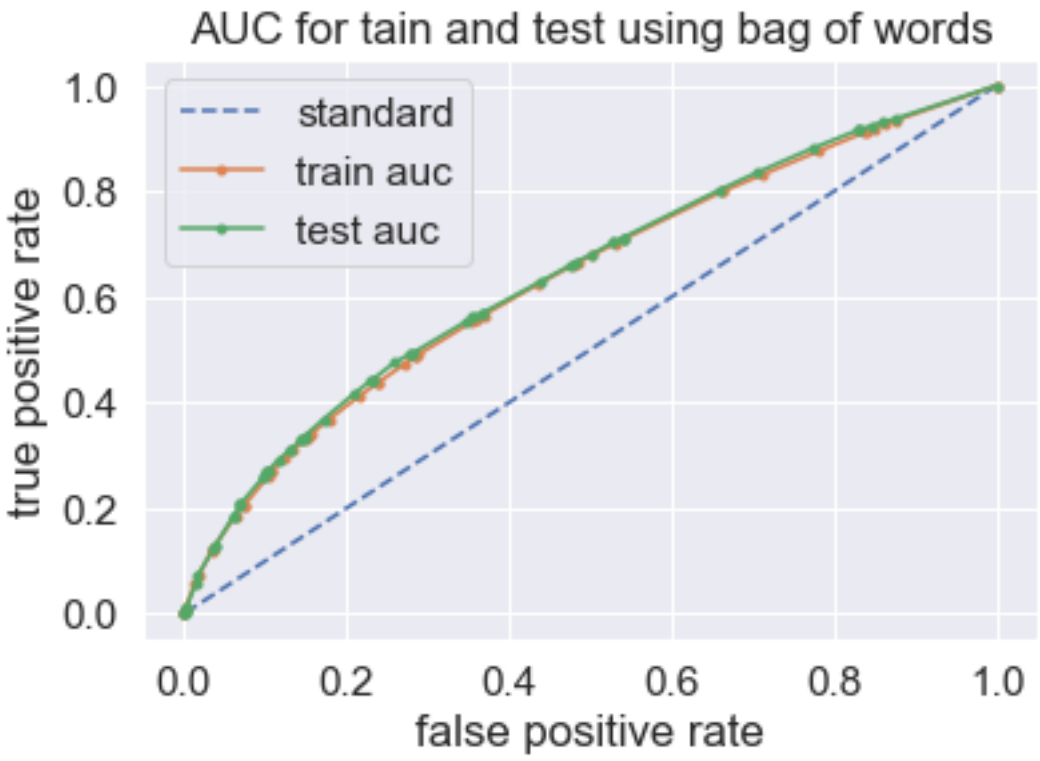
```
Out[85]: SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
                       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                       n_iter=None, n_iter_no_change=5, n_jobs=2, penalty='l2',
                       power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
                       validation_fraction=0.1, verbose=0, warm_start=False)
```

```
In [86]: from sklearn.calibration import CalibratedClassifierCV
         model12 = CalibratedClassifierCV(model11,method='isotonic', cv='prefit')
         model12.fit(TFIDF_W2V,project_data_Y_train)
```

Out[86]:
```
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.0001, average=False, class
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=2, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
       validation_fraction=0.1, verbose=0, warm_start=False),
            cv='prefit', method='isotonic')
```

In [87]:
```
#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 = model12.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 = model12.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 bag of words")
plt.xlabel("false positive rate")
plt.ylabel("true positive rate")
plt.show()
```

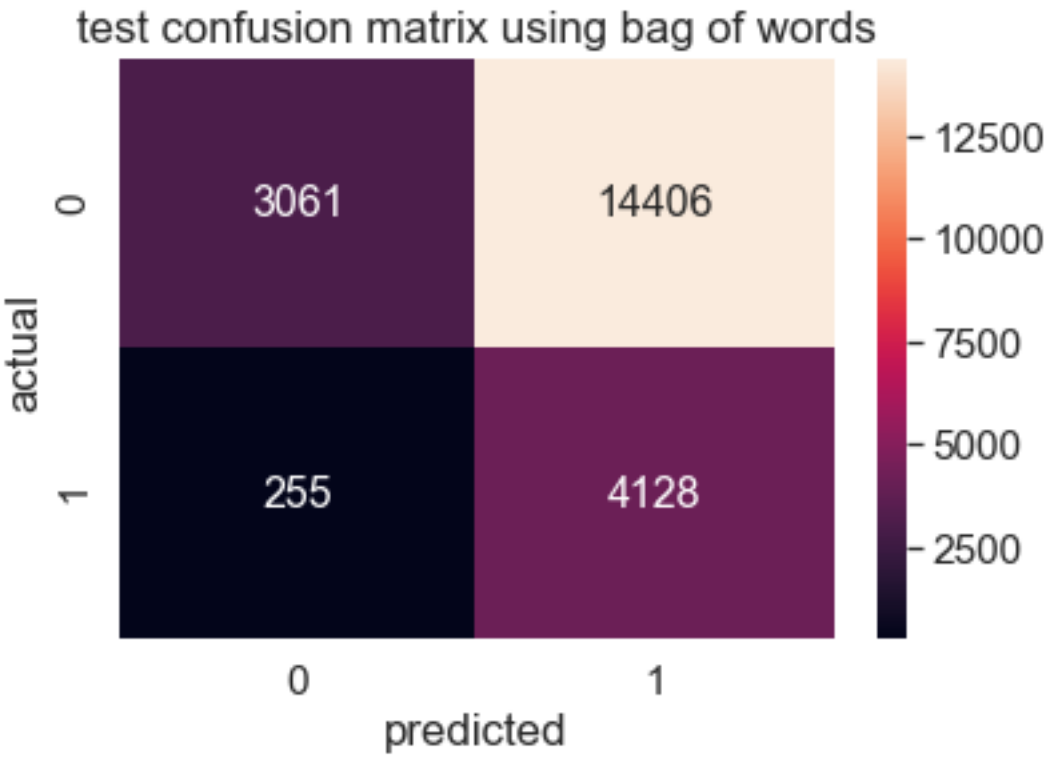
```
AUC: 0.642
AUC: 0.636
```



In [88]:
```
#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=model11.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("test confusion matrix using bag of words")
plt.xlabel("predicted")
plt.ylabel("actual")
plt.show()
```

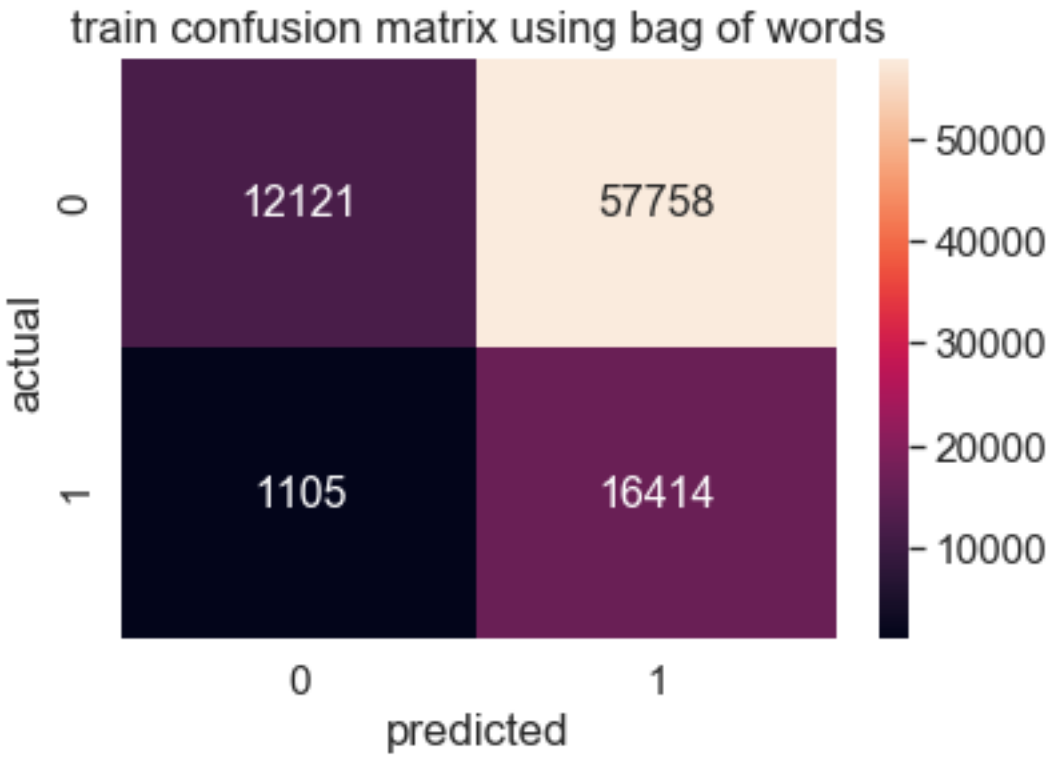
```
3061 255 14406 4128
true positive rate 0.22272580123017158
true negaitive rate 0.9231001206272618
```

```
[[3061, 14406], [255, 4128]]
```



In [89]:
```
#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=model11.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("train confusion matrix using bag of words")
plt.xlabel("predicted")
plt.ylabel("actual")
plt.show()
```

```
12121 1105 57758 16414
true positive rate 0.2212964460982581
true negaitive rate 0.916452442159383
[[12121, 57758], [1105, 16414]]
```



2.5 Support Vector Machines with added Features Set 5

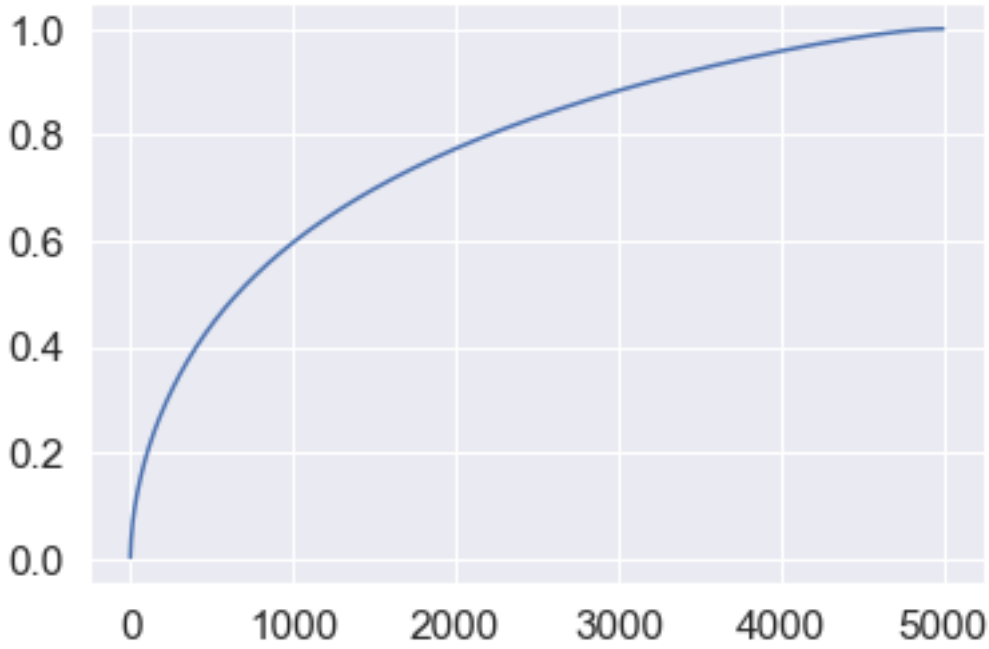
```
In [90]: from sklearn.decomposition import TruncatedSVD
         model12=TruncatedSVD(n_components=text_tfidf_train.shape[1]-1)
         model12.fit(text_tfidf_train)
         print(np.cumsum(model12.explained_variance_ratio_))
```

```
[0.00199884 0.01157373 0.01987435 ... 0.99999917 0.99999957 0.99999996]
```

```
In [91]: y_axis=np.cumsum(model12.explained_variance_ratio_)
         print(len(y_axis))
         x_axis=np.arange(1,5000)
         plt.plot(x_axis,y_axis)
```

```
4999
```

```
Out[91]: [<matplotlib.lines.Line2D at 0x289a60466d8>]
```



By taking 3500 points i can preserve 90% of variance

```
In [92]: print(text_tfidf_train.shape)
         print(text_tfidf_test.shape)
         model13=TruncatedSVD(n_components=3500)
         text_tfidf_train_TRSVD=model13.fit_transform(text_tfidf_train)
         text_tfidf_test_TRSVD=model13.transform(text_tfidf_test)
         print(text_tfidf_train_TRSVD.shape)
         print(text_tfidf_test_TRSVD.shape)
```

```
(87398, 5000)
(21850, 5000)
(87398, 3500)
(21850, 3500)
```

```
In [93]: from nltk.tokenize import word_tokenize
         nltk.download('punkt')
         len_word_essay_train=[]
         len_word_essay_test=[]
         len_word_title_train=[]
         len_word_title_test=[]
         for i in tqdm(project_data_X_train.essay.values):
             len_word_essay_train.append(len(word_tokenize(i)))
         for i in tqdm(project_data_X_test.essay.values):
             len_word_essay_test.append(len(word_tokenize(i)))
```

```
        for i in tqdm(project_data_X_train.project_title.values):
            len_word_title_train.append(len(word_tokenize(i)))
        for i in tqdm(project_data_X_test.project_title.values):
            len_word_title_test.append(len(word_tokenize(i)))
        print(len(len_word_essay_train))
        print(len(len_word_essay_test))
        print(len(len_word_title_train))
        print(len(len_word_title_test))
```

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

```
100%|| 87398/87398 [00:52<00:00, 1652.75it/s]
100%|| 21850/21850 [00:13<00:00, 1653.97it/s]
100%|| 87398/87398 [00:05<00:00, 15953.29it/s]
100%|| 21850/21850 [00:01<00:00, 15979.61it/s]
```

```
87398
21850
87398
21850
```

In [94]:
```
        # 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
        project_data_X_train['wcEssay']=len_word_essay_train
        project_data_X_train['wcTitle']=len_word_title_train
        project_data_X_test['wcEssay']=len_word_essay_test
        project_data_X_test['wcTitle']=len_word_title_test
        #text_tfidf_test_TRSVD,project_data_X_test['wcEssay'],project_data_X_test['wcTitle'])
        #text_tfidf_train_TRSVD,project_data_X_train['wcEssay'],project_data_X_train['wcTitle
        special_model_train = hstack((categories_one_hot_train, sub_categories_one_hot_train,s
        print(special_model_train.shape)
        special_model_test= hstack((categories_one_hot_test, sub_categories_one_hot_test,schoo
        print(special_model_test.shape)
```

```
(87398, 3603)
(21850, 3603)
```

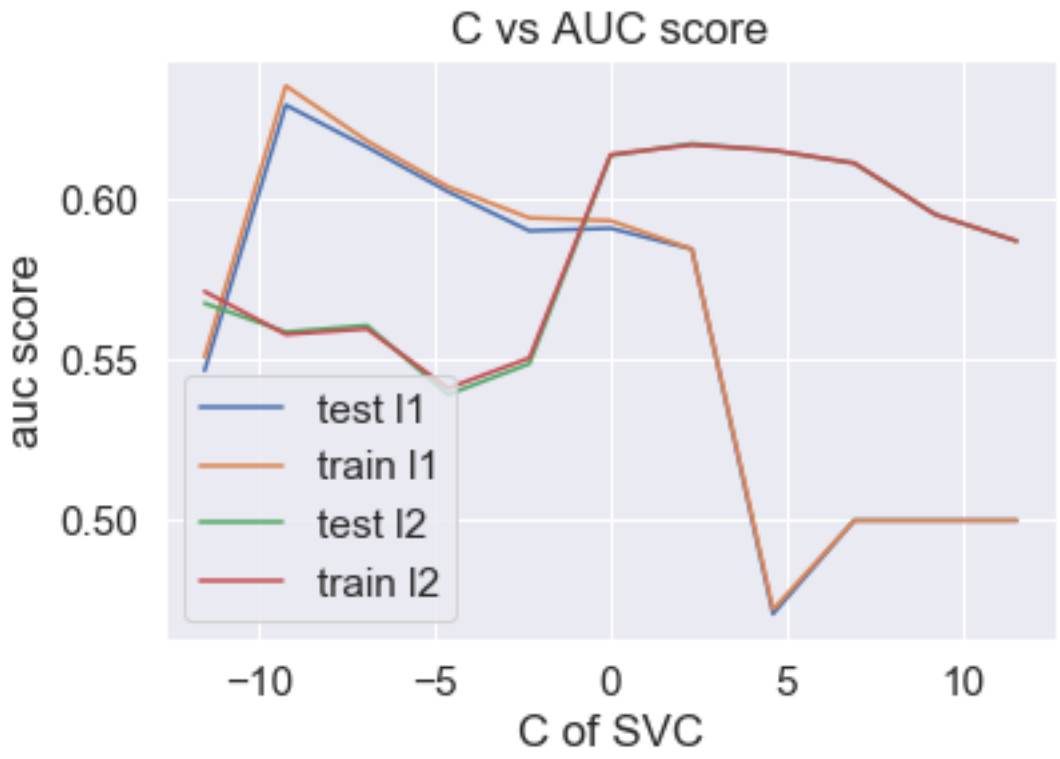
In [95]:
```
        from sklearn.linear_model import SGDClassifier
        from sklearn.model_selection import GridSearchCV
        model13=SGDClassifier(class_weight='balanced',tol=10**-7)
        a=[10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]
        print(a)
```

```
        b=['l1','l2']
        parameters = {'alpha': a, 'penalty':b}
        clf = GridSearchCV(model13, parameters,scoring='roc_auc',n_jobs=2,verbose=10)
        clf.fit(special_model_train,project_data_Y_train)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
Fitting 3 folds for each of 22 candidates, totalling 66 fits


[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   1 tasks      | elapsed:  5.4min
[Parallel(n_jobs=2)]: Done   4 tasks      | elapsed:  7.4min
[Parallel(n_jobs=2)]: Done   9 tasks      | elapsed:  9.5min
[Parallel(n_jobs=2)]: Done  14 tasks      | elapsed: 11.7min
[Parallel(n_jobs=2)]: Done  21 tasks      | elapsed: 14.2min
[Parallel(n_jobs=2)]: Done  28 tasks      | elapsed: 17.5min
[Parallel(n_jobs=2)]: Done  37 tasks      | elapsed: 21.0min
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed: 22.6min
[Parallel(n_jobs=2)]: Done  57 tasks      | elapsed: 24.9min
[Parallel(n_jobs=2)]: Done  66 out of  66 | elapsed: 33.6min finished


Out[95]: GridSearchCV(cv='warn', error_score='raise-deprecating',
                estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
                early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
                power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
                validation_fraction=0.1, verbose=0, warm_start=False),
                fit_params=None, iid='warn', n_jobs=2,
                param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='roc_auc', verbose=10)
```

```
In [97]: k=a
         auc_cv=clf.cv_results_['mean_test_score']
         auc_train=clf.cv_results_['mean_train_score']
         plt.plot(np.log(k),auc_cv[::2])
         plt.plot(np.log(k),auc_train[::2])
         plt.plot(np.log(k),auc_cv[1::2])
         plt.plot(np.log(k),auc_train[1::2])
         plt.title('C vs AUC score')
         plt.xlabel('C of SVC')
         plt.ylabel('auc score')
         plt.legend({"test l1":"","train l1":"","test l2":"","train l2":""})
```

```
Out[97]: <matplotlib.legend.Legend at 0x289bfa31240>
```




```
In [98]: print(k)
         print(np.log(k))
         np.exp(2.5)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
[-11.51292546  -9.21034037  -6.90775528  -4.60517019  -2.30258509
   0.           2.30258509   4.60517019   6.90775528   9.21034037
  11.51292546]
```

```
Out[98]: 12.182493960703473
```

```
In [99]: model14=SGDClassifier(alpha=2.5,penalty='l2',n_jobs=2,loss='hinge',class_weight='bala
         model14.fit(special_model_train,project_data_Y_train)
```

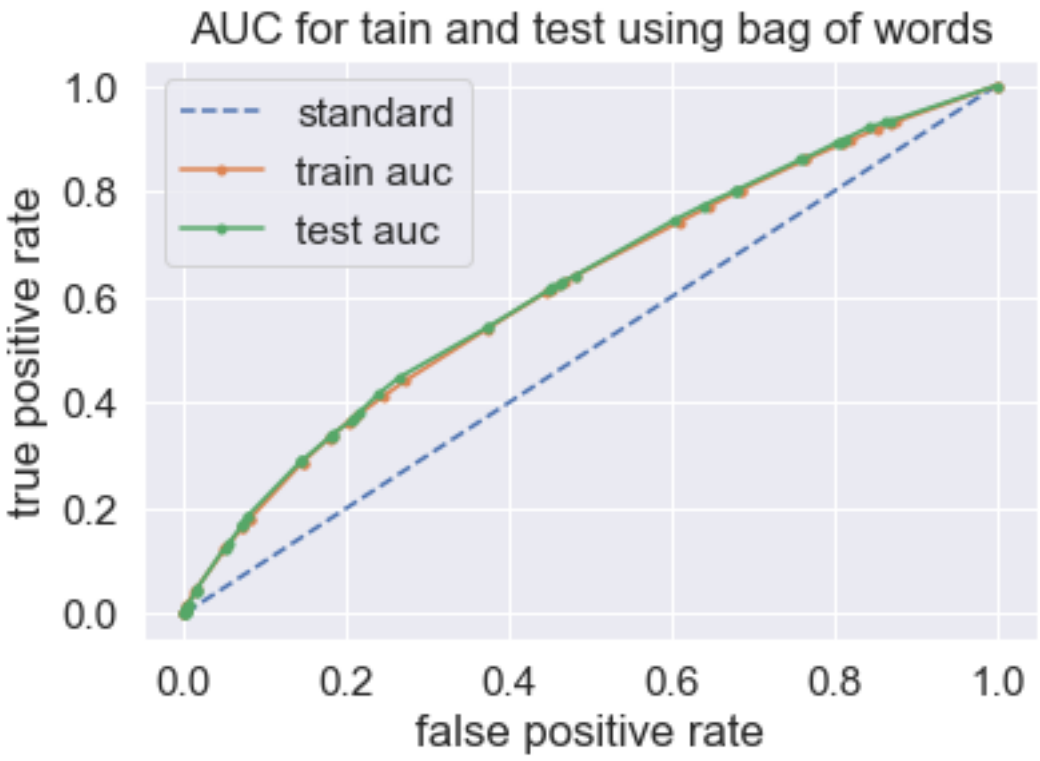
```
Out[99]: SGDClassifier(alpha=2.5, average=False, class_weight='balanced',
                       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                       n_iter=None, n_iter_no_change=5, n_jobs=2, penalty='l2',
                       power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
                       validation_fraction=0.1, verbose=0, warm_start=False)
```

```
In [100]: from sklearn.calibration import CalibratedClassifierCV
          model15 = CalibratedClassifierCV(model14,method='isotonic', cv='prefit')
          model15.fit(special_model_train,project_data_Y_train)
```

```
Out[100]: CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=2.5, average=False, class_w
                early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                n_iter=None, n_iter_no_change=5, n_jobs=2, penalty='l2',
                power_t=0.5, random_state=None, shuffle=True, tol=1e-07,
                validation_fraction=0.1, verbose=0, warm_start=False),
                     cv='prefit', method='isotonic')
```

```
In [101]: #https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-class
          from sklearn.metrics import roc_curve
          from sklearn.metrics import roc_auc_score
          from tqdm import tqdm

          probs_test = model15.predict_proba(special_model_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 = model15.predict_proba(special_model_train)
          # 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()
```

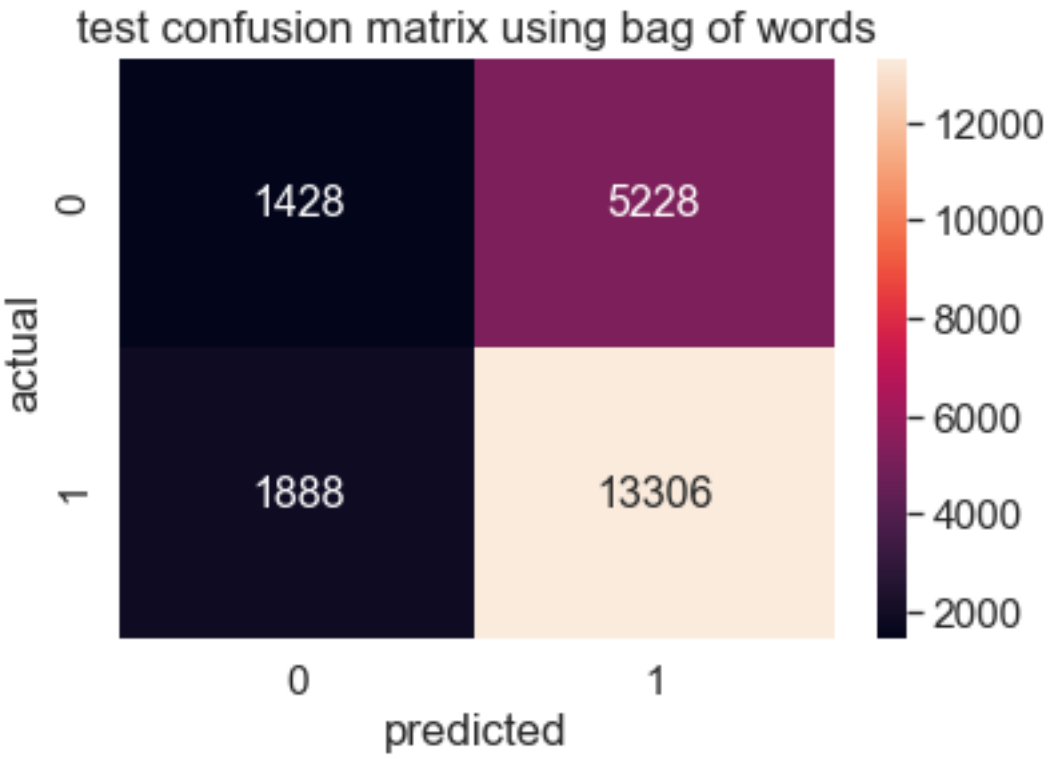
```
AUC: 0.620
AUC: 0.616
```



In [102]:
```
#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=model14.predict(special_model_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("predicted")
plt.ylabel("actual")
plt.show()
```

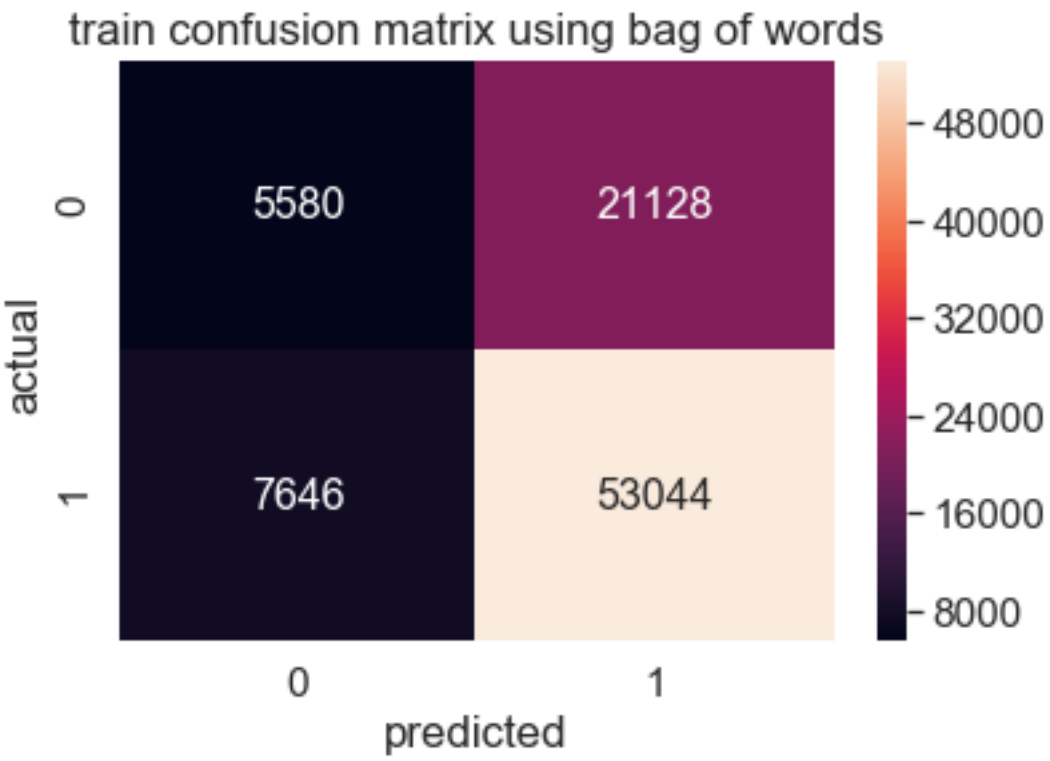
```
1428 1888 5228 13306
true positive rate 0.7179238156900831
true negaitive rate 0.43063932448733416
```

```
[[1428, 5228], [1888, 13306]]
```



In [103]:
```
#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=model14.predict(special_model_train)
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("predicted")
plt.ylabel("actual")
plt.show()
```

```
5580 7646 21128 53044
true positive rate 0.7151485735857197
true negaitive rate 0.4218962649327083
[[5580, 21128], [7646, 53044]]
```



3. Conclusion

```
In [105]: from prettytable import PrettyTable
          x = PrettyTable()
          x.field_names = ["Vectorizer", "Model","penalty", "alpha", "AUC"]
          x.add_row(["BAG of words", "SVM", "l2", 0.01,0.724])
          x.add_row(["TFIDF", "SVM" , "l1", 0.0001,0.663])
          x.add_row(["Average W2V", "SVM","l1", 0.0001,0.663])
          x.add_row(["TFIDF W2V", "SVM" ,"l2",  0.0001,0.642])
          x.add_row(["Secial set", "SVM" ,"l2", 2.5,0.620])
          x.border=True
          print(x)
```

```
+--------------+-------+---------+--------+-------+
|  Vectorizer  | Model | penalty | alpha  |  AUC  |
+--------------+-------+---------+--------+-------+
| BAG of words |  SVM  |    l2   |  0.01  | 0.724 |
```

TFIDF	SVM	l1	0.0001	0.663
Average W2V	SVM	l1	0.0001	0.663
TFIDF W2V	SVM	l2	0.0001	0.642
Secial set	SVM	l2	2.5	0.62

In [ ]: