Objective: To train the model using Decision Tree.

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
```

```
C:\Users\Bhuvana Chandrahasan\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning:
detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

Reading Data

In [2]:

```
project_data = pd.read_csv(r'C:\Users\Bhuvana Chandrahasan\train_data.csv')
resource_data = pd.read_csv(r'C:\Users\Bhuvana Chandrahasan\resources.csv')
```

In [3]:

```
print("Number of data points in train data", project_data.shape)
print(project_data.columns)
```

```
Number of data points in train data (109248, 17)
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'],
      dtype='object')
```

dtype= object)

In [4]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]


project_data.head(2)
```

Out[4]:

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

In [5]:

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

```
Number of data points in resource data (1541272, 4)
Number of data points in resource data Index(['id', 'description', 'quantity', 'price'],
dtype='object')
```

Out[5]:

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

In [6]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
```

Out[6]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [7]:

```
# join two dataframes in python:
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [8]:

```
project_data.head(2)
```

Out[8]:

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

In [9]:

```
project_data.shape
```

Out[9]:

```
(109248, 19)
```

In [10]:

```
final_appr = project_data[project_data['project_is_approved'] == 1]
final_appr = final_appr.sample(frac=0.5,random_state=1)
final_appr.shape
```

Out[10]:

```
(46353, 19)
```

In [11]:

```
final_rej =  project_data[project_data['project_is_approved'] == 0]
final_rej = final_rej.sample(n=5000)
final_rej.shape
```

Out[11]:

```
(5000, 19)
```

In [12]:

```
final=pd.concat([final_appr,final_rej])
final=final.sort_values('Date', axis=0, ascending=True, inplace=False, kind='quicksort', na_positio
n='last')
final.shape
```

Out[12]:

```
(51353, 19)
```

In [13]:

```
project_data = final
```

In [14]:

```
project_data.shape
```

Out[14]:

(51353, 19)

1.1 Preprocessing of project_subject_categories

In [15]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

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

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

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

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

1.2 Preprocessing of project_subject_subcategories

In [16]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
```

```
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]))
```

2.Text Preprocessing

2.1 Preprocessing of essay

In [17]:

```
# 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 [18]:

```
project_data.head(2)
```

Out[18]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Sensory Tools for Focus
3	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	Flexible Seating for Flexible Learning

In [19]:

```
# 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)
```

```
Imagine being 8-9 years old. You're in your third grade classroom. You see bright lights, the kid
next to you is chewing gum, the birds are making noise, the street outside is buzzing with cars, i
t's hot, and your teacher is asking you to focus on learning. Ack! You need a break! So do my stud
ents.Most of my students have autism, anxiety, another disability, or all of the above. It is toug
h to focus in school due to sensory overload or emotions. My students have a lot to deal with in s
chool, but I think that makes them the most incredible kids on the planet. They are kind, caring,
and sympathetic. They know what it's like to be overwhelmed, so they understand when someone else
is struggling. They are open-minded and compassionate. They are the kids who will someday change t
he world.It is tough to do more than one thing at a time. When sensory overload gets in the way, i
t is the hardest thing in the world to focus on learning. My students need many breaks throughout
the day, and one of the best items we've used is a Boogie Board. If we had a few in our own classr
oom, my students could take a break exactly when they need one, regardless of which other rooms in
the school are occupied. Many of my students need to do something with their hands in order to
focus on the task at hand. Putty will give the sensory input they need in order to focus, it will
calm them when they are overloaded, it will help improve motor skills, and it will make school mor
e fun.When my students are able to calm themselves down, they are ready to learn. When they are ab
le to focus, they will learn more and retain more. They will get the sensory input they need and i
```

```
t will prevent meltdowns (which are scary for everyone in the room). This will lead to a better, h
appier classroom community that is able to learn the most they can in the best way possible.
==================================================
Clock ticking. Lights flickering. Perfume. New clothes. These daily encounters often go unnoticed
by most, but these, and many other experiences can cause sensory overload for students with autism
or sensory processing disorders.I have the joy of teaching 12 students in a 2nd-4th grade special
education classroom setting which includes children with mild intellectual disabilities, autism, a
nd other handicapping impairments. Each day provides a new learning experience not only for my stu
dents, but for me as well!Sensory overload occurs when an individual has difficulty processing eve
ryday sensory information.  Sensory overload causes stress and anxiety, which can lead to
withdrawal, challenging behaviors, and even meltdowns. Students can be over- and under-stimulated.
That’s where a sensory room comes in. Sensory rooms are designed to give individuals with sensory
processing disorders the opportunity to learn to relax and self-regulate through learning and prac
ticing stress management techniques, and through sensory integration.  A sensory room provides act
ivities to calm or stimulate the body and mind through each of the senses. Research has shown that
individuals who participate in sensory integration show an increase in calmness, concentration, fo
cus,and alertness, as well as a decrease in aggression. My desire is to create a safe environment
that will provide opportunities for my students to meet their visual, olfactory, oral-motor,
proprioception, tactile, and auditory needs.This sensory room will be used by my students and the
three other special education classes at our school. It will also be available for any student who
needs to have their sensory needs met. By having my project supported I will be able to turn an
empty classroom into a sensory room that will allow me to meet the sensory input needs of my
students, which in turn will allow me to greater meet their academic needs as well. Thank you for
supporting my classroom and my students!
==================================================
My current class has made a lot of gains this year in their reading abilities. They are able to wo
rk cooperatively in groups, self-select working areas that meet their needs, and build stamina in
their reading (read for longer periods of time). Alternative seating will help them to make new ga
ins.My students come from low income households. We are a title one school with free breakfast and
lunch.\r\nMany of these students do not have access to books or Internet or anything else to help
them with their homework. What they learn at school is all the learning that they get in a week. I
n order to maximize our learning, I would like to invest in an alternative seating classroom. This
will provide my students with alternatives to traditional desk seating all day long. They will hav
e the option to stand at a desk, use a stability ball, use a pillow on the carpet, or even to stil
l sit at a traditional desk and chair. These accommodations will provide active students with the
opportunity to move while still learning. It will also help students who learn better when they ar
e in different positions.Students will be able to find comfortable areas to work in instead of
always sitting in uncomfortable hard chairs. This idea is based off of the \"Starbucks Lounge\" mo
del. Think of a Starbucks lounge. Does it have hard chairs and tables or is it an inviting area fo
r people to work in, collaborate in, and engage others in? Students in the classroom work the same
way. They need an engaging and inviting environment to get them to their best learning
state.Donations to this project will help students to learn more about themselves and their
learning needs. It will help them to become more aware of what they need in order to help them
learn. This will in turn lead to more responsibility in students' personal growth.
==================================================
My students are group of 4th graders discovering how the world works around them. They are curious
and ask many questions. \r\nMy classroom is inquiry based. I find out what they know and what they
want to know about how the world works. Their questions drive my instruction. My students are the
future. With our ever changing world, my students need to be given opportunities to not only learn
subject matter, but also need to be given opportunities to create, experiment and build.The Next G
eneration Science Standards are now being implemented in our schools. These new standards go beyon
d students acquiring facts and knowledge but instead concentrate on  students asking questions, de
veloping hypotheses, testing models, making evidence-based arguments and learning other skills tha
t real scientists use all the time. \r\nTeachers today are not equipped with materials to give stu
dents these real experience. My project supplies include \"read aloud\" stories to motivate
students. Having the large white boards will enable students to write, model and design
collaboratively. The 4th grade physical science standards focus on \"energy\". Having the Snap Cir
cuits will give my students an opportunity to build projects they create.nannan
==================================================
```

In [20]:

```
# 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 [21]:

```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My students are group of 4th graders discovering how the world works around them. They are curious
and ask many questions. \r\nMy classroom is inquiry based. I find out what they know and what they
want to know about how the world works. Their questions drive my instruction. My students are the
future. With our ever changing world, my students need to be given opportunities to not only learn
subject matter, but also need to be given opportunities to create, experiment and build.The Next G
eneration Science Standards are now being implemented in our schools. These new standards go beyon
d students acquiring facts and knowledge but instead concentrate on  students asking questions, de
veloping hypotheses, testing models, making evidence-based arguments and learning other skills tha
t real scientists use all the time. \r\nTeachers today are not equipped with materials to give stu
dents these real experience. My project supplies include \"read aloud\" stories to motivate
students. Having the large white boards will enable students to write, model and design
collaboratively. The 4th grade physical science standards focus on \"energy\". Having the Snap Cir
cuits will give my students an opportunity to build projects they create.nannan
==================================================
```

In [22]:

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

```
My students are group of 4th graders discovering how the world works around them. They are curious
and ask many questions.   My classroom is inquiry based. I find out what they know and what they w
ant to know about how the world works. Their questions drive my instruction. My students are the f
uture. With our ever changing world, my students need to be given opportunities to not only learn
subject matter, but also need to be given opportunities to create, experiment and build.The Next G
eneration Science Standards are now being implemented in our schools. These new standards go beyon
d students acquiring facts and knowledge but instead concentrate on  students asking questions, de
veloping hypotheses, testing models, making evidence-based arguments and learning other skills tha
t real scientists use all the time.   Teachers today are not equipped with materials to give stude
nts these real experience. My project supplies include  read aloud  stories to motivate students.
Having the large white boards will enable students to write, model and design collaboratively. The
4th grade physical science standards focus on  energy . Having the Snap Circuits will give my stud
ents an opportunity to build projects they create.nannan
```

In [23]:

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

```
My students are group of 4th graders discovering how the world works around them They are curious
and ask many questions My classroom is inquiry based I find out what they know and what they want
to know about how the world works Their questions drive my instruction My students are the future
With our ever changing world my students need to be given opportunities to not only learn subject
matter but also need to be given opportunities to create experiment and build The Next Generation
Science Standards are now being implemented in our schools These new standards go beyond students
acquiring facts and knowledge but instead concentrate on students asking questions developing hypo
theses testing models making evidence based arguments and learning other skills that real
scientists use all the time Teachers today are not equipped with materials to give students these
real experience My project supplies include read aloud stories to motivate students Having the lar
ge white boards will enable students to write model and design collaboratively The 4th grade physi
cal science standards focus on energy Having the Snap Circuits will give my students an
opportunity to build projects they create nannan
```

In [24]:

```
# https://gist.github.com/sebleier/554280
```

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

In [25]:

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

```
100%|██████████████████████████████████| 51353/51353 [01:40<00:00, 512.22it/s]
```

In [26]:

```
# after preprocesing
preprocessed_essays[20000]
```

Out[26]:

```
'students group 4th graders discovering world works around curious ask many questions classroom in
quiry based find know want know world works questions drive instruction students future ever
changing world students need given opportunities not learn subject matter also need given
opportunities create experiment build next generation science standards implemented schools new st
andards go beyond students acquiring facts knowledge instead concentrate students asking questions
developing hypotheses testing models making evidence based arguments learning skills real
scientists use time teachers today not equipped materials give students real experience project su
pplies include read aloud stories motivate students large white boards enable students write model
design collaboratively 4th grade physical science standards focus energy snap circuits give
students opportunity build projects create nannan'
```

2.2 Preprocessing of project_title

In [27]:

```
# printing some random reviews
print(project_data['project_title'].values[0])
```

```
print("="*50)
print(project_data['project_title'].values[150])
print("="*50)
print(project_data['project_title'].values[1000])
print("="*50)
print(project_data['project_title'].values[20000])
print("="*50)
```

```
Sensory Tools for Focus
==================================================
Soothing to the senses
==================================================
Alternative Seating Classroom
==================================================
Creating Future Engineers\r\n
==================================================
```

In [28]:

```
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 [29]:

```
sent = decontracted(project_data['project_title'].values[20000])
print(sent)
print("="*50)
```

```
Creating Future Engineers\r\n
==================================================
```

In [30]:

```
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
Creating Future Engineers
```

In [31]:

```
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
Creating Future Engineers
```

In [32]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# 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_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████| 51353/51353 [00:05<00:00, 8993.77it/s]
```

In [33]:

```
# after preprocesing
preprocessed_essays[20000]
```

Out[33]:

```
'creating future engineers'
```

In [34]:

```
project_data.head(2)
```

Out[34]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Sensory Tools for Focus
3	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	Flexible Seating for Flexible Learning

In [35]:

```
print(project_data.shape)
print(project_data.columns)
```

```
(51353, 20)
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity', 'clean_categories', 'clean_subcategories',
       'essay'],
      dtype='object')
```

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

In [36]:

```
Y = project_data['project_is_approved'].values
X = project_data
```

In [37]:

```
project_data.shape
```

Out[37]:

```
(51353, 20)
```

In [38]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Flase)# this i
s for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33,stratify=Y) # this is rand
om splitting
```

In [39]:

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(34406, 20) (34406,)
(16947, 20) (16947,)
```

4.Vectorizing Text data

4.1 Essay

In [40]:

```
vectorizer_essay = CountVectorizer(min_df=10, max_features=5000)
vectorizer_essay.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer_essay.transform(X_train['essay'].values)
X_test_essay_bow = vectorizer_essay.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
After vectorizations
(34406, 5000) (34406,)
(16947, 5000) (16947,)
```

4.2 Project_title

In [41]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_title = CountVectorizer(min_df=10, max_features=5000)
vectorizer_title.fit(X_train['project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer_title.transform(X_train['project_title'].values)
X_test_title_bow = vectorizer_title.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_test_title_bow.shape, y_test.shape)
```

```
After vectorizations
(34406, 1669) (34406,)
(16947, 1669) (16947,)
```

4.3 Project_resource_summary

In [42]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_summ = CountVectorizer(min_df=10, max_features=5000)
vectorizer_summ.fit(X_train['project_resource_summary'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_res_sum_bow = vectorizer_summ.transform(X_train['project_resource_summary'].values)
X_test_res_sum_bow = vectorizer_summ.transform(X_test['project_resource_summary'].values)

print("After vectorizations")
print(X_train_res_sum_bow.shape, y_train.shape)
print(X_test_res_sum_bow.shape, y_test.shape)
```

```
After vectorizations
(34406, 3255) (34406,)
(16947, 3255) (16947,)
```

5.Catogorical features:one hot encoding

5.1 Clean_categories

In [43]:

```
vectorizer_cat = CountVectorizer()
vectorizer_cat.fit(X_train['clean_categories'].values) # fit has to happen only on train data


# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_category_ohe = vectorizer_cat.transform(X_train['clean_categories'].values)
X_test_clean_category_ohe = vectorizer_cat.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_category_ohe.shape, y_train.shape)
print(X_test_clean_category_ohe.shape, y_test.shape)
print(vectorizer_cat.get_feature_names())
```

```
After vectorizations
(34406, 9) (34406,)
(16947, 9) (16947,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
```

5.2 Clean_subcategories

In [44]:

```
vectorizer_scat = CountVectorizer()
vectorizer_scat.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data


# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategory_ohe = vectorizer_scat.transform(X_train['clean_subcategories'].values)
X_test_clean_subcategory_ohe = vectorizer_scat.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_clean_subcategory_ohe.shape, y_train.shape)
print(X_test_clean_subcategory_ohe.shape, y_test.shape)
print(vectorizer_scat.get_feature_names())
```

```
After vectorizations
(34406, 30) (34406,)
(16947, 30) (16947,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
```

```
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
```

5.3 Teacher_prefix

In [45]:

```
X_train.teacher_prefix = X_train.teacher_prefix.fillna('')
X_train['teacher_prefix'].value_counts()
```

Out[45]:

```
Mrs.       18017
Ms.        12366
Mr.         3323
Teacher      695
Dr.            3
               2
Name: teacher_prefix, dtype: int64
```

In [46]:

```
X_test.teacher_prefix = X_test.teacher_prefix.fillna('')
X_test['teacher_prefix'].value_counts()
```

Out[46]:

```
Mrs.       8993
Ms.        5941
Mr.        1641
Teacher     369
Dr.           3
Name: teacher_prefix, dtype: int64
```

In [47]:

```
vectorizer_pre = CountVectorizer()
vectorizer_pre.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer_pre.transform(X_train['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer_pre.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer_pre.get_feature_names())
```

```
After vectorizations
(34406, 5) (34406,)
(16947, 5) (16947,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

5.4 School_state

In [48]:

```
vectorizer_st = CountVectorizer()
vectorizer_st.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer_st.transform(X_train['school_state'].values)
X_test_state_ohe = vectorizer_st.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer_st.get_feature_names())
```

```
After vectorizations
(34406, 51) (34406,)
(16947, 51) (16947,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
```

5.5 Project_grade_category

In [49]:

```
X_train.project_grade_category = X_train.project_grade_category.str.replace('\s+', '_')
X_train.project_grade_category = X_train.project_grade_category.str.replace('-', '_')
X_train['project_grade_category'].value_counts()
```

Out[49]:

```
Grades_PreK_2    13895
Grades_3_5       11825
Grades_6_8        5276
Grades_9_12       3410
Name: project_grade_category, dtype: int64
```

In [50]:

```
vectorizer_grd = CountVectorizer(lowercase=False, binary=True)
vectorizer_grd.fit(X_train['project_grade_category'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer_grd.transform(X_train['project_grade_category'].values)
X_test_grade_ohe = vectorizer_grd.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer_grd.get_feature_names())
```

```
After vectorizations
(34406, 4) (34406,)
(16947, 4) (16947,)
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
```

6.Numerical features

6.1 Price

In [51]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
standard_vec.fit(X_train['price'].values.reshape(-1,1))

X_train_price_std = standard_vec.transform(X_train['price'].values.reshape(-1,1))
X_test_price_std = standard_vec.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_std.shape, y_train.shape)
print(X_test_price_std.shape, y_test.shape)
```

```
After vectorizations
(34406, 1) (34406,)
```

```
(16947, 1) (16947,)
```

6.2 Teacher_number_of_previously_posted_projects

In [52]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
standard_vec.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_prev_projects_std =
standard_vec.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1
))
X_test_prev_projects_std =
standard_vec.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))


print("After vectorizations")
print(X_train_prev_projects_std.shape, y_train.shape)
print(X_test_prev_projects_std.shape, y_test.shape)
```

```
C:\Users\Bhuvana Chandrahasan\Anaconda3\lib\site-packages\sklearn\utils\validation.py:429:
DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\Bhuvana Chandrahasan\Anaconda3\lib\site-packages\sklearn\utils\validation.py:429:
DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\Bhuvana Chandrahasan\Anaconda3\lib\site-packages\sklearn\utils\validation.py:429:
DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.


After vectorizations
(34406, 1) (34406,)
(16947, 1) (16947,)
```

6.3 Quantity

In [53]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
standard_vec.fit(X_train['quantity'].values.reshape(-1,1))

X_train_quantity_std = standard_vec.transform(X_train['quantity'].values.reshape(-1,1))
X_test_quantity_std = standard_vec.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_quantity_std.shape, y_train.shape)
print(X_test_quantity_std.shape, y_test.shape)
```

```
C:\Users\Bhuvana Chandrahasan\Anaconda3\lib\site-packages\sklearn\utils\validation.py:429:
DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
C:\Users\Bhuvana Chandrahasan\Anaconda3\lib\site-packages\sklearn\utils\validation.py:429:
DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\Bhuvana Chandrahasan\Anaconda3\lib\site-packages\sklearn\utils\validation.py:429:
DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
After vectorizations
(34406, 1) (34406,)
(16947, 1) (16947,)
```

7. Decision Tree

7.1 Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)

Merging all the above features

In [54]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_train_bow =
hstack((X_train_essay_bow,X_train_title_bow,X_train_res_sum_bow,X_train_clean_category_ohe,X_train_
clean_subcategory_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_train_price_std,X_train_prev_projects_std,X_train_quantity_std)).tocsr()
X_test_bow =
hstack((X_test_essay_bow,X_test_title_bow,X_test_res_sum_bow,X_test_clean_category_ohe,X_test_clear
_subcategory_ohe, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_std,X_test_p
rev_projects_std,X_test_quantity_std)).tocsr()

print("Final Data matrix")
print(X_train_bow.shape, y_train.shape)
print(X_test_bow.shape, y_test.shape)
```

```
Final Data matrix
(34406, 10026) (34406,)
(16947, 10026) (16947,)
```

In [55]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier

DT_BOW = DecisionTreeClassifier()
params = {'max_depth':[1, 5, 10, 50, 100, 500, 100], 'min_samples_split': [5, 10, 100, 500]}


grid = RandomizedSearchCV(DT_BOW , params, cv = 3, scoring = 'roc_auc', random_state = 0)
grid.fit(X_train_bow,y_train)
print(grid.best_params_)
```

```
{'min_samples_split': 500, 'max_depth': 10}
```

In [56]:

```
train_auc= grid.cv_results_['mean_train_score']
train_auc_std= grid.cv_results_['std_train_score']
cv_auc = grid.cv_results_['mean_test_score']
cv_auc_std= grid.cv_results_['std_test_score']
```

In [57]:

```
train_auc
```

Out[57]:

```
array([ 0.53680376,  0.98478171,  0.90674621,  0.9741428 ,  0.66773454,
        0.70552867,  0.97454723,  0.92420598,  0.99899747,  0.97416751])
```

In [58]:

```
cv_auc
```

Out[58]:

```
array([ 0.53680331,  0.54618815,  0.53271272,  0.4926965 ,  0.64093465,
        0.66226525,  0.48516683,  0.49678353,  0.52770977,  0.48795073])
```

In [59]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
```

In [60]:

```
x1=[ 0.53680376,  0.98478171,  0.90674621,  0.9741428 ,  0.66773454,
        0.70552867,  0.97454723,  0.92420598,  0.99899747,  0.97416751]
```

In [61]:

```
x2=[ 0.53680331,  0.54618815,  0.53271272,  0.4926965 ,  0.64093465,
        0.66226525,  0.48516683,  0.49678353,  0.52770977,  0.48795073]
```

In [62]:

```
z1 = pd.Series([1,5,10,10,50,50,100,100,500,500],index = x1)
```

In [63]:

```
y1 = pd.Series([5,10,100,500,5,10,100,500,5,10], index = x1)
```

In [64]:

```
trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
        colorscale='Rainbow',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Rainbow',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
```

```
)
```

In [65]:

```
data = [trace1, trace2]
```

In [66]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- BoW Data',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

In [67]:

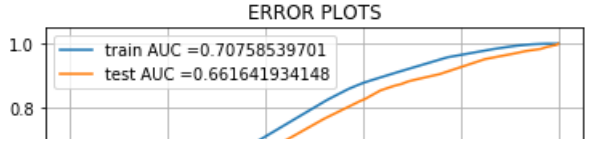
```
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

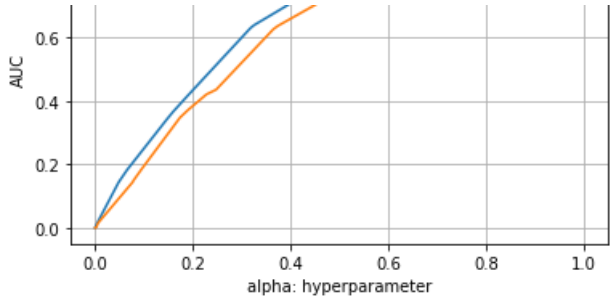
In [68]:

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

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

    y_data_pred.extend(clf.predict_proba(final[tr_loop:])[:,1])

    return y_data_pred
```

In [69]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

DT_BOW = DecisionTreeClassifier(max_depth = 10, min_samples_split = 500)
DT_BOW.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(DT_BOW, X_train_bow)
y_test_pred = batch_predict(DT_BOW, X_test_bow)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





In [70]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [71]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249356204054 for threshold 0.894
[[ 1590  1760]
 [ 5534 25522]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2496 for threshold 0.895
[[  792   858]
 [ 3623 11674]]
```

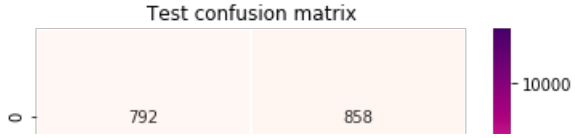
In [72]:

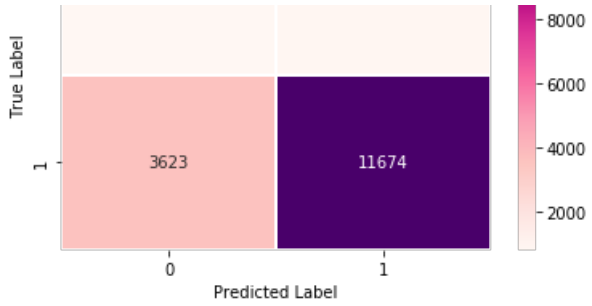
```
frame_confusion_train = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)),range(2),range(2))
frame_confusion_test = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

```
the maximum value of tpr*(1-fpr) 0.249356204054 for threshold 0.894
the maximum value of tpr*(1-fpr) 0.2496 for threshold 0.895
```

In [73]:

```
sns.heatmap(frame_confusion_test, annot = True, fmt="d", cmap="RdPu", linewidths=.5)
plt.title("Test confusion matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```





Plot the box plot with the `price` of these `false positive data points`

In [74]:

```
bow_test = X_test_essay_bow.todense()
bow_test.shape
```

Out[74]:

```
(16947, 5000)
```

In [75]:

```
vectorizer_bow = CountVectorizer(min_df=10)
X = vectorizer_bow.fit(X_train["essay"])
```

In [76]:

```
bow_features = X.get_feature_names()
```

In [77]:

```
len(bow_features)
```

Out[77]:

```
10900
```

In [78]:

```
y_test_converted = list(y_test[::])
```

In [79]:

```
false_positives_index_a = []
fp_count = 0

for i in tqdm(range(len(y_test_pred))):
    if y_test_converted[i] == 0 and y_test_pred[i] <= 0.839:
        false_positives_index_a.append(i)
        fp_count = fp_count + 1
    else :
        continue
```

```
100%|██████████████████████████████████| 16947/16947 [00:00<00:00, 385043.04it/s]
```

In [80]:

```
fp_count
```

Out[80]:

```
530
```

In [81]:

```
false_positives_index_a[0:5]
```

Out[81]:

```
[24, 76, 121, 124, 149]
```

In [82]:

```
df1 = pd.DataFrame(bow_test)
```

In [83]:

```
df1_final = df1.iloc[false_positives_index_a,:]
```

In [84]:

```
df1_final.shape
```

Out[84]:

```
(530, 5000)
```

In [85]:

```
df1_final[0].sum()
```

Out[85]:

```
0
```

In [87]:

```
best_indices = []

for j in range(5000):

    s = df1_final[j].sum()

    if s >= 100 :
        best_indices.append(j)
    else :
        continue
```

In [88]:

```
len(best_indices)
```

Out[88]:

```
191
```

In [89]:

```
best_indices[0:10]
```

Out[89]:

```
[85, 86, 145, 225, 227, 239, 245, 261, 269, 321]
```

In [90]:

```
bow_features[0:10]
```

Out[90]:

```
['00', '000', '10', '100', '1000', '101', '10th', '11', '110', '1100']
```

['00' , '000' , '10' , '100' , '1000' , '101' , '10th' , '11' , '110' , '1100']

In [91]:

```
fp_words = []

for a in best_indices :
    fp_words.append(str(bow_features[a]))
```

In [92]:

```
fp_words[0:10]
```

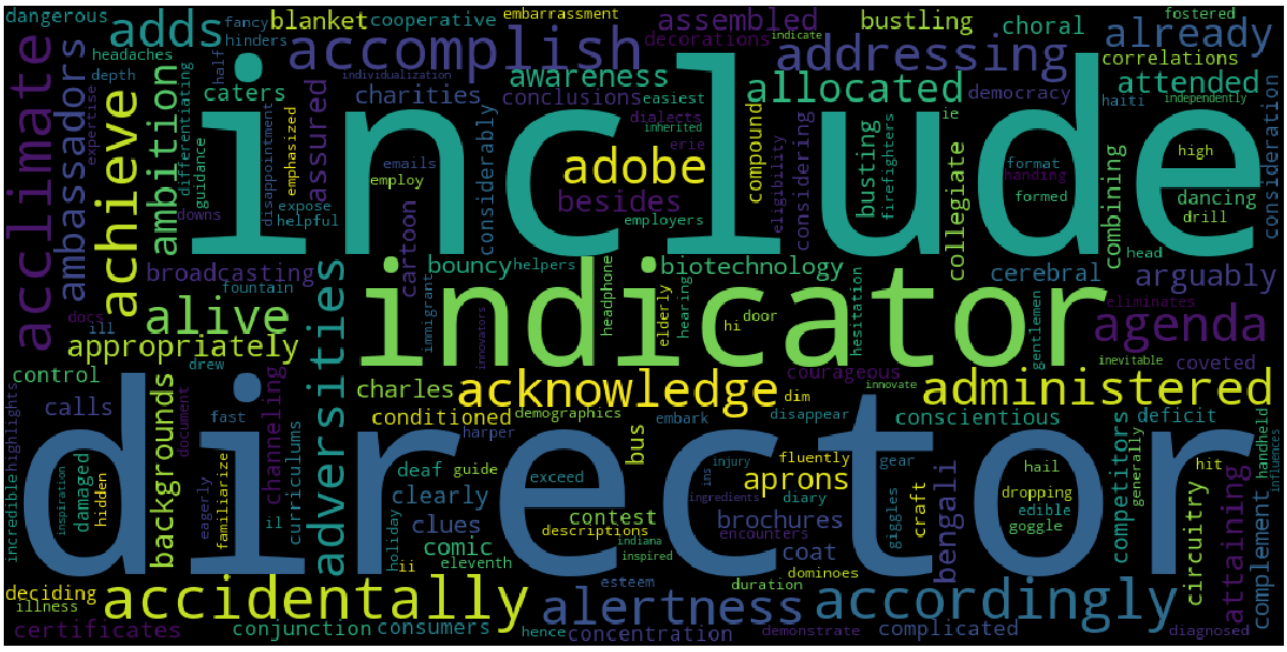
Out[92]:

```
['38',
 '39',
 '78',
 'accidentally',
 'acclimate',
 'accomplish',
 'accordingly',
 'achieve',
 'acknowledge',
 'addressing']
```

Word Cloud

In [93]:

```
from wordcloud import WordCloud
#convert list to string and generate
unique_string=(" ").join(fp_words)
wordcloud = WordCloud(width = 1000, height = 500).generate(unique_string)
plt.figure(figsize=(25,10))
plt.imshow(wordcloud)
plt.axis("off")
plt.savefig("your_file_name"+".png", bbox_inches='tight')
plt.show()
plt.close()
```



Graphviz_bow

In [122]:

```
feature_names_bow = []
```

In [123]:

```
#categorical and text features

for i in vectorizer_essay.get_feature_names() :
    feature_names_bow.append(i)
for i in vectorizer_title.get_feature_names() :
    feature_names_bow.append(i)
for i in vectorizer_summ.get_feature_names() :
    feature_names_bow.append(i)
for i in vectorizer_cat.get_feature_names() :
    feature_names_bow.append(i)
for i in vectorizer_scat.get_feature_names() :
    feature_names_bow.append(i)
for i in vectorizer_grd.get_feature_names() :
    feature_names_bow.append(i)
for i in vectorizer_st.get_feature_names() :
    feature_names_bow.append(i)
for i in vectorizer_pre.get_feature_names() :
    feature_names_bow.append(i)
```

In [125]:

```
#numerical features

feature_names_bow.append("price")
feature_names_bow.append("quantity")
feature_names_bow.append("teacher_number_of_previously_posted_projects")
```

In [126]:

```
len(feature_names_bow)
```

Out[126]:

```
10053
```

In [137]:

```
from sklearn.tree import export_graphviz

# train model on the best alpha
DT_BOW = DecisionTreeClassifier(max_depth=2,min_samples_split=500)

# fitting the model on crossvalidation train
DT_BOW.fit(X_train_bow, y_train)

dot_data = export_graphviz(DT_BOW,
                           feature_names=feature_names_bow,
                           class_names=["+","-"],
                           out_file='DT_BOW.dot',
                           filled=True,
                           rounded=True)
```

Box Plot

In [166]:

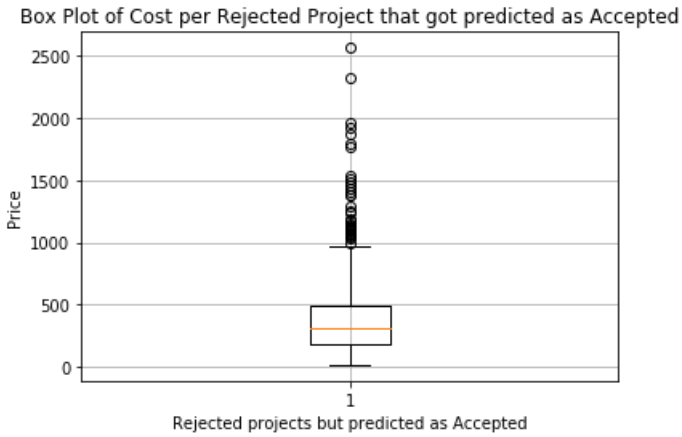
```
price = pd.DataFrame(X_test['price'])
```

In [167]:

```
price = price.iloc[false_positives_index_a,:]
```

In [168]:

```
plt.boxplot(price.values)
plt.title('Box Plot of Cost per Rejected Project that got predicted as Accepted')
plt.xlabel('Rejected projects but predicted as Accepted')
plt.ylabel('Price')
plt.grid()
plt.show()
```



PDF - Teacher_number_of_previously_posted_projects of these False Positive data points

In [169]:

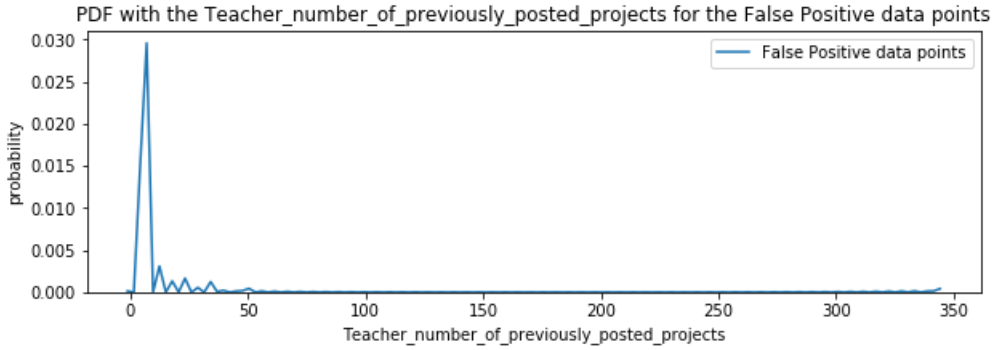
```
prev_proj = pd.DataFrame(X_test['teacher_number_of_previously_posted_projects'])
```

In [170]:

```
prev_proj = prev_proj.iloc[false_positives_index_a,:]
```

In [171]:

```
plt.figure(figsize=(10,3))
sns.distplot(prev_proj.values, hist=False, label="False Positive data points")
plt.title('PDF with the Teacher_number_of_previously_posted_projects for the False Positive data p
oints')
plt.xlabel('Teacher_number_of_previously_posted_projects')
plt.ylabel('probability')
plt.legend()
plt.show()
```



SET 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)

In [94]:

```
vectorizer_essay = TfidfVectorizer(min_df=10)
vectorizer_essay.fit(X_train['essay'].values) # fit has to happen only on train data
```

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

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
After vectorizations
(34406, 10900) (34406,)
(16947, 10900) (16947,)
```

In [95]:

```
vectorizer_title = TfidfVectorizer(min_df=10)
vectorizer_title.fit(X_train['project_title'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_test_title_tfidf.shape, y_test.shape)
```

```
After vectorizations
(34406, 1669) (34406,)
(16947, 1669) (16947,)
```

In [96]:

```
vectorizer_summ = TfidfVectorizer(min_df=10)
vectorizer_summ.fit(X_train['project_resource_summary'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_res_sum_tfidf = vectorizer_summ.transform(X_train['project_resource_summary'].values)
X_test_res_sum_tfidf = vectorizer_summ.transform(X_test['project_resource_summary'].values)

print("After vectorizations")
print(X_train_res_sum_tfidf.shape, y_train.shape)
print(X_test_res_sum_tfidf.shape, y_test.shape)
```

```
After vectorizations
(34406, 3255) (34406,)
(16947, 3255) (16947,)
```

In [97]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_train_tfidf =
hstack((X_train_essay_tfidf,X_train_title_tfidf,X_train_res_sum_tfidf,X_train_clean_category_ohe,X
_train_clean_subcategory_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_train_price_std,X_train_prev_projects_std,X_train_quantity_std)).tocsr()
X_test_tfidf =
hstack((X_test_essay_tfidf,X_test_title_tfidf,X_test_res_sum_tfidf,X_test_clean_category_ohe,X_tes
t_clean_subcategory_ohe, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_std,X
_test_prev_projects_std,X_test_quantity_std)).tocsr()

print("Final Data matrix")
print(X_train_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
Final Data matrix
(34406, 15926) (34406,)
(16947, 15926) (16947,)
```

In [98]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier

DT_TFIDF = DecisionTreeClassifier()
params = {'max_depth':[1, 5, 10, 50, 100, 500, 100], 'min_samples_split': [5, 10, 100, 500]}


grid = RandomizedSearchCV(DT_TFIDF , params, cv = 3, scoring = 'roc_auc', random_state = 0)
grid.fit(X_train_tfidf,y_train)
print(grid.best_params_)
```

```
{'min_samples_split': 500, 'max_depth': 10}
```

In [99]:

```
train_auc= grid.cv_results_['mean_train_score']
train_auc_std= grid.cv_results_['std_train_score']
cv_auc = grid.cv_results_['mean_test_score']
cv_auc_std= grid.cv_results_['std_test_score']
```

In [100]:

```
train_auc
```

Out[100]:

```
array([ 0.53614609,  0.99072468,  0.87332719,  0.94064112,  0.66221625,
        0.69276502,  0.94588404,  0.88661323,  0.99949907,  0.94348158])
```

In [101]:

```
cv_auc
```

Out[101]:

```
array([ 0.534352  ,  0.54993029,  0.5446959 ,  0.49000216,  0.6353686 ,
        0.6377913 ,  0.47911785,  0.51038463,  0.52797567,  0.48680643])
```

In [102]:

```
x1=[ 0.53614609,  0.99072468,  0.87332719,  0.94064112,  0.66221625,
        0.69276502,  0.94588404,  0.88661323,  0.99949907,  0.94348158]
```

In [103]:

```
x2=[ 0.534352  ,  0.54993029,  0.5446959 ,  0.49000216,  0.6353686 ,
        0.6377913 ,  0.47911785,  0.51038463,  0.52797567,  0.48680643]
```

In [104]:

```
z1 = pd.Series([1,5,10,10,50,50,100,100,500,500],index = x1)
```

In [105]:

```
y1 = pd.Series([5,10,100,500,5,10,100,500,5,10], index = x1)
```

In [106]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
```

```
    marker=dict(
        size=4,
        colorscale='Rainbow',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Rainbow',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
)
```

In [107]:

```
data = [trace1, trace2]
```

In [108]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- BoW Data',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

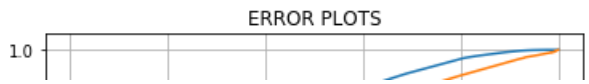
In [109]:

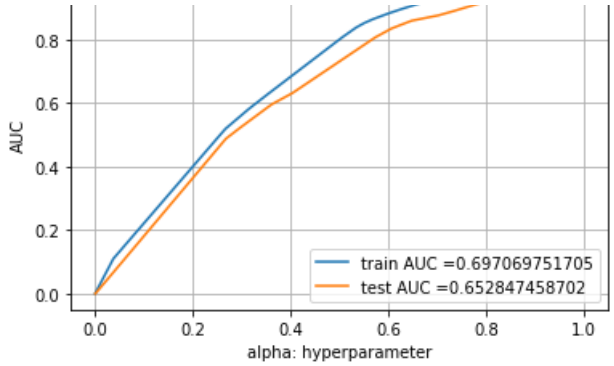
```
fig = go.Figure(data=data, layout=layout)
```

```
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [110]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

DT_TFIDF = DecisionTreeClassifier(max_depth = 10, min_samples_split = 500)
DT_TFIDF.fit(X_train_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs


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

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





In [111]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249960703943 for threshold 0.916
[[ 1654  1696]
 [ 6004 25052]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.244532966024 for threshold 0.916
[[  703   947]
 [ 2959 12338]]
```

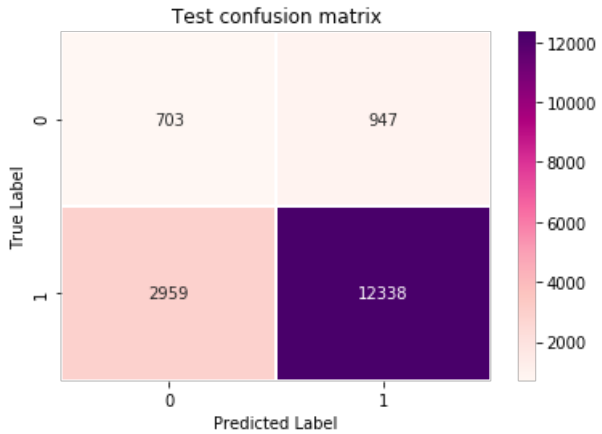
In [112]:

```
frame_confusion_train = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)),range(2),range(2))
frame_confusion_test = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

```
the maximum value of tpr*(1-fpr) 0.249960703943 for threshold 0.916
the maximum value of tpr*(1-fpr) 0.244532966024 for threshold 0.916
```

In [113]:

```
sns.heatmap(frame_confusion_test, annot = True, fmt="d", cmap="RdPu", linewidths=.5)
plt.title("Test confusion matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



In [114]:

```
tfidf_test = X_test_essay_tfidf.todense()
```

In [116]:

```
tfidf_test.shape
```

Out[116]:

```
(16947, 10900)
```

In [117]:

```
vectorizer_tfidf = CountVectorizer(min_df=10)
Y = vectorizer_tfidf.fit(X_train["essay"])
```

In [118]:

```
tfidf_features = Y.get_feature_names()
```

In [119]:

```
len(tfidf_features)
```

Out[119]:

```
10900
```

In [120]:

```
y_test_converted = list(y_test[::])
```

In [121]:

```
false_positives_index_b = []
fp_count = 0

for i in tqdm(range(len(y_test_pred))):
    if y_test_converted[i] == 0 and y_test_pred[i] <= 0.839:
        false_positives_index_b.append(i)
        fp_count = fp_count + 1
    else :
        continue
```

```
100%|██████████████████████████████████| 16947/16947 [00:00<00:00, 282008.75it/s]
```

In [122]:

```
fp_count
```

Out[122]:

```
489
```

In [123]:

```
false_positives_index_b[0:5]
```

Out[123]:

```
[24, 50, 51, 76, 112]
```

In [124]:

```
df2 = pd.DataFrame(tfidf_test)
```

In [125]:

```
df2_final = df2.iloc[false_positives_index_b,:]
```

In [126]:

```
df2_final.shape
```

Out[126]:

```
(489, 10900)
```

In [127]:

```
df2_final[0].sum()
```

Out[127]:

```
0.12351088739679801
```

In [139]:

```
best_indices_b = []

for j in range(10900):

    s = df2_final[j].sum()

    if s >= 10:
        best_indices_b.append(j)
    else :
        continue
```

In [140]:

```
len(best_indices_b)
```

Out[140]:

```
47
```

In [141]:

```
best_indices_b[0:10]
```

Out[141]:

```
[475, 549, 670, 722, 781, 968, 1450, 1700, 1780, 1901]
```

In [142]:

```
tfidf_features[0:10]
```

Out[142]:

```
['00', '000', '10', '100', '1000', '101', '10th', '11', '110', '1100']
```

In [143]:

```
fp_words_b = []

for a in best_indices_b :
    fp_words_b.append(str(tfidf_features[a]))
```

In [144]:

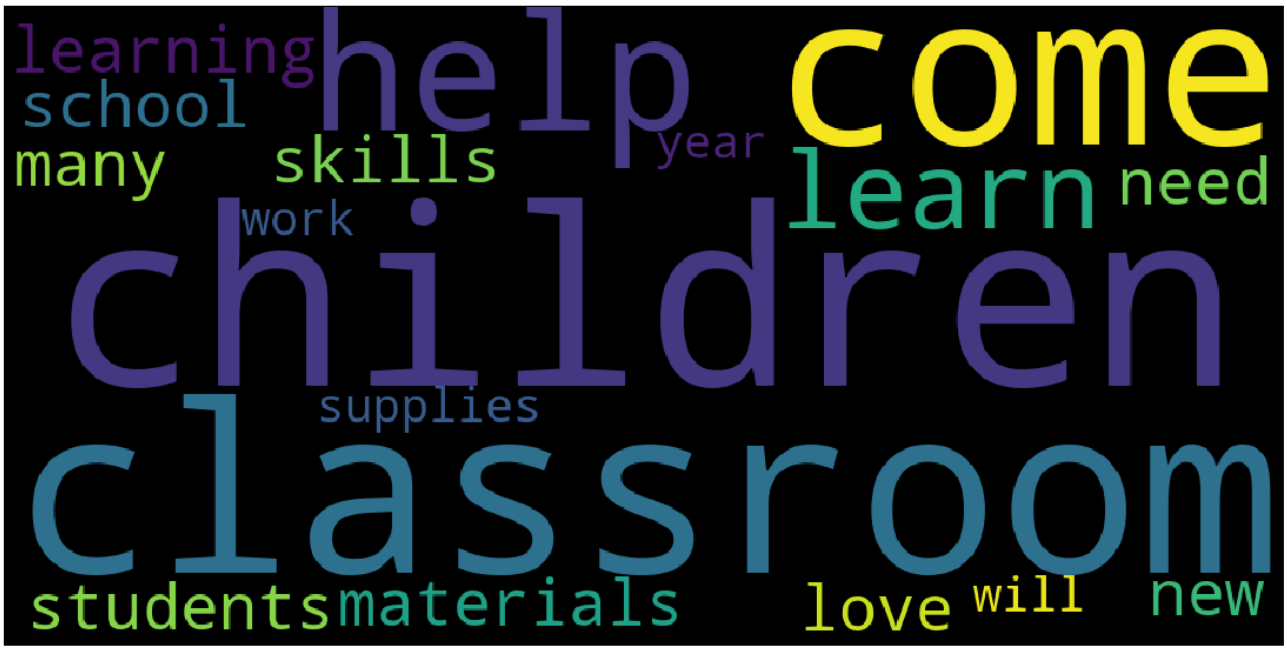
```
fp_words_b[0:10]
```

Out[144]:

['all', 'and', 'are', 'as', 'at', 'be', 'can', 'children', 'classroom', 'come']

In [145]:

```
from wordcloud import WordCloud
#convert list to string and generate
unique_string=(" ").join(fp_words_b)
wordcloud = WordCloud(width = 1000, height = 500).generate(unique_string)
plt.figure(figsize=(20,10))
plt.imshow(wordcloud)
plt.axis("off")
plt.savefig("your_file_name"+".png", bbox_inches='tight')
plt.show()
plt.close()
```



Graphviz

In [104]:

```
feature_names_tfidf =[]
```

In [105]:

```
#categorical and text features

for i in vectorizer_essay.get_feature_names() :
    feature_names_tfidf.append(i)
for i in vectorizer_title.get_feature_names() :
    feature_names_tfidf.append(i)
for i in vectorizer_summ.get_feature_names() :
    feature_names_tfidf.append(i)
for i in vectorizer_cat.get_feature_names() :
    feature_names_tfidf.append(i)
for i in vectorizer_scat.get_feature_names() :
    feature_names_tfidf.append(i)
for i in vectorizer_grd.get_feature_names() :
    feature_names_tfidf.append(i)
for i in vectorizer_st.get_feature_names() :
    feature_names_tfidf.append(i)
for i in vectorizer_pre.get_feature_names() :
    feature_names_tfidf.append(i)
```

In [106]:

```
#numerical features
```

```
feature_names_tfidf.append("price")
feature_names_tfidf.append("quantity")
feature_names_tfidf.append("teacher_number_of_previously_posted_projects")
```

In [107]:

```
len(feature_names_tfidf)
```

Out[107]:

16013

In [108]:

```
from sklearn.tree import export_graphviz

# train model on the best alpha
DT_TFIDF = DecisionTreeClassifier(max_depth=2,min_samples_split=500)

# fitting the model on crossvalidation train
DT_TFIDF.fit(X_train_tfidf, y_train)

dot_data = export_graphviz(DT_TFIDF,
                           feature_names=feature_names_tfidf,
                           class_names=["+","-"],
                           out_file='DT_TFIDF.dot',
                           filled=True,
                           rounded=True)
```

Box Plot

In [263]:

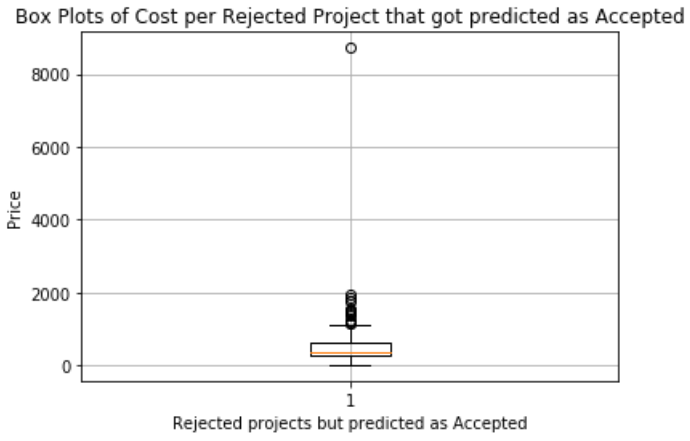
```
price = pd.DataFrame(X_test['price'])
```

In [264]:

```
price = price.iloc[false_positives_index_b,:]
```

In [265]:

```
plt.boxplot(price.values)
plt.title('Box Plots of Cost per Rejected Project that got predicted as Accepted')
plt.xlabel('Rejected projects but predicted as Accepted')
plt.ylabel('Price')
plt.grid()
plt.show()
```



PDF for teacher_number_of_previously_posted_projects

In [148]:

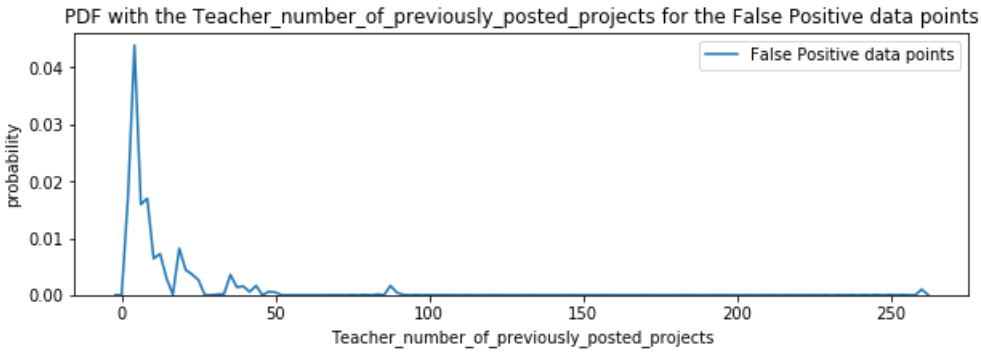
```
prev_proj = pd.DataFrame(X_test['teacher_number_of_previously_posted_projects'])
```

In [149]:

```
prev_proj = prev_proj.iloc[false_positives_index_b,:]
```

In [150]:

```
plt.figure(figsize=(10,3))
sns.distplot(prev_proj.values, hist=False, label="False Positive data points")
plt.title('PDF with the Teacher_number_of_previously_posted_projects for the False Positive data p
oints')
plt.xlabel('Teacher_number_of_previously_posted_projects')
plt.ylabel('probability')
plt.legend()
plt.show()
```



Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)

In [54]:

```
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
import numpy as np
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(r'C:\Users\Bhuvana Chandrahasan\glove.42B.300d.txt',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
model = loadGloveModel('glove.42B.300d.txt')
```

```
Loading Glove Model
```

```
1917495it [17:40, 1808.19it/s]
```

```
Done. 1917495  words loaded!
```

In [55]:

```
words = []
for i in project_data['essay']:
    words.extend(i.split(' '))

for i in project_data['project_title']:
    words.extend(i.split(' '))
print("all the words in the corpus", len(words))
```

```
print("all the words in the corpus", len(words))
words = set(words)
print("the unique words in the corpus", len(words))

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

```
all the words in the corpus 13634993
the unique words in the corpus 227745
The number of words that are present in both glove vectors and our corpus 34931 ( 15.338 %)
```

In [56]:

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

```
word 2 vec length 34931
```

In [57]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_corpus, f)
```

In [58]:

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

In [86]:

```
# average Word2V# compute average word2vec for each review.
X_train_essay_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay']): # 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
    X_train_essay_avg_w2v_vectors.append(vector)
print("X_train")
print(len(X_train_essay_avg_w2v_vectors))
print(len(X_train_essay_avg_w2v_vectors[0]))


# average Word2V# compute average word2vec for each review.
X_test_essay_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay']): # 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
    X_test_essay_avg_w2v_vectors.append(vector)
print("X_test")
```

```
print(len(X_test_essay_avg_w2v_vectors))
print(len(X_test_essay_avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████████| 34406/34406 [08:39<00:00, 66.19it/s]
```

```
X_train
34406
300
```

```
100%|████████████████████████████████████| 16947/16947 [00:54<00:00, 308.25it/s]
```

```
X_test
16947
300
```

In [87]:

```
X_train_essay_avg_w2v_vectors = np.array(X_train_essay_avg_w2v_vectors)
X_test_essay_avg_w2v_vectors = np.array(X_test_essay_avg_w2v_vectors)
```

In [88]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title']): # 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
    X_train_avg_w2v_vectors.append(vector)
print("X_train")
print(len(X_train_avg_w2v_vectors))
print(len(X_train_avg_w2v_vectors[0]))

# average Word2Vec
# compute average word2vec for each review.
X_test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title']): # 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
    X_test_avg_w2v_vectors.append(vector)
print("X_test")
print(len(X_test_avg_w2v_vectors))
print(len(X_test_avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████████| 34406/34406 [00:02<00:00, 12861.08it/s]
```

```
X_train
34406
300
```

```
100%|████████████████████████████████████| 16947/16947 [00:00<00:00, 19599.15it/s]
```

```
X_test
16947
300
```

In [89]:

In [89]:

```
X_train_avg_w2v_vectors = np.array(X_train_avg_w2v_vectors)
X_test_avg_w2v_vectors = np.array(X_test_avg_w2v_vectors)
```

In [90]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_train_w2v =
hstack((X_train_essay_avg_w2v_vectors,X_train_avg_w2v_vectors,X_train_clean_category_ohe,X_train_clean_subcategory_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_std, X_train_prev_projects_std,X_train_quantity_std)).tocsr()
X_test_w2v = hstack((X_test_essay_avg_w2v_vectors,X_test_avg_w2v_vectors,X_test_clean_category_ohe,X_test_clean_subcategory_ohe, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_std,X_test_prev_projects_std,X_test_quantity_std)).tocsr()

print("Final Data matrix")
print(X_train_w2v.shape, y_train.shape)
print(X_test_w2v.shape, y_test.shape)
```

```
Final Data matrix
(34406, 702) (34406,)
(16947, 702) (16947,)
```

In [109]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier

DT_W2V = DecisionTreeClassifier()
params = {'max_depth':[1, 5, 10, 50, 100, 500, 100], 'min_samples_split': [5, 10, 100, 500]}


grid = RandomizedSearchCV(DT_W2V , params, cv = 3, scoring = 'roc_auc', random_state = 0)
grid.fit(X_train_w2v,y_train)
print(grid.best_params_)
```

```
{'min_samples_split': 10, 'max_depth': 5}
```

In [110]:

```
train_auc= grid.cv_results_['mean_train_score']
train_auc_std= grid.cv_results_['std_train_score']
cv_auc = grid.cv_results_['mean_test_score']
cv_auc_std= grid.cv_results_['std_test_score']
```

In [111]:

```
train_auc
```

Out[111]:

```
array([ 0.57295902,  0.96015586,  0.95473877,  0.99534848,  0.67452667,
        0.72567406,  0.9992328 ,  0.99127962,  0.99924379,  0.995698  ])
```

In [112]:

```
cv_auc
```

Out[112]:

```
array([ 0.57113245,  0.55087784,  0.5363499 ,  0.52651983,  0.63952032,
        0.63569687,  0.5235427 ,  0.5045484 ,  0.52823669,  0.52499898])
```

In [115]:

```
x1=[ 0.57295902,  0.96015586,  0.95473877,  0.99534848,  0.67452667,
```

```
        0.72567406,  0.9992328 ,  0.99127962,  0.99924379,  0.995698  ]
```

In [116]:

```
x2=[ 0.57113245,  0.55087784,  0.5363499 ,  0.52651983,  0.63952032,
        0.63569687,  0.5235427 ,  0.5045484 ,  0.52823669,  0.52499898]
```

In [117]:

```
z1 = pd.Series([1,5,10,10,50,50,100,100,500,500],index = x1)
```

In [118]:

```
y1 = pd.Series([5,10,100,500,5,10,100,500,5,10], index = x1)
```

In [119]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
        colorscale='Rainbow',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Rainbow',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
)
```

In [120]:

```
data=[trace1,trace2]
```

In [121]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- BoW Data',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
```

```
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

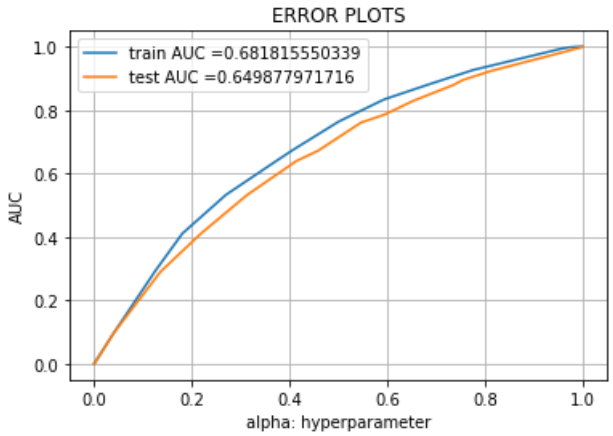
In [122]:

```
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [124]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
```

```
from sklearn.metrics import roc_curve, auc


DT_W2V = DecisionTreeClassifier(max_depth = 5, min_samples_split = 10)
DT_W2V.fit(X_train_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs


y_train_pred = batch_predict(DT_W2V, X_train_w2v)
y_test_pred = batch_predict(DT_W2V, X_test_w2v)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [125]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999919804 for threshold 0.898
[[ 1672  1678]
 [ 7332 23724]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.248301561065 for threshold 0.898
[[  748   902]
 [ 3659 11638]]
```

In [126]:

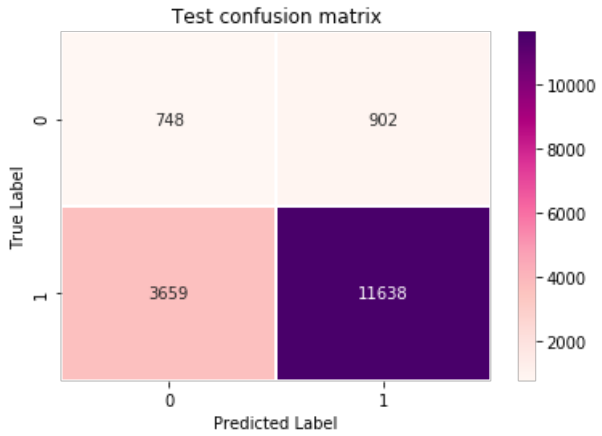
```
frame_confusion_train = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)),range(2),range(2))
frame_confusion_test = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

```
the maximum value of tpr*(1-fpr) 0.24999919804 for threshold 0.898
the maximum value of tpr*(1-fpr) 0.248301561065 for threshold 0.898
```

In [127]:

```
sns.heatmap(frame_confusion_test, annot=True, fmt="d", cmap="RdPu", linewidths=.5)
```

```
sns.heatmap(frame_confusion_test, annot = True, fmt="d", cmap="RdPu", linewidths=.5)
plt.title("Test confusion matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

In [59]:

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

In [60]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_train['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_essay_tfidf_w2v_vectors.append(vector)

print("X_train_essay_tfidf_w2v")
print(len(X_train_essay_tfidf_w2v_vectors))
print(len(X_train_essay_tfidf_w2v_vectors[0]))

# average Word2Vec
# compute average word2vec for each review.
X_test_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
```

```
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_essay_tfidf_w2v_vectors.append(vector)

print("X_test_essay_tfidf_w2v")
print(len(X_test_essay_tfidf_w2v_vectors))
print(len(X_test_essay_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████| 34406/34406 [11:38<00:00, 49.27it/s]
```

```
X_train_essay_tfidf_w2v
34406
300
```

```
100%|████████████████████████████████████| 16947/16947 [05:47<00:00, 48.78it/s]
```

```
X_test_essay_tfidf_w2v
16947
300
```

In [61]:

```
X_train_essay_tfidf_w2v_vectors = np.array(list(X_train_essay_tfidf_w2v_vectors))
X_test_essay_tfidf_w2v_vectors = np.array(list(X_test_essay_tfidf_w2v_vectors))
```

In [62]:

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

In [63]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_train['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_title_tfidf_w2v_vectors.append(vector)

print("X_train_title_tfidf_w2v")
print(len(X_train_title_tfidf_w2v_vectors))
print(len(X_train_title_tfidf_w2v_vectors[0]))

# average Word2Vec
# compute average word2vec for each review.
X_test_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
```

```
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_title_tfidf_w2v_vectors.append(vector)

print("X_test_title_tfidf_w2v")
print(len(X_test_title_tfidf_w2v_vectors))
print(len(X_test_title_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████| 34406/34406 [00:02<00:00, 12733.58it/s]
```

```
X_train_title_tfidf_w2v
34406
300
```

```
100%|████████████████████████████████| 16947/16947 [00:01<00:00, 11204.22it/s]
```

```
X_test_title_tfidf_w2v
16947
300
```

In [64]:

```
X_train_title_tfidf_w2v_vectors = np.array(list(X_train_title_tfidf_w2v_vectors))
X_test_title_tfidf_w2v_vectors = np.array(list(X_test_title_tfidf_w2v_vectors))
```

In [84]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_train_tfidf_w2v =
hstack((X_train_essay_tfidf_w2v_vectors,X_train_title_tfidf_w2v_vectors,X_train_clean_category_ohe
,X_train_clean_subcategory_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_train_price_std,X_train_prev_projects_std,X_train_quantity_std)).tocsr()
X_test_tfidf_w2v =
hstack((X_test_essay_tfidf_w2v_vectors,X_test_title_tfidf_w2v_vectors,X_test_clean_category_ohe,X_
test_clean_subcategory_ohe, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
X_test_price_std,X_test_prev_projects_std,X_test_quantity_std)).tocsr()

print("Final Data matrix")
print(X_train_tfidf_w2v.shape, y_train.shape)
print(X_test_tfidf_w2v.shape, y_test.shape)
```

```
Final Data matrix
(34406, 702) (34406,)
(16947, 702) (16947,)
```

In [66]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier

DT_TFIDF_W2V = DecisionTreeClassifier()
params = {'max_depth':[1, 5, 10, 50, 100, 500, 100], 'min_samples_split': [5, 10, 100, 500]}


grid = RandomizedSearchCV(DT_TFIDF_W2V , params, cv = 3, scoring = 'roc_auc', random_state = 0)
grid.fit(X_train_tfidf_w2v,y_train)
print(grid.best_params_)
```

```
{'min_samples_split': 500, 'max_depth': 10}
```

In [67]:

```
train_auc= grid.cv_results_['mean_train_score']
train_auc_std= grid.cv_results_['std_train_score']
cv_auc = grid.cv_results_['mean_test_score']
cv_auc_std= grid.cv_results_['std_test_score']
```

In [68]:

```
train_auc
```

Out[68]:

```
array([ 0.57543333,  0.95270901,  0.95280367,  0.99551608,  0.6867124 ,
        0.73427425,  0.99935731,  0.99522332,  0.99935414,  0.9957447 ])
```

In [70]:

```
cv_auc
```

Out[70]:

```
array([ 0.57314205,  0.56779182,  0.57056551,  0.52896091,  0.64780811,
        0.65132886,  0.53142132,  0.52622823,  0.52821305,  0.53351032])
```

In [71]:

```
x1=[ 0.57543333,  0.95270901,  0.95280367,  0.99551608,  0.6867124 ,
        0.73427425,  0.99935731,  0.99522332,  0.99935414,  0.9957447 ]
```

In [72]:

```
x2=[[ 0.57314205,  0.56779182,  0.57056551,  0.52896091,  0.64780811,
        0.65132886,  0.53142132,  0.52622823,  0.52821305,  0.53351032]]
```

In [73]:

```
z1 = pd.Series([1,5,10,10,50,50,100,100,500,500],index = x1)
```

In [74]:

```
y1 = pd.Series([5,10,100,500,5,10,100,500,5,10], index = x1)
```

In [75]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
        colorscale='Rainbow',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Rainbow',
    ),
```

```
    line=dict(
        color='#b45c1f',
        width=1
    )
)
```

In [76]:

```
data=[trace1,trace2]
```

In [77]:

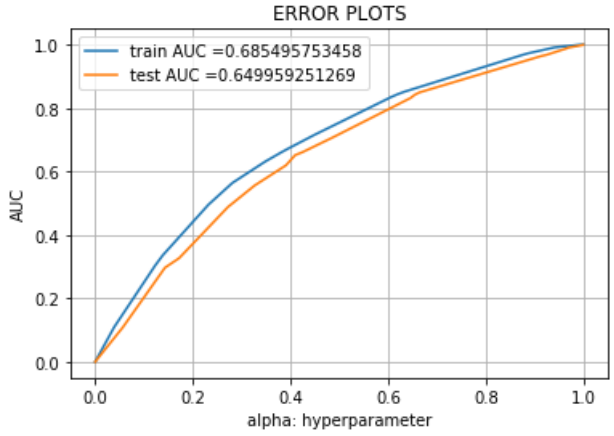
```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- BoW Data',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

In [78]:

```
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [85]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


DT_TFIDF_W2V = DecisionTreeClassifier(max_depth = 5, min_samples_split = 10)
DT_TFIDF_W2V.fit(X_train_tfidf_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(DT_TFIDF_W2V, X_train_tfidf_w2v)
y_test_pred = batch_predict(DT_TFIDF_W2V, X_test_tfidf_w2v)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [91]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.247424816217 for threshold 0.882
[[ 1845  1505]
 [ 8826 22230]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.249805693297 for threshold 0.882
[[  848   802]
 [ 4480 10817]]
```

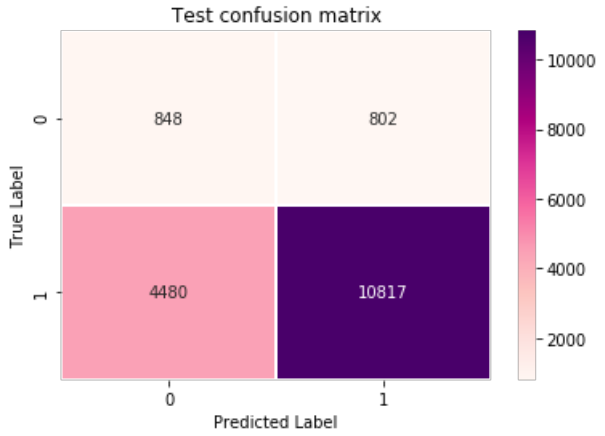
In [92]:

```
frame_confusion_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds
, train_fpr, train_fpr)),range(2),range(2))
frame_confusion_test = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)),range(2),range(2))
```

```
the maximum value of tpr*(1-fpr) 0.247424816217 for threshold 0.882
the maximum value of tpr*(1-fpr) 0.249805693297 for threshold 0.882
```

In [93]:

```
sns.heatmap(frame_confusion_test, annot = True, fmt="d", cmap="RdPu", linewidths=.5)
plt.title("Test confusion matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



Set 5: Select best5k from set2 using feature importance

In [99]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_train_tfidf =
hstack((X_train_essay_tfidf,X_train_title_tfidf,X_train_res_sum_tfidf,X_train_clean_category_ohe,X
_train_clean_subcategory_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_train_price_std,X_train_prev_projects_std,X_train_quantity_std)).tocsr()
X_test_tfidf =
hstack((X_test_essay_tfidf,X_test_title_tfidf,X_test_res_sum_tfidf,X_test_clean_category_ohe,X_tes
t_clean_subcategory_ohe, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_std,X
_test_prev_projects_std,X_test_quantity_std)).tocsr()

print("Final Data matrix")
print(X_train_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
Final Data matrix
```

```
Final Data matrix
(34406, 15925) (34406,)
(16947, 15925) (16947,)
```

In [100]:

```
from sklearn.feature_selection import SelectKBest, chi2

X_train_set5 = SelectKBest(chi2, k=5000).fit_transform(X_train_tfidf, y_train)
X_test_set5 = SelectKBest(chi2, k=5000).fit_transform(X_test_tfidf, y_test)
```

In [101]:

```
print(X_train_set5.shape, y_train.shape)
print(X_test_set5.shape, y_test.shape)
```

```
(34406, 5000) (34406,)
(16947, 5000) (16947,)
```

In [102]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier

DT_SET5 = DecisionTreeClassifier()
params = {'max_depth':[1, 5, 10, 50, 100, 500, 100], 'min_samples_split': [5, 10, 100, 500]}


grid = RandomizedSearchCV(DT_SET5 , params, cv = 3, scoring = 'roc_auc', random_state = 0)
grid.fit(X_train_set5,y_train)
print(grid.best_params_)
```

```
{'min_samples_split': 500, 'max_depth': 10}
```

In [103]:

```
train_auc= grid.cv_results_['mean_train_score']
train_auc_std= grid.cv_results_['std_train_score']
cv_auc = grid.cv_results_['mean_test_score']
cv_auc_std= grid.cv_results_['std_test_score']
```

In [104]:

```
train_auc
```

Out[104]:

```
array([ 0.5723174 ,  0.98661843,  0.88595414,  0.97032555,  0.67308429,
        0.71299012,  0.97300043,  0.90575763,  0.99927497,  0.97069097])
```

In [105]:

```
cv_auc
```

Out[105]:

```
array([ 0.56778285,  0.56718671,  0.57105146,  0.50103202,  0.64790594,
        0.65853432,  0.50027661,  0.52610971,  0.54299047,  0.49935239])
```

In [106]:

```
x1=[ 0.5723174 ,  0.98661843,  0.88595414,  0.97032555,  0.67308429,
        0.71299012,  0.97300043,  0.90575763,  0.99927497,  0.97069097]
```

In [107]:

```
x2=[ 0.56778285,  0.56718671,  0.57105146,  0.50103202,  0.64790594,
        0.65853432,  0.50027661,  0.52610971,  0.54299047,  0.49935239]
```

In [108]:

```
z1 = pd.Series([1,5,10,10,50,50,100,100,500,500],index = x1)
```

In [109]:

```
y1 = pd.Series([5,10,100,500,5,10,100,500,5,10], index = x1)
```

In [110]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

trace1 = go.Scatter3d(
    x=x1, y=y1, z=z1,
    name = 'Train',
    marker=dict(
        size=4,
        colorscale='Rainbow',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    )
)

trace2 = go.Scatter3d(
    x=x2, y=y1, z=z1,
    name = 'Test',
    marker=dict(
        size=4,
        colorscale='Rainbow',
    ),
    line=dict(
        color='#b45c1f',
        width=1
    )
)
```

In [111]:

```
data=[trace1,trace2]
```

In [112]:

```
layout = dict(
    width=800,
    height=700,
    autosize=False,
    title='Hyper Parameter Tuning -- BoW Data',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
```

```
            backgroundcolor='rgb(230, 230,230)'
        ),
        camera=dict(
            up=dict(
                x=0,
                y=0,
                z=1
            ),
            eye=dict(
                x=-1.7428,
                y=1.0707,
                z=0.7100,
            )
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    ),
)
```

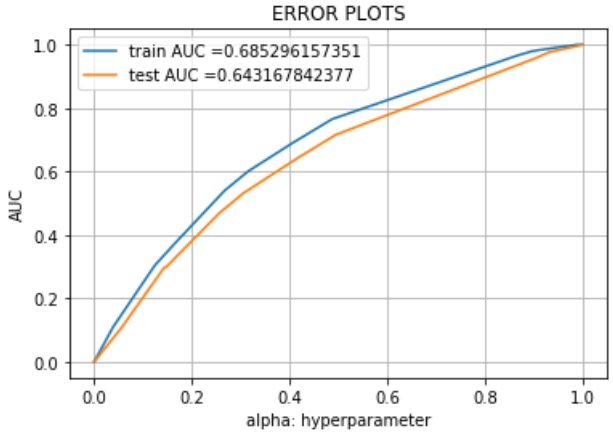
In [118]:

```
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [114]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

DT_SET5 = DecisionTreeClassifier(max_depth = 5, min_samples_split = 10)
DT_SET5.fit(X_train_set5, y_train)
```

```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(DT_SET5, X_train_set5)
y_test_pred = batch_predict(DT_SET5, X_test_set5)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [115]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249930140343 for threshold 0.84
[[ 1703  1647]
 [ 7216 23840]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.249955555556 for threshold 0.896
[[  836   814]
 [ 4369 10928]]
```

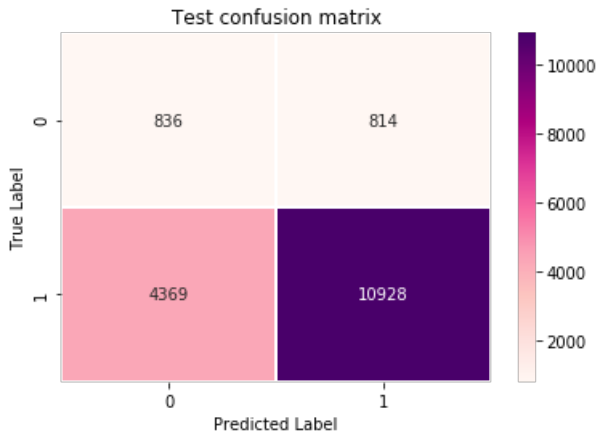
In [116]:

```
frame_confusion_train = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)),range(2),range(2))
frame_confusion_test = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)), range(2),range(2))
```

```
the maximum value of tpr*(1-fpr) 0.249930140343 for threshold 0.84
the maximum value of tpr*(1-fpr) 0.249955555556 for threshold 0.896
```

In [117]:

```
sns.heatmap(frame_confusion_test, annot = True, fmt="d", cmap="RdPu", linewidths=.5)
plt.title("Test confusion matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



Summary

Pretty Table

In [119]:

```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer","max_depth","mim_sample_split","AUC"]
x.add_row(["BOW",10,500,0.6595])
x.add_row(["TFIDF",10,500, 0.6566])
x.add_row(["W2V",5,10,0.6498])
x.add_row(["TFIDF_w2v",5,10, 0.6499])
x.add_row(["best5k",5,10, 0.6431])
print(x)
```

```
+------------+-----------+------------------+--------+
| Vectorizer | max_depth | mim_sample_split |  AUC   |
+------------+-----------+------------------+--------+
|    BOW     |     10    |        500       | 0.6595 |
|   TFIDF    |     10    |        500       | 0.6566 |
|    W2V     |     5     |        10        | 0.6498 |
| TFIDF_w2v  |     5     |        10        | 0.6499 |
|   best5k   |     5     |        10        | 0.6431 |
+------------+-----------+------------------+--------+
```