

```
1 %matplotlib inline
2 import warnings
3 warnings.filterwarnings("ignore")
4
5 import sqlite3
6 import pandas as pd
7 import numpy as np
8 import nltk
9 import string
10 import matplotlib.pyplot as plt
11 import seaborn as sns
12 from sklearn.feature_extraction.text import TfidfTransformer
13 from sklearn.feature_extraction.text import TfidfVectorizer
14
15 from sklearn.feature_extraction.text import CountVectorizer
16 from sklearn.metrics import confusion_matrix
17 from sklearn import metrics
18 from sklearn.metrics import roc_curve, auc
19 from nltk.stem.porter import PorterStemmer
20
21 import re
22 # Tutorial about Python regular expressions: https://pymotw.com/2/re/
23 import string
24 from nltk.corpus import stopwords
25 from nltk.stem import PorterStemmer
26 from nltk.stem.wordnet import WordNetLemmatizer
27
28 from gensim.models import Word2Vec
29 from gensim.models import KeyedVectors
30 import pickle
31
32 from tqdm import tqdm
33 import os
34
35 # from plotly import plotly
36 # import plotly.offline as offline
37 # import plotly.graph_objs as go
38 # offline.init_notebook_mode()
39 from collections import Counter
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
1 #getting the file from google drive (test data)
2 import gdown
3
4 url = 'https://drive.google.com/uc?id=1bDLwb_Vq7q2W9S89JB96PgmZG3LsLns9'
5 output = 'train.csv'
6 # https://drive.google.com/file/d/1bDLwb_Vq7q2W9S89JB96PgmZG3LsLns9/view?usp=sharing
7 gdown.download(url, output, quiet=False)
```

↳ Downloading...
From: https://drive.google.com/uc?id=1bDLwb_Vq7q2W9S89JB96PgmZG3LsLns9
To: /content/train.csv
201MB [00:02, 77.1MB/s]
'train.csv'

```
1 #getting the data from google drive (resources data)resources data
2 import gdown
3 url = 'https://drive.google.com/uc?id=14OVXWu_SJU-lJD-jkMOClD14EZ21lYYe'
4 output = 'resources.csv'
5 gdown.download(url, output, quiet=False)
6
```

↳ Downloading...
From: https://drive.google.com/uc?id=14OVXWu_SJU-lJD-jkMOClD14EZ21lYYe
To: /content/resources.csv
127MB [00:00, 73.4MB/s]
'resources.csv'

```
1 dft = pd.read_csv('train.csv',nrows=50000)
2 dfr = pd.read_csv('resources.csv')
```

```
1 print("Number of data points in train data", dft.shape)
2 print('^^'*50)
3 print("The attributes of data :", dft.columns.values)
4 print('^^'*50)
5 print(dfr.shape)
6 print(dfr.columns.values)
```

↳

Number of data points in train data (50000, 17)

```
1 # sort the datapoints by date and time column
2 # list comprehension python :# https://stackoverflow.com/a/2582163/4084039
3 cols = ['Date' if x=='project_submitted_datetime' else x for x in list(dft.columns)]
4 #sort dataframe based on time uisng pandas to_datetime function : https://stackoverflow.com/a/49702492/4084039
5 dft['Date'] = pd.to_datetime(dft['project_submitted_datetime'])
6 dft.drop('project_submitted_datetime', axis=1, inplace=True)# we drop the col
7 dft.sort_values(by=['Date'], inplace=True)# sort the values y date
8 dft.head(2)
```

↳

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_grade_category	project
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	Grades PreK-2
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	Grades 3-5

▼ 1.1 Text preprocessing

```
1 # merge two column text dataframe:
2 dft["essay"] = dft["project_essay_1"].map(str) + \
3               dft["project_essay_2"].map(str) + \
4               dft["project_essay_3"].map(str) + \
5               dft["project_essay_4"].map(str)
```

```
1 # https://stackoverflow.com/a/47091490/4084039
2 import re
3
4 def decontracted(phrase):
5     # specific
6     phrase = re.sub(r"won't", "will not", phrase)
7     phrase = re.sub(r"can't", "can not", phrase)
8
```

```

9      # general
10     phrase = re.sub(r"\n\t", " not", phrase)
11     phrase = re.sub(r"\re", " are", phrase)
12     phrase = re.sub(r"\s", " is", phrase)
13     phrase = re.sub(r"\d", " would", phrase)
14     phrase = re.sub(r"\ll", " will", phrase)
15     phrase = re.sub(r"\t", " not", phrase)
16     phrase = re.sub(r"\ve", " have", phrase)
17     phrase = re.sub(r"\m", " am", phrase)
18     return phrase

1 # https://gist.github.com/sebleier/554280
2 # we are removing the words from the stop words list so as to get btter prediction : that is , no , not ,etc .
3 stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", \
4             "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
5             'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', \
6             'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
7             'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
8             'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
9             'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', \
10            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', \
11            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', \
12            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
13            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
14            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', \
15            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', \
16            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
17            'won', "won't", 'wouldn', "wouldn't"]

```

▼ Preprocessing of the ****project_subject_categories****

```

1 categories = list(dft['project_subject_categories'].values)
2 # remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039
3 # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
4 # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
5 # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
6 cat_list = []
7 for i in categories:
8     temp = ""
9     for j in i.split(','):
10         if 'The' in j.split():
11             j=j.replace('The','')
12             j = j.replace(' ','')
13             temp+=j.strip()+" "
14         temp = temp.replace('&','_')
15     cat_list.append(temp.strip())
16

```

```

17 dft['clean_categories'] = cat_list
18 dft.drop(['project_subject_categories'], axis=1, inplace=True)
19
20 from collections import Counter
21 my_counter = Counter()
22 for word in dft['clean_categories'].values:
23     my_counter.update(word.split())
24
25 cat_dict = dict(my_counter)
26 project_subject_categories_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
27 print(project_subject_categories_dict)

```

```

{ 'Warmth': 643, 'Care_Hunger': 643, 'History_Civics': 2689, 'Music_Arts': 4699, 'AppliedLearning': 5569, 'SpecialNeeds'

```

▼ preprocessing of the ****project_subject_subcategories****

```

1 sub_catogories = list(dft['project_subject_subcategories'].values)
2 # # remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039
3 # # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
4 # # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
5 # # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
6
7 sub_cat_list = []
8 for i in sub_catogories:
9     temp = ""
10     for j in i.split(','):
11         if 'The' in j.split():
12             j=j.replace('The','')
13             j = j.replace(' ','')
14             temp +=j.strip()+" "
15         temp = temp.replace('&','_')
16     sub_cat_list.append(temp.strip())
17
18 dft['clean_subcategories'] = sub_cat_list
19 dft.drop(['project_subject_subcategories'], axis=1, inplace=True)
20
21 # count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
22 my_counter = Counter()
23 for word in dft['clean_subcategories'].values:
24     my_counter.update(word.split())
25
26 sub_cat_dict = dict(my_counter)
27 project_subject_subcategories_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
28 project_subject_subcategories_dict

```

```

{
  'AppliedSciences': 4901,
  'Care_Hunger': 643,
  'CharacterEducation': 958,
  'Civics_Government': 380,
  'College_CareerPrep': 1168,
  'CommunityService': 214,
  'ESL': 1999,
  'EarlyDevelopment': 1937,
  'Economics': 127,
  'EnvironmentalScience': 2533,
  'Extracurricular': 373,
  'FinancialLiteracy': 253,
  'ForeignLanguages': 388,
  'Gym_Fitness': 2068,
  'Health_LifeScience': 1876,
  'Health_Wellness': 4732,
  'History_Geography': 1433,
  'Literacy': 15611,
  'Literature_Writing': 10127,
  'Mathematics': 12832,
  'Music': 1432,
  'NutritionEducation': 617,
  'Other': 1128,
  'ParentInvolvement': 302,
  'PerformingArts': 910,
  'SocialSciences': 864,
  'SpecialNeeds': 6233,
  'TeamSports': 995,
  'VisualArts': 2865,
  'Warmth': 643}

```

▼ preprocessing of the ****project_grade_category****

```

1 grade_cat_list = []
2 for grade in dft['project_grade_category'].values:
3     grade = grade.replace("-", "_").lower()
4     grade = grade.replace(" ", "_").lower()
5     grade_cat_list.append(grade)
6
7

```

```

8 dft['clean_grade'] = grade_cat_list
9 dft.drop(['project_grade_category'], axis=1, inplace=True)
10
11 my_counter = Counter()
12 for word in dft['clean_grade'].values:
13     my_counter.update(word.split())
14 project_grade_category_dict= dict(my_counter)
15 sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda kv: kv[1]))
16 sorted_project_grade_category_dict

```

```

{ 'grades_3_5': 16968,
  'grades_6_8': 7750,
  'grades_9_12': 4966,
  'grades_prek_2': 20316}

```

▼ Preparing data matrix for the models for Model

```

1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(dft,dft['project_is_approved'],stratify= dft['project_is_approved'],test_size
3 X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify= y_train,test_size = 0.33)

1 print("the shape of the Y_train data ", y_train.value_counts())
2 print("&"*25)
3 print("the shape of the Y_test data ",y_test.value_counts())
4 print("&"*25)
5 print("the shape of the Y_cv data ",y_cv.value_counts())
6 print("&"*25)
7 print("the above representaion shows a huge imbalance in data ")

```

```

{

```

```

the shape of the Y_train data 1    18982
0    3463
Name: project_is_approved, dtype: int64
#####
the shape of the Y_test data 1    13954
0    2546
Name: project_is_approved, dtype: int64
#####
the shape of the Y_cv data 1    9350
0    1705
Name: project_is_approved, dtype: int64
#####
the above representaion shows a huge imbalance in data

```

```

1 # # Dropping the target variable coloums
2 X_train.drop(["project_is_approved"], axis = 1, inplace = True)
3 #x_test =
4 X_test.drop(["project_is_approved"], axis = 1, inplace = True)
5 #x_cv =
6 X_cv.drop(["project_is_approved"], axis = 1, inplace = True)
7

```

▼ Pre-processing the Text Features ,

here we are independtenly doing the preprocessing as i observed that if we pre-processes and then divide the data , it is casuing in lower AUC score

```

1 #Proprocessing for essay
2 # Combining all the above students
3 from tqdm import tqdm
4 preprocessed_essays_train = []
5 # tqdm is for printing the status bar
6 for sentance in tqdm(X_train['essay'].values):
7     sent = decontracted(sentance)
8     sent = sent.replace('\r', ' ')
9     sent = sent.replace('\n', ' ')
10    sent = sent.replace('\n', ' ')
11    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
12 # https://gist.github.com/sebleier/554280
13    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
14    preprocessed_essays_train.append(sent.lower().strip())
15

```


100%|██████████| 22445/22445 [00:13<00:00, 1712.52it/s]

```

1 #Proprocessing for essay
2 # Combining all the above students
3 from tqdm import tqdm
4 preprocessed_essays_test = []
5 # tqdm is for printing the status bar
6 for sentance in tqdm(X_test['essay'].values):
7     sent = decontracted(sentance)
8     sent = sent.replace('\r', ' ')
9     sent = sent.replace('\n', ' ')
10    sent = sent.replace('\n', ' ')
11    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
12 # https://gist.github.com/sebleier/554280
13    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
14    preprocessed_essays_test.append(sent.lower().strip())

```

100%|██████████| 16500/16500 [00:10<00:00, 1540.34it/s]

```

1 #Proprocessing for essay
2 # Combining all the above students
3 from tqdm import tqdm
4 preprocessed_essays_cv = []
5 # tqdm is for printing the status bar
6 for sentance in tqdm(X_cv['essay'].values):
7     sent = decontracted(sentance)
8     sent = sent.replace('\r', ' ')
9     sent = sent.replace('\n', ' ')
10    sent = sent.replace('\n', ' ')
11    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
12 # https://gist.github.com/sebleier/554280
13    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
14    preprocessed_essays_cv.append(sent.lower().strip())
15

```

100%|██████████| 11055/11055 [00:06<00:00, 1712.96it/s]

```

1 #Proprocessing for essay
2 # Combining all the above students
3 from tqdm import tqdm
4 preprocessed_titles_train = []
5 # tqdm is for printing the status bar
6 for sentance in tqdm(X_train['project_title'].values):
7     sent = decontracted(sentance)

```

```

8     sent = sent.replace('\\r', ' ')
9     sent = sent.replace('\\n', ' ')
10    sent = sent.replace('\\n', ' ')
11    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
12    # https://gist.github.com/sebleier/554280
13    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
14    preprocessed_titles_train.append(sent.lower().strip())
15

```

100%|██████████| 22445/22445 [00:00<00:00, 39737.28it/s]

```

1 #Proprocessing for essay
2 # Combining all the above students
3 from tqdm import tqdm
4 preprocessed_titles_cv = []
5 # tqdm is for printing the status bar
6 for sentence in tqdm(X_cv['project_title'].values):
7     sent = decontracted(sentence)
8     sent = sent.replace('\\r', ' ')
9     sent = sent.replace('\\n', ' ')
10    sent = sent.replace('\\n', ' ')
11    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
12    # https://gist.github.com/sebleier/554280
13    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
14    preprocessed_titles_cv.append(sent.lower().strip())

```

100%|██████████| 11055/11055 [00:00<00:00, 39204.44it/s]

```

1 #Proprocessing for essay
2 # Combining all the above students
3 from tqdm import tqdm
4 preprocessed_titles_test = []
5 # tqdm is for printing the status bar
6 for sentence in tqdm(X_test['project_title'].values):
7     sent = decontracted(sentence)
8     sent = sent.replace('\\r', ' ')
9     sent = sent.replace('\\n', ' ')
10    sent = sent.replace('\\n', ' ')
11    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
12    # https://gist.github.com/sebleier/554280
13    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
14    preprocessed_titles_test.append(sent.lower().strip())

```

100%|██████████| 16500/16500 [00:00<00:00, 39630.83it/s]

▼ Vectorizing the Numerical and categorical data

```

1 from collections import Counter
2 my_counter = Counter()
3 for word in X_train['clean_categories'].values:
4     my_counter.update(word.split())
5
6 cat_dict = dict(my_counter)
7 project_subject_categories_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
8
9
10
11 from sklearn.feature_extraction.text import CountVectorizer
12 vectorizer1 = CountVectorizer(vocabulary=list(project_subject_categories_dict.keys()), lowercase=False, binary=True)
13 vectorizer1.fit(X_train['clean_categories'].values)
14 X_train_cat = vectorizer1.transform(X_train['clean_categories'].values)
15 X_cv_cat = vectorizer1.transform(X_cv['clean_categories'].values)
16 X_test_cat = vectorizer1.transform(X_test['clean_categories'].values)
17 print(vectorizer1.get_feature_names())

```

☞ ['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Sci'

```

1 print("After vectorizations")
2 print(X_train_cat.shape, y_train.shape)
3 print(X_cv_cat.shape, y_cv.shape)
4 print(X_test_cat.shape, y_test.shape)
5 print("=="*100)

```

☞ After vectorizations

```

(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
=====

```

```

1 my_counter = Counter()
2 for word in X_train['clean_subcategories'].values:
3     my_counter.update(word.split())
4
5 sub_cat_dict = dict(my_counter)
6 project_subject_subcategories_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
7 project_subject_subcategories_dict
8
9 from sklearn.feature_extraction.text import CountVectorizer

```

```

10 vectorizer2 = CountVectorizer(vocabulary=list(project_subject_subcategories_dict.keys()), lowercase=False, binary=True)
11 vectorizer2.fit(X_train['clean_subcategories'].values)
12 # firstly convert fit the train data into the vectoriaer then it learn hte vocablery
13 # we use the fitted CountVectorizer to convert the text to vector
14 X_train_subcat = vectorizer2.transform(X_train['clean_subcategories'].values)
15 X_cv_subcat = vectorizer2.transform(X_cv['clean_subcategories'].values)
16 X_test_subcat = vectorizer2.transform(X_test['clean_subcategories'].values)
17 print(vectorizer2.get_feature_names())
18
19 print("After vectorizations")
20 print(X_train_subcat.shape, y_train.shape)
21 print(X_cv_subcat.shape, y_cv.shape)
22 print(X_test_subcat.shape, y_test.shape)
23 print("=="*100)

```

```

↳ ['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Civics_Government', 'Extracurricular', 'Fo
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
=====

```

```

1 from collections import Counter
2 my_counter = Counter()
3 for word in X_train['school_state'].values:
4     my_counter.update(word.split())# count the words
5 school_state_dict = dict(my_counter)# store in dictionary
6 sorted_school_state_dict = dict(sorted(school_state_dict.items(), key=lambda kv: kv[1]))
7
8 from sklearn.feature_extraction.text import CountVectorizer
9 vectorizer3 = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercase=False, binary=True)
10 vectorizer3.fit(dft['school_state'].values)
11 # firstly convert fit the train data into the vector then it learn the vocablery
12 # we use the fitted CountVectorizer to convert the text to vector
13 X_train_school_state = vectorizer3.transform(X_train['school_state'].values)
14 X_cv_school_state = vectorizer3.transform(X_cv['school_state'].values)
15 X_test_school_state = vectorizer3.transform(X_test['school_state'].values)
16 print(vectorizer3.get_feature_names())
17 print("After vectorizations")
18 print(X_train_school_state .shape, y_train.shape)
19 print(X_cv_school_state .shape, y_cv.shape)
20 print(X_test_school_state .shape, y_test.shape)
21 print("=="*100)

```

```

↳

```

```
['VT', 'WY', 'ND', 'MT', 'NE', 'RI', 'SD', 'NH', 'AK', 'DE', 'WV', 'NM', 'ME', 'DC', 'HI', 'IA', 'ID', 'KS', 'AR', 'CO']
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
=====
```

```
1 my_counter = Counter()
2 for word in X_train['clean_grade'].values:
3     my_counter.update(word.split())
4 project_grade_category_dict= dict(my_counter)
5 sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda kv: kv[1]))
6 sorted_project_grade_category_dict
7
8
9 dft['clean_grade']=dft['clean_grade'].fillna("#")# fill the null values with space
10 vectorizer4 = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()),lowercase=False, binary=True)
11 vectorizer4.fit(dft['clean_grade'].values)
12 # firstly convert fit the train data into the vectoriaer then it learn hte vocablery
13 # we use the fitted CountVectorizer to convert the text to vector
14 X_train_project_grade_category = vectorizer4.transform(X_train['clean_grade'].values)
15 X_cv_project_grade_category = vectorizer4.transform(X_cv['clean_grade'].values)
16 X_test_project_grade_category = vectorizer4.transform(X_test['clean_grade'].values)
17 print(vectorizer4.get_feature_names())
18
19 print("After vectorizations")
20 print(X_train_project_grade_category .shape, y_train.shape)
21 print(X_cv_project_grade_category .shape, y_cv.shape)
22 print(X_test_project_grade_category .shape, y_test.shape)
23 print("=="*100).
```

```
↳ ['grades_9_12', 'grades_6_8', 'grades_3_5', 'grades_prek_2']
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
=====
```

```
1 dft['teacher_prefix']=dft['teacher_prefix'].fillna(" ")# filll the null valueswith space
2 my_counter = Counter()
3 for word in dft['teacher_prefix'].values:
4     my_counter.update(word.split())
5 teacher_cat_dict = dict(my_counter)
```

```

6 sorted_teacher_prefix_dict = dict(sorted(teacher_cat_dict.items(), key=lambda kv: kv[1]))
7
8 from sklearn.feature_extraction.text import CountVectorizer
9 vectorizer5 = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lowercase=False, binary=True)
10 vectorizer5.fit(dft['teacher_prefix'].values.astype('U'))
11 # firstly convert fit the train data into the vectorizer
12 # we use the fitted CountVectorizer to convert the text to vector
13 X_train_teacher_prefix = vectorizer5.transform(X_train['teacher_prefix'].values.astype('U'))
14 X_cv_teacher_prefix = vectorizer5.transform(X_cv['teacher_prefix'].values.astype('U'))
15 X_test_teacher_prefix = vectorizer5.transform(X_test['teacher_prefix'].values.astype('U'))
16 print(vectorizer5.get_feature_names())
17
18 print("After vectorizations")
19 print(X_train_teacher_prefix .shape, y_train.shape)
20 print(X_cv_teacher_prefix .shape, y_cv.shape)
21 print(X_test_teacher_prefix .shape, y_test.shape)
22 print("="*100)

```

```

↳ ['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
=====

```

```

1 # chainging the names
2 X_train_essay=preprocessed_essays_train
3 X_cv_essay=preprocessed_essays_cv
4 X_test_essay=preprocessed_essays_test

1 # Considering only the words which appeared in at least 10 documents(rows or projects).
2 vectorizer6 = CountVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
3 vectorizer6.fit(X_train_essay)# that is learned from trained data
4
5 # we use the fitted CountVectorizer to convert the text to vector
6 X_train_bow = vectorizer6.transform(X_train_essay)
7 X_cv_bow = vectorizer6.transform(X_cv_essay)
8 X_test_bow = vectorizer6.transform(X_test_essay)
9 print("After vectorizations")
10 print(X_train_bow.shape, y_train.shape)
11 print(X_cv_bow.shape, y_cv.shape)
12 print(X_test_bow.shape, y_test.shape)
13 print("="*100)

```

```
↳
```

```
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
=====
```

```
1 X_train_title=preprocessed_titles_train
2 X_cv_title=preprocessed_titles_cv
3 X_test_title=preprocessed_titles_test
```

```
1 #bow featurization title
2 vectorizer7 = CountVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
3 vectorizer7.fit(X_train_title)# that is learned from trained data
4 X_train_bow_title = vectorizer7.transform(X_train_title)
5 X_cv_bow_title= vectorizer7.transform(X_cv_title)
6 X_test_bow_title = vectorizer7.transform(X_test_title)
7 print("After vectorizations")
8 print(X_train_bow_title.shape, y_train.shape)
9 print(X_cv_bow_title.shape, y_cv.shape)
10 print(X_test_bow_title.shape, y_test.shape)
11 print("=="*100)
12
```

```
↳ After vectorizations
(22445, 1609) (22445,)
(11055, 1609) (11055,)
(16500, 1609) (16500,)
=====
```

▼ TFIDF Vectorization of the data metrix

```
1 #for titles
2 from sklearn.feature_extraction.text import TfidfVectorizer
3 # We are considering only the words which appeared in at least 10 documents(rows or projects).
4 vectorizer8 = TfidfVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
5 vectorizer8.fit(X_train_title)# that is learned from trained data
6
7 # we use the fitted CountVectorizer to convert the text to vector
8 X_train_tf_title = vectorizer8.transform(X_train_title)
9 X_cv_tf_title= vectorizer8.transform(X_cv_title)
```

```

10 X_test_tf_title = vectorizer8.transform(X_test_title)
11 print("After vectorizations")
12 print(X_train_tf_title.shape, y_train.shape)
13 print(X_cv_tf_title.shape, y_cv.shape)
14 print(X_test_tf_title.shape, y_test.shape)
15 print("=*100)
16

```

```

↳ After vectorizations
(22445, 1609) (22445,)
(11055, 1609) (11055,)
(16500, 1609) (16500,)
=====

```

```

1 #for essay
2 from sklearn.feature_extraction.text import TfidfVectorizer
3 # We are considering only the words which appeared in at least 10 documents(rows or projects).
4 vectorizer9 = TfidfVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
5 vectorizer9.fit(X_train_essay)# that is learned from trained data
6 # we use the fitted CountVectorizer to convert the text to vector
7 X_train_tf_essay = vectorizer9.transform(X_train_essay)
8 X_cv_tf_essay= vectorizer9.transform(X_cv_essay)
9 X_test_tf_essay = vectorizer9.transform(X_test_essay)
10 print("After vectorizations")
11 print(X_train_tf_essay.shape, y_train.shape)
12 print(X_cv_tf_essay.shape, y_cv.shape)
13 print(X_test_tf_essay.shape, y_test.shape)
14 print("=*100)

```

```

↳ After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
=====


```

▼ AVG Word2Vec Vectorization using the pretrained models

```

1 import gdown
2
3 url = 'https://drive.google.com/uc?id=1MqUasf7jYoPbG35MJ28VQc0jjNp-ZDDp'
4 output = 'glove_vectors'
5 gdown.download(url, output, quiet=False)

```


 Downloading...
 From: <https://drive.google.com/uc?id=1MqUasf7jYoPbG35MJ28VQcOjjNp-ZDDp>
 To: /content/glove_vectors
 128MB [00:02, 52.4MB/s]
 'glove_vectors'

```

1 import pickle
2 with open('glove_vectors', 'rb') as f:
3     model = pickle.load(f)
4     glove_words = set(model.keys())

1 def AVG_W2V(values):
2     # AVG W2V Vectorization.
3
4     train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
5     for sentence in tqdm(values): # for each review/sentence
6         vector = np.zeros(300) # as word vectors are of zero length # we are taking the 300 dimensions very large
7         cnt_words = 0; # num of words with a valid vector in the sentence/review
8         for word in sentence.split(): # for each word in a review/sentence
9             if word in glove_words:
10                 vector += model[word]
11                 cnt_words += 1
12         if cnt_words != 0:
13             vector /= cnt_words
14         train_avg_w2v_vectors.append(vector)
15
16     print(len(train_avg_w2v_vectors))
17     print(len(train_avg_w2v_vectors[0]))
18     return train_avg_w2v_vectors
19

1 train_avg_w2v_vectors=AVG_W2V(preprocessed_essays_train)
2 test_avg_w2v_vectors=AVG_W2V(preprocessed_essays_test)
3 cv_avg_w2v_vectors=AVG_W2V(preprocessed_essays_cv)
4 print("After vectorizations")
5 print(len(train_avg_w2v_vectors), y_train.shape)
6 print("=*100)
  
```



```

100%|██████████| 22445/22445 [00:06<00:00, 3582.45it/s]
  2%|█          | 340/16500 [00:00<00:04, 3395.42it/s]22445
300
100%|██████████| 16500/16500 [00:04<00:00, 3546.40it/s]
  3%|█          | 370/11055 [00:00<00:02, 3697.19it/s]16500
300
100%|██████████| 11055/11055 [00:03<00:00, 3603.87it/s]11055
300
After vectorizations
22445 (22445,)
=====

```

```

1 # AVG W2V for preprocessed_titles
2 train_avg_w2v_vectors_title=AVG_W2V(preprocessed_titles_train)
3 test_avg_w2v_vectors_title=AVG_W2V(preprocessed_titles_test)
4 cv_avg_w2v_vectors_title=AVG_W2V(preprocessed_titles_cv)
5

```

↳

```

100%|██████████| 22445/22445 [00:00<00:00, 70982.27it/s]
 36%|███        | 6005/16500 [00:00<00:00, 60049.20it/s]22445
300
100%|██████████| 16500/16500 [00:00<00:00, 65943.15it/s]
100%|██████████| 11055/11055 [00:00<00:00, 75073.64it/s]16500
300
11055
300

```

▼ 3.4 TFIDF weighted W2V vectorization using the Pretrained Models

```

1 tfidf_model = TfidfVectorizer()
2 tfidf_model.fit(preprocessed_essays_train)
3 # we are converting a dictionary with word as a key, and the idf as a value
4 dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
5 tfidf_words = set(tfidf_model.get_feature_names())

```

```

1 def tf_idf(word_list):

```

```

2 train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
3 for sentence in tqdm(word_list): # for each review/sentence
4     vector = np.zeros(300) # as word vectors are of zero length
5     tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
6     for word in sentence.split(): # for each word in a review/sentence
7         if (word in glove_words) and (word in tfidf_words):
8             #vec = model.wv[word]
9             vec = model[word] # getting the vector for each word
10            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
11            vector += (vec * tf_idf) # calculating tfidf weighted w2v
12            tf_idf_weight += tf_idf
13        if tf_idf_weight != 0:
14            vector /= tf_idf_weight
15    train_title_tfidf_w2v_vectors.append(vector)
16    print(len(train_title_tfidf_w2v_vectors))
17    print(len(train_title_tfidf_w2v_vectors[0]))
18    return train_title_tfidf_w2v_vectors
19

```

```

1 train_tfidf_w2v_vectors=tf_idf(preprocessed_essays_train)
2 test_tfidf_w2v_vectors=tf_idf(preprocessed_essays_test)
3 cv_tfidf_w2v_vectors=tf_idf(preprocessed_essays_cv).

```

```

100%|██████████| 22445/22445 [00:38<00:00, 583.23it/s]
  0%|          | 60/16500 [00:00<00:27, 594.22it/s]22445
300
100%|██████████| 16500/16500 [00:28<00:00, 586.50it/s]
  1%|          | 60/11055 [00:00<00:18, 595.49it/s]16500
300
100%|██████████| 11055/11055 [00:18<00:00, 584.14it/s]11055
300

```

```

1 train_title_tfidf_w2v_vectors=tf_idf(preprocessed_titles_train)
2 test_title_tfidf_w2v_vectors=tf_idf(preprocessed_titles_test)
3 cv_title_tfidf_w2v_vectors=tf_idf(preprocessed_titles_cv).

```



```

100%|██████████| 22445/22445 [00:00<00:00, 38113.87it/s]
 31%|███| 5129/16500 [00:00<00:00, 26755.79it/s]22445
300
100%|██████████| 16500/16500 [00:00<00:00, 27463.81it/s]
 36%|███| 3949/11055 [00:00<00:00, 39486.74it/s]16500
300
100%|██████████| 11055/11055 [00:00<00:00, 37477.29it/s]
11055
300

```

▼ 4.1 Vectorization of the Numerical features

```

1 price_data = dfr.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
2 dft = pd.merge(dft, price_data, on='id', how='left')
3 X_train = pd.merge(X_train, price_data, on = "id", how = "left")
4 X_test = pd.merge(X_test, price_data, on = "id", how = "left")
5 X_cv = pd.merge(X_cv, price_data, on = "id", how = "left")

```

```

1 from sklearn.preprocessing import StandardScaler
2 # https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
3
4 scalar = StandardScaler()
5 scalar.fit(X_train['price'].values.reshape(-1,1))
6
7 train_price_standar = scalar.transform(X_train['price'].values.reshape(-1, 1))
8 test_price_standar = scalar.transform(X_test['price'].values.reshape(-1, 1))
9 cv_price_standar = scalar.transform(X_cv['price'].values.reshape(-1, 1))

```

```

1 scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
2 train_prev_proj_standar = scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
3 test_prev_proj_standar = scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
4 cv_prev_proj_standar = scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))

```

```

1 scalar.fit(X_train['quantity'].values.reshape(-1,1))
2 train_qnty_standar = scalar.transform(X_train['quantity'].values.reshape(-1, 1))
3 cv_qnty_standar = scalar.transform(X_cv['quantity'].values.reshape(-1, 1))
4 test_qnty_standar = scalar.transform(X_test['quantity'].values.reshape(-1, 1))

```

▼ **Merging of all the Data matrix for different sets of operations **

1. categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
2. Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
3. Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
4. Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

```

1 from scipy.sparse import hstack
2 # with the same hstack function we are concatenating a sparse matrix and a dense matrix :)
3 X_set1_train = hstack((X_train_bow_title,X_train_bow,X_train_teacher_prefix,X_train_cat,X_train_subcat ,X_train_project_grade_cat
4                       X_train_school_state,train_qnty_standar,train_price_standar,train_prev_proj_standar))
5 # printing the shape of X_set1_train data matrix
6 print(" printing the shape of X_set1_train data matrix",X_set1_train.shape, y_train.shape)
7
8 X_set1_cv = hstack((X_cv_bow_title,X_cv_bow,X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,X_cv_project_grade_category,X_cv_school_state
9                   cv_qnty_standar,cv_price_standar,cv_prev_proj_standar))
10 # printing the shape of X_set1_cv data matrix
11 print("printing the shape of X_set1_cv data matrix",X_set1_cv.shape, y_cv.shape)
12
13 X_set1_test = hstack((X_test_bow_title,X_test_bow,X_test_teacher_prefix,X_test_cat,X_test_subcat,X_test_project_grade_category,X_t
14                     test_qnty_standar,test_price_standar,test_prev_proj_standar))
15
16 # printing the shape of X_set1_test data matrix
17 print("printing the shape of X_set1_test data matrix ",X_set1_test.shape, y_test.shape)
18

```

```

↳ printing the shape of X_set1_train data matrix (22445, 6711) (22445,)
   printing the shape of X_set1_cv data matrix (11055, 6711) (11055,)
   printing the shape of X_set1_test data matrix (16500, 6711) (16500,)

```

```

1 X_set2_train = hstack((X_train_tf_essay,X_train_tf_title,X_train_teacher_prefix,X_train_cat,X_train_subcat,X_train_project_grade_cat
2                       train_qnty_standar,train_price_standar,train_prev_proj_standar))
3
4
5 print("printing the shape of X_set2_train data matrix",X_set2_train.shape, y_train.shape)
6
7 print(""*50)
8 X_set2_cv = hstack((X_cv_tf_essay,X_cv_tf_title,X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,X_cv_project_grade_category,X_cv_school_s
9                   cv_qnty_standar,cv_price_standar,cv_prev_proj_standar))
10
11 print("printing the shape of X_set2_cv data matrix",X_set2_cv.shape, y_cv.shape)
12
13 print(""*50)
14 X_set2_test = hstack((X_test_tf_essay,X_test_tf_title,X_test_teacher_prefix,X_test_cat,X_test_subcat, X_test_project_grade_catego

```

```

15         test_qnty_standar,test_price_standar,test_prev_proj_standar))
16
17 print("printing the shape of X_set2_test data metrix",X_set2_test.shape, y_test.shape)
18
19

```

☞ printing the shape of X_set2_train data metrix (22445, 6711) (22445,)

 printing the shape of X_set2_cv data metrix (11055, 6711) (11055,)

 printing the shape of X_set2_test data metrix (16500, 6711) (16500,)

```

1 X_set3_train = hstack((train_avg_w2v_vectors,train_avg_w2v_vectors_title,train_prev_proj_standar,train_price_standar,train_qnty_s
2                       X_train_teacher_prefix,X_train_cat,X_train_subcat,
3                       X_train_project_grade_category,X_train_school_state))
4
5
6 print("printing the shape of X_set3_train data metrix",X_set3_train.shape, y_train.shape)
7 print("*****50)
8
9 X_set3_cv = hstack((cv_avg_w2v_vectors,cv_avg_w2v_vectors_title,cv_prev_proj_standar,cv_price_standar,cv_qnty_standar,
10                    X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
11                    X_cv_project_grade_category,X_cv_school_state))
12
13
14 print("printing the shape of X_set3_cv data metrix",X_set3_cv.shape, y_cv.shape)
15 print("*****50)
16
17 X_set3_test = hstack((test_avg_w2v_vectors,test_avg_w2v_vectors_title,test_prev_proj_standar,test_price_standar,test_qnty_standar
18                      X_test_teacher_prefix,X_test_cat,X_test_subcat,
19                      X_test_project_grade_category,X_test_school_state))
20
21
22 print("printing the shape of X_set3_test data metrix",X_set3_test.shape, y_test.shape)
23
24

```

☞ printing the shape of X_set3_train data metrix (22445, 702) (22445,)

 printing the shape of X_set3_cv data metrix (11055, 702) (11055,)

 printing the shape of X_set3_test data metrix (16500, 702) (16500,)

```

1 X_set4_train = hstack((train_tfidf_w2v_vectors,train_title_tfidf_w2v_vectors,train_prev_proj_standar,train_price_standar,train_qn
2                       X_train_teacher_prefix,X_train_cat,X_train_subcat,

```

```

3         X_train_project_grade_category,X_train_school_state))
4
5
6 print("printing the shape of X_set4_train data metrix",X_set4_train.shape, y_train.shape)
7 print("*****50)
8
9 X_set4_cv = hstack((cv_tfidf_w2v_vectors,cv_title_tfidf_w2v_vectors,cv_prev_proj_standar,cv_price_standar,cv_qnty_standar,
10                    X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
11                    X_cv_project_grade_category,X_cv_school_state))
12
13
14 print("printing the shape of X_set4_CV data metrix",X_set4_cv.shape, y_cv.shape)
15
16 print("*****50)
17 X_set4_test = hstack((test_title_tfidf_w2v_vectors,test_tfidf_w2v_vectors,test_prev_proj_standar,test_price_standar,test_qnty_star
18                    X_test_teacher_prefix,X_test_cat,X_test_subcat,
19                    X_test_project_grade_category,X_test_school_state))
20
21
22 print("printing the shape of X_set4_test data metrix",X_set4_test.shape, y_test.shape)

```

```

↳ printing the shape of X_set4_train data metrix (22445, 702) (22445,)
*****
printing the shape of X_set4_CV data metrix (11055, 702) (11055,)
*****
printing the shape of X_set4_test data metrix (16500, 702) (16500,)

```

▼ Applying the SGDClassifier on SET:1

```

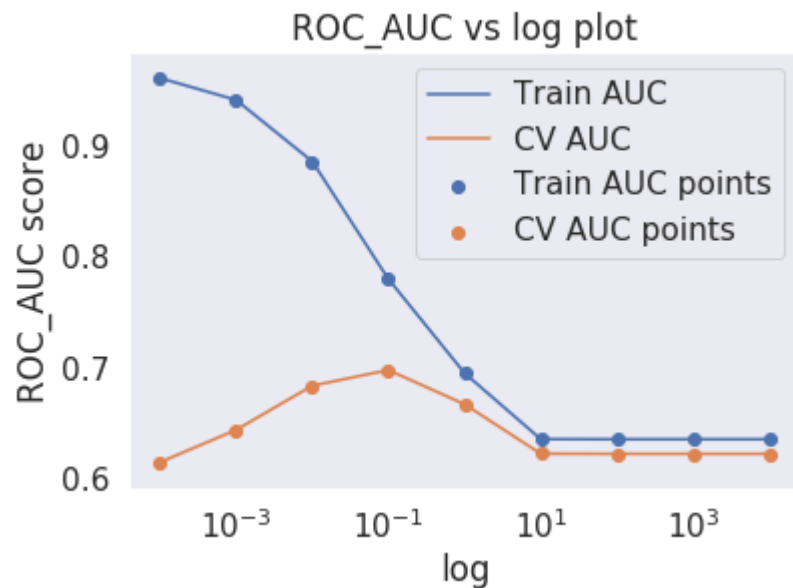
1 ## By using "l2" Regulrizer
2 from sklearn.metrics import roc_auc_score
3 import matplotlib.pyplot as plt
4 from sklearn.model_selection import GridSearchCV
5 from sklearn.linear_model import SGDClassifier
6
7 # hyperparameter tuning with l2 reg
8 parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
9 sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
10 classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
11 classifier.fit(X_set1_train, y_train)
12
13 train_auc = classifier.cv_results_['mean_train_score']
14 cv_auc= classifier.cv_results_['mean_test_score']
15
16 plt.plot(parameters['alpha'], train_auc, label='Train AUC')
17 plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
18 plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')

```

```

19 plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
20 plt.legend()
21 plt.xlabel("log")
22 plt.xscale('log')
23 plt.ylabel("ROC_AUC score")
24 plt.title("ROC_AUC vs log plot")
25 plt.grid()
26 plt.show()
27

```



```

1 #By using "l1" Regularization
2 import warnings
3 warnings.filterwarnings("ignore")
4
5 parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
6 sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
7 classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
8 classifier.fit(X_set1_train, y_train)
9 train_auc = classifier.cv_results_['mean_train_score']
10 cv_auc= classifier.cv_results_['mean_test_score']
11
12 plt.plot(parameters['alpha'], train_auc, label='Train AUC')
13 plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
14
15 plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
16 plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
17 plt.legend()
18 plt.xlabel("Alpha")

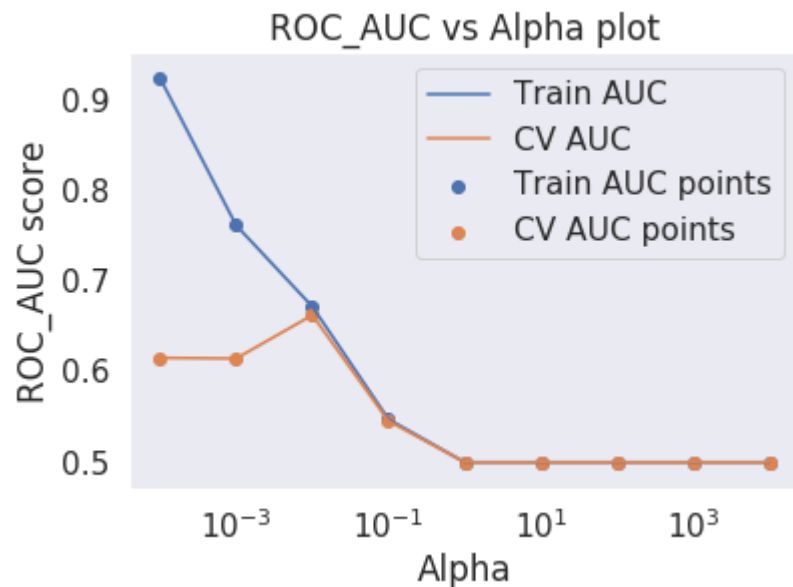
```



```

19 plt.xscale('log').
20 plt.ylabel("ROC_AUC score")
21 plt.title("ROC_AUC vs Alpha plot")
22 plt.grid()
23 plt.show()

```



```

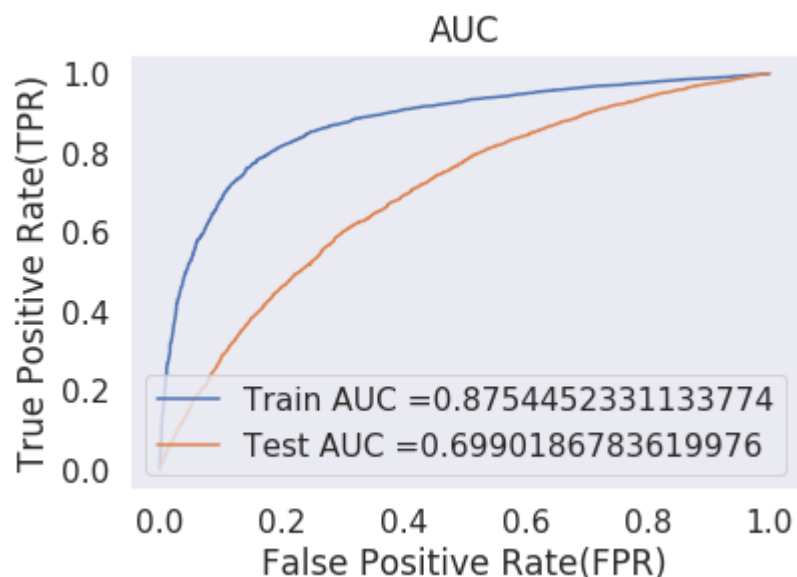
1 # using the L2 regularization
2 from sklearn.calibration import CalibratedClassifierCV
3 from sklearn.metrics import roc_curve, auc
4 Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced', alpha = 10**-2)
5 Classifier_bow.fit(X_set1_train, y_train)
6
7 clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3, method='isotonic')
8 clfcalibrated.fit(X_set1_train, y_train)
9
10 y_train_pred = clfcalibrated.predict_proba(X_set1_train)[: , 1]
11 y_test_pred1 = clfcalibrated.predict_proba(X_set1_test)[: , 1]
12 train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
13 test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred1)
14
15 plt.plot(train_fpr, train_tpr, label="Train AUC =" + str(auc(train_fpr, train_tpr)))
16 plt.plot(test_fpr, test_tpr, label="Test AUC =" + str(auc(test_fpr, test_tpr)))
17 plt.legend()
18 plt.ylabel("True Positive Rate(TPR)")
19 plt.xlabel("False Positive Rate(FPR)")
20 plt.title("AUC")
21 plt.grid()
22 plt.show()

```

```

23 # print(y_train_pred.shape)
24 # y_test_pred.shape

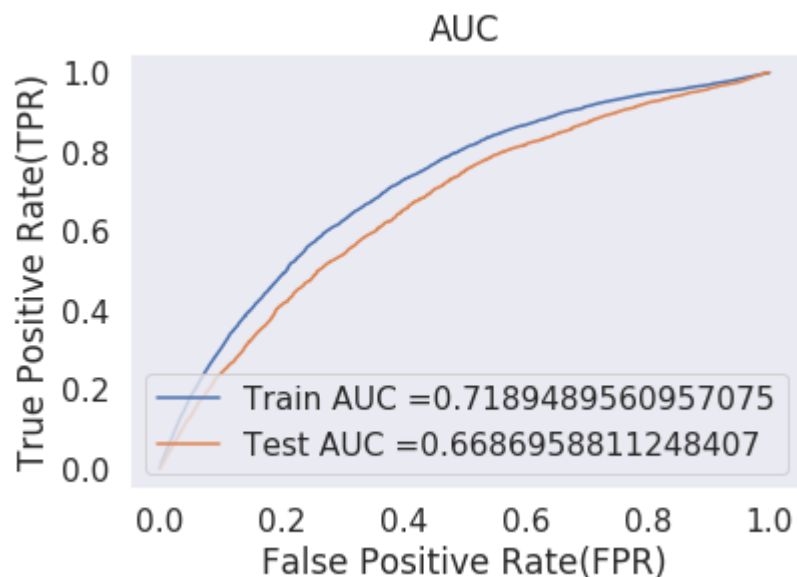
```



```

1 # using the L1 regularization
2 from sklearn.calibration import CalibratedClassifierCV
3 from sklearn.metrics import roc_curve, auc
4 Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l1', alpha = 10**-3)
5 Classifier_bow.fit(X_set1_train, y_train)
6
7 clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3, method='isotonic')
8 clfcalibrated.fit(X_set1_train, y_train)
9
10 y_train_pred = clfcalibrated.predict_proba(X_set1_train)[: , 1]
11 y_test_pred = clfcalibrated.predict_proba(X_set1_test)[: , 1]
12 train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
13 test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
14
15 plt.plot(train_fpr, train_tpr, label="Train AUC =" + str(auc(train_fpr, train_tpr)))
16 plt.plot(test_fpr, test_tpr, label="Test AUC =" + str(auc(test_fpr, test_tpr)))
17 plt.legend()
18 plt.ylabel("True Positive Rate(TPR)")
19 plt.xlabel("False Positive Rate(FPR)")
20 plt.title("AUC")
21 plt.grid()
22 plt.show()
23 # print(y_train_pred.shape)
24 # y_test_pred.shape

```



```

1 def predict(proba, threshold, fpr, tpr):
2
3     t = threshold[np.argmax(fpr*(1-tpr))]
4     print("the maximum value of tpr*(1-fpr)", np.round(max(tpr*(1-fpr)),2) , "for threshold", np.round(t,2))
5     predictions = []
6     for i in proba:
7         if i>=t:
8             predictions.append(1)
9         else:
10            predictions.append(0)
11     return predictions
12

```

```

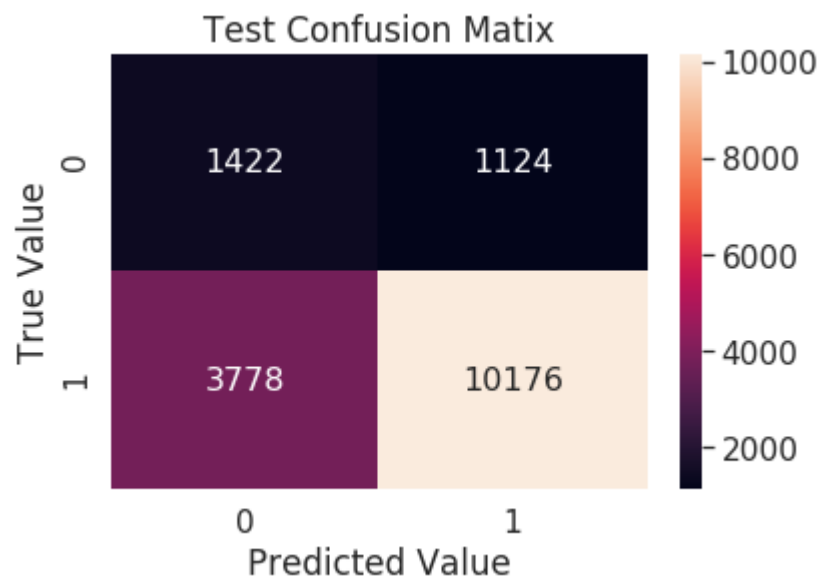
1 import seaborn as sea
2 test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred1, te_thresholds,test_fpr,test_fpr)),range(2),rang
3 sea.set(font_scale=1.4)
4 sea.heatmap(test_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
5 plt.xlabel("Predicted Value")
6 plt.ylabel("True Value")
7 plt.title("Test Confusion Matix")

```



the maximum value of $tpr \cdot (1 - fpr)$ 0.25 for threshold 0.82

Text(0.5, 1.0, 'Test Confusion Matix')



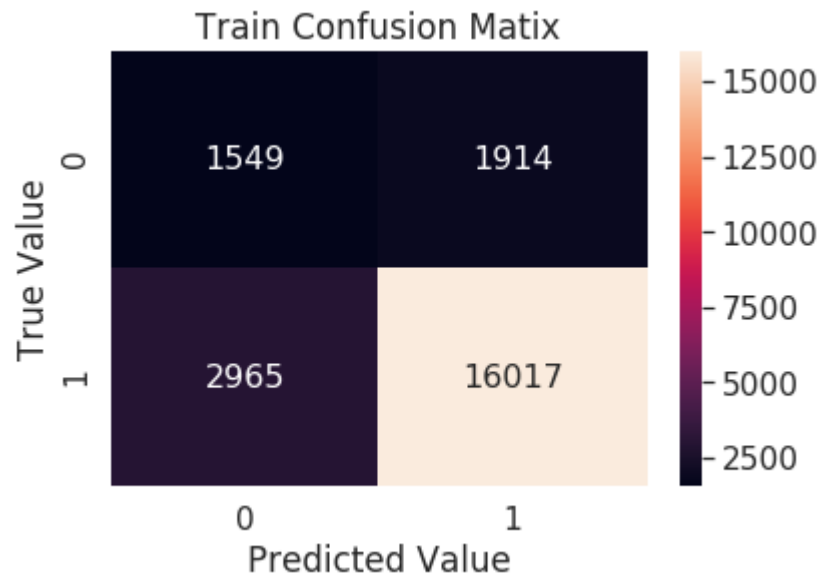
```

1 import seaborn as sea
2 train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, te_thresholds, train_fpr, train_fpr)), range(2)).
3 sea.set(font_scale=1.4)
4 sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
5 plt.xlabel("Predicted Value")
6 plt.ylabel("True Value")
7 plt.title("Train Confusion Matix")

```



the maximum value of $\text{tpr} \times (1 - \text{fpr})$ 0.25 for threshold 0.8
 Text(0.5, 1.0, 'Train Confusion Matix')



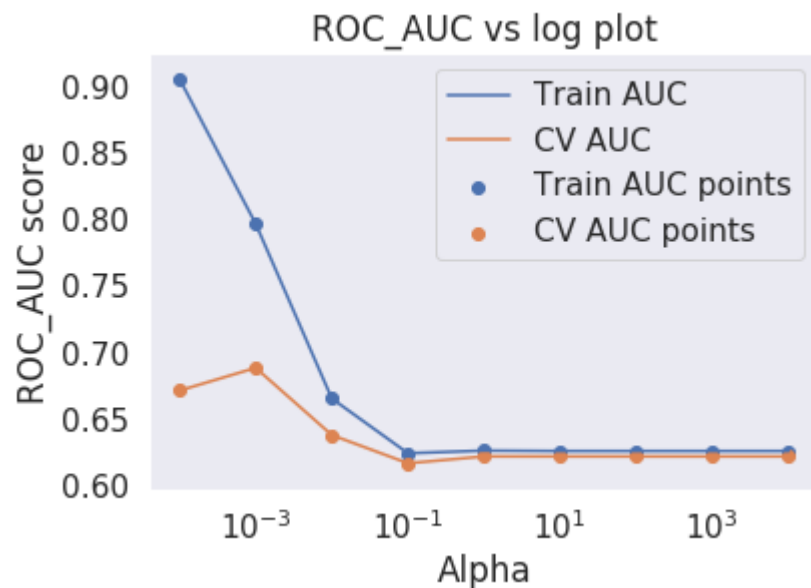
▼ Applying the SGDClassifier on SET:2

```

1 #BY USING L2 REGULARISER
2 # hyperparameter tuning with l2 reg
3 parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
4 sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
5 classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
6 classifier.fit(X_set2_train, y_train)
7
8 train_auc = classifier.cv_results_['mean_train_score']
9 cv_auc= classifier.cv_results_['mean_test_score']
10
11 plt.plot(parameters['alpha'], train_auc, label='Train AUC')
12 plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
13 plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
14 plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
15
16 plt.legend()
17 plt.xscale('log')
18 plt.xlabel("Alpha")
19 plt.ylabel("ROC_AUC score")
20 plt.title("ROC_AUC vs log plot")
21 plt.grid()

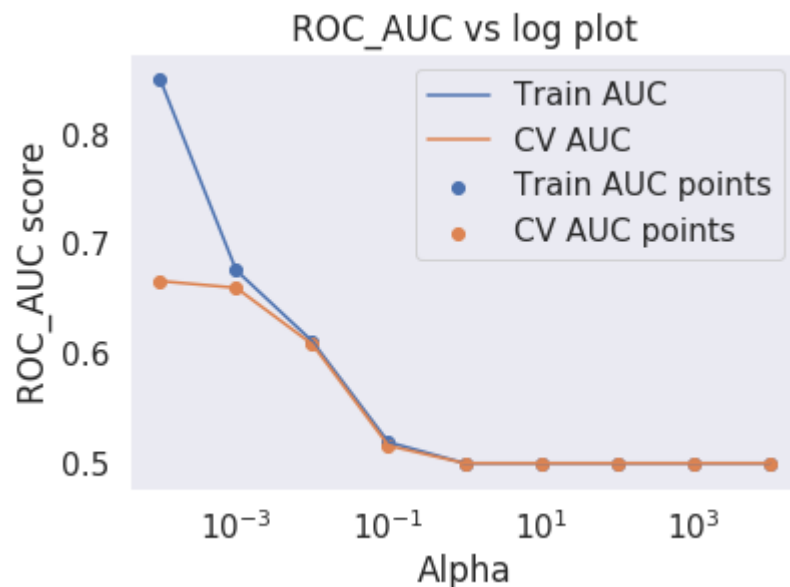
```

```
22 plt.show()
```



```
1 #BY USING "L1" REGULARISER
2 # hyperparameter tuning with 12 reg reduce the alpha values in list
3 parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
4 sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
5 classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
6 classifier.fit(X_set2_train, y_train)
7 train_auc = classifier.cv_results_['mean_train_score']
8 cv_auc= classifier.cv_results_['mean_test_score']
9 plt.plot(parameters['alpha'], train_auc, label='Train AUC')
10 plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
11 plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
12 plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
13 plt.legend()
14 plt.xscale('log')
15 plt.xlabel("Alpha")
16 plt.ylabel("ROC_AUC score")
17 plt.title("ROC_AUC vs log plot")
18 plt.grid()
19 plt.show()
20
```

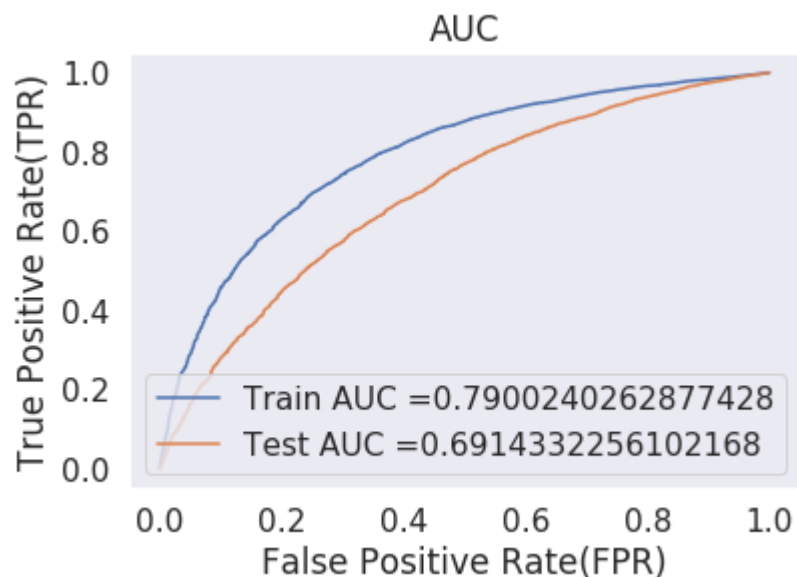




```

1 # using the L2 regularization
2 from sklearn.calibration import CalibratedClassifierCV
3 from sklearn.metrics import roc_curve, auc
4 Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced', alpha = 10**-3)
5 Classifier_bow.fit(X_set2_train, y_train)
6
7 clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3, method='isotonic')
8 clfcalibrated.fit(X_set2_train, y_train)
9
10 y_train_pred2 = clfcalibrated.predict_proba(X_set2_train)[:, 1]
11 y_test_pred2 = clfcalibrated.predict_proba(X_set2_test)[:, 1]
12 train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred2)
13 test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred2)
14
15 plt.plot(train_fpr, train_tpr, label="Train AUC =" + str(auc(train_fpr, train_tpr)))
16 plt.plot(test_fpr, test_tpr, label="Test AUC =" + str(auc(test_fpr, test_tpr)))
17 plt.legend()
18 plt.ylabel("True Positive Rate(TPR)")
19 plt.xlabel("False Positive Rate(FPR)")
20 plt.title("AUC")
21 plt.grid()
22 plt.show()
23 # print(y_train_pred.shape)
24 # y_test_pred.shape

```

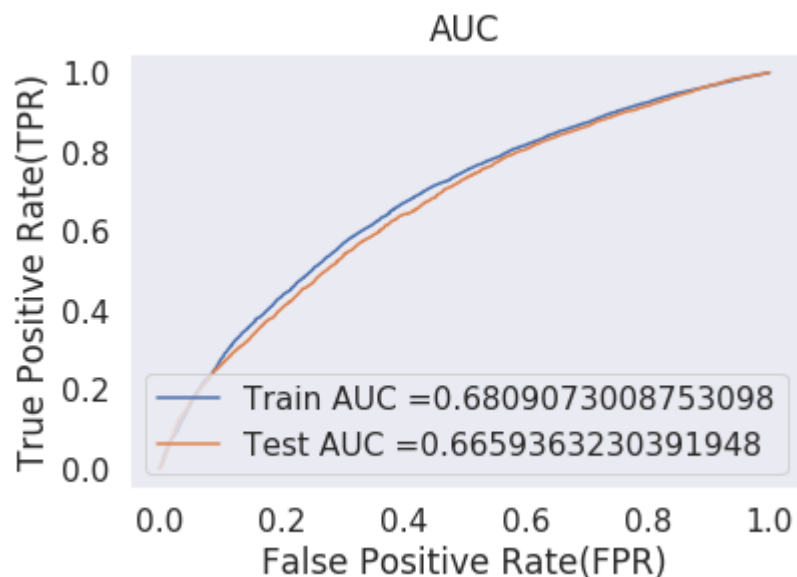


```

1 # using the L2 regularization
2 from sklearn.calibration import CalibratedClassifierCV
3 from sklearn.metrics import roc_curve, auc
4 Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced', alpha = 10**-3)
5 Classifier_bow.fit(X_set2_train, y_train)
6
7 clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3, method='isotonic')
8 clfcalibrated.fit(X_set2_train, y_train)
9
10 y_train_pred = clfcalibrated.predict_proba(X_set2_train)[: , 1]
11 y_test_pred = clfcalibrated.predict_proba(X_set2_test)[: , 1]
12 train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
13 test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
14
15 plt.plot(train_fpr, train_tpr, label="Train AUC =" + str(auc(train_fpr, train_tpr)))
16 plt.plot(test_fpr, test_tpr, label="Test AUC =" + str(auc(test_fpr, test_tpr)))
17 plt.legend()
18 plt.ylabel("True Positive Rate(TPR)")
19 plt.xlabel("False Positive Rate(FPR)")
20 plt.title("AUC")
21 plt.grid()
22 plt.show()
23 # print(y_train_pred.shape)
24 # y_test_pred.shape

```





```

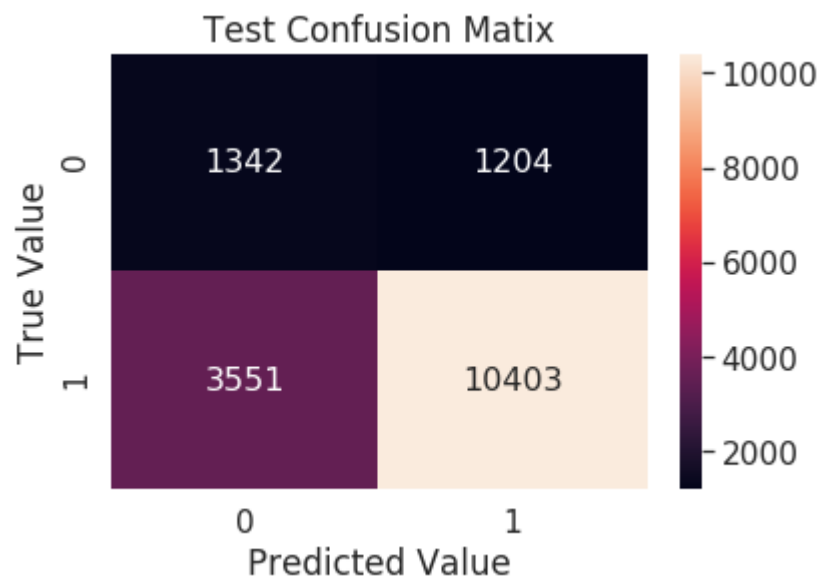
1 #CONFUSION MATRIX
2 import seaborn as sea
3 test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred2, te_thresholds, test_fpr, test_fpr)), range(2), range(2))
4 sea.set(font_scale=1.4)
5 sea.heatmap(test_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
6 plt.xlabel("Predicted Value")
7 plt.ylabel("True Value")
8 plt.title("Test Confusion Matix")
9

```



the maximum value of $tpr \cdot (1 - fpr)$ 0.25 for threshold 0.81

Text(0.5, 1.0, 'Test Confusion Matix')



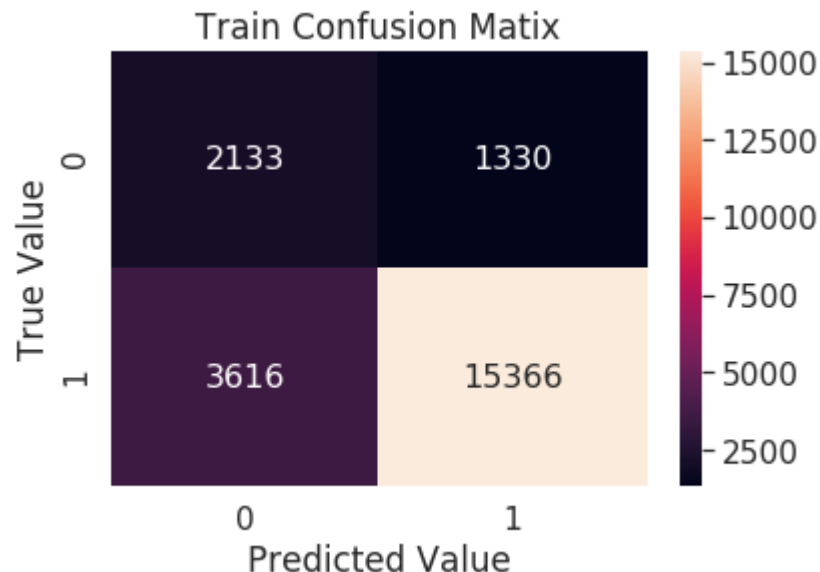
```

1 import seaborn as sea
2 train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred2, te_thresholds, train_fpr, train_fpr)), range(2)
3 sea.set(font_scale=1.4)
4 sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
5 plt.xlabel("Predicted Value")
6 plt.ylabel("True Value")
7 plt.title("Train Confusion Matix")

```



the maximum value of $\text{tpr} \cdot (1 - \text{fpr})$ 0.25 for threshold 0.79
 Text(0.5, 1.0, 'Train Confusion Matix')



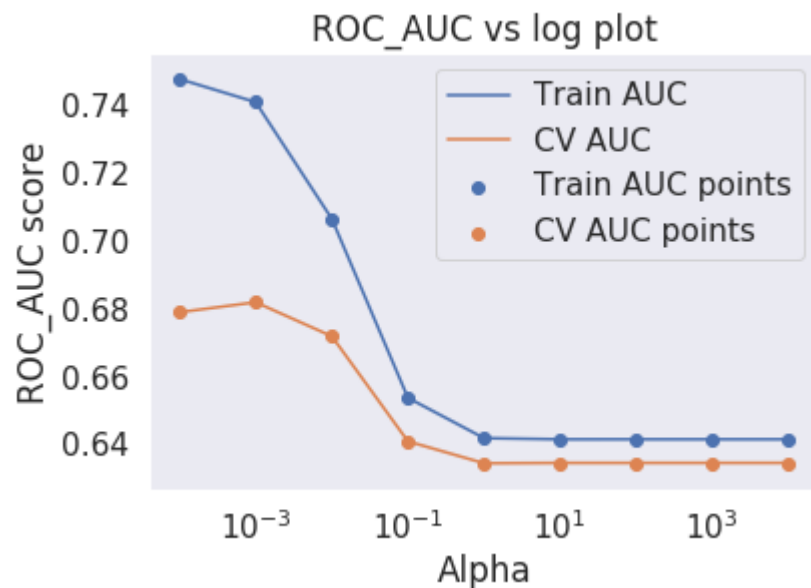
▼ Applying the SGDClassifier on SET:3

```

1 #BY USING "L2" REGULARISER
2 # hyperparameter tuning with l2 reg
3 parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
4 sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
5
6 classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
7 classifier.fit(X_set3_train, y_train)
8
9 train_auc = classifier.cv_results_['mean_train_score']
10 cv_auc= classifier.cv_results_['mean_test_score']
11
12 plt.plot(parameters['alpha'], train_auc, label='Train AUC')
13 plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
14 plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
15 plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
16 plt.legend()
17 plt.xscale('log')
18 plt.xlabel("Alpha")
19 plt.ylabel("ROC_AUC score")
20 plt.title("ROC_AUC vs log plot")
21 plt.grid()

```

```
22 plt.show()
```

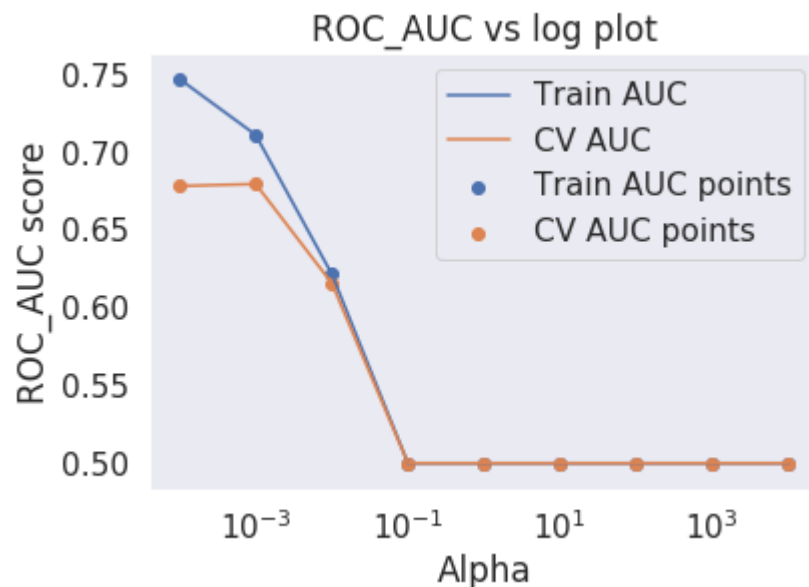


```

1 #BY USING "L1" REGULARISER
2 parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
3 sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
4
5 classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
6 classifier.fit(X_set3_train, y_train)
7
8 train_auc = classifier.cv_results_['mean_train_score']
9 cv_auc= classifier.cv_results_['mean_test_score']
10
11 plt.plot(parameters['alpha'], train_auc, label='Train AUC')
12 plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
13 plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
14 plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
15 plt.legend()
16 plt.xscale('log')
17 plt.xlabel("Alpha")
18 plt.ylabel("ROC_AUC score")
19 plt.title("ROC_AUC vs log plot")
20 plt.grid()
21 plt.show()
22

```

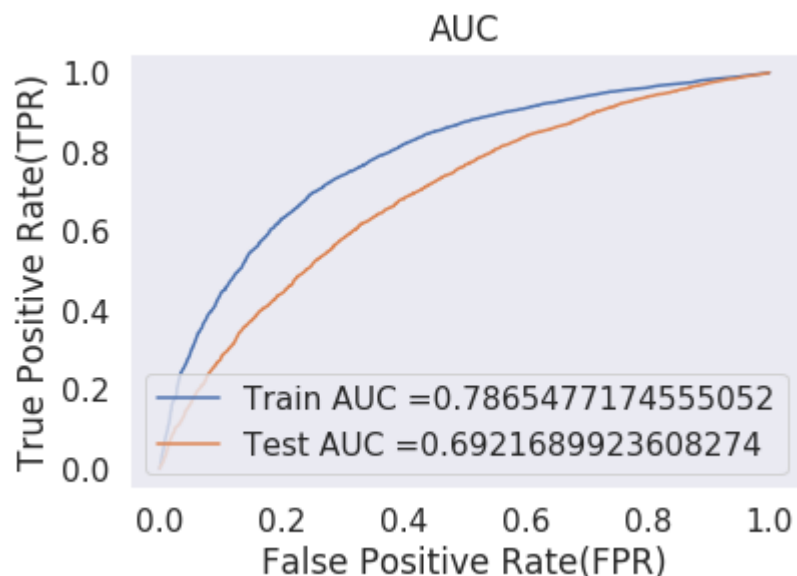




```

1 # using the L2 regularization
2 from sklearn.calibration import CalibratedClassifierCV
3 from sklearn.metrics import roc_curve, auc
4 Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced', alpha = 10**-3)
5 Classifier_bow.fit(X_set2_train, y_train)
6
7 clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3, method='isotonic')
8 clfcalibrated.fit(X_set2_train, y_train)
9
10 y_train_pred3 = clfcalibrated.predict_proba(X_set2_train)[:, 1]
11 y_test_pred3 = clfcalibrated.predict_proba(X_set2_test)[:, 1]
12 train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred3)
13 test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred3)
14
15 plt.plot(train_fpr, train_tpr, label="Train AUC =" + str(auc(train_fpr, train_tpr)))
16 plt.plot(test_fpr, test_tpr, label="Test AUC =" + str(auc(test_fpr, test_tpr)))
17 plt.legend()
18 plt.ylabel("True Positive Rate(TPR)")
19 plt.xlabel("False Positive Rate(FPR)")
20 plt.title("AUC")
21 plt.grid()
22 plt.show()
23 # print(y_train_pred.shape)
24 # y_test_pred.shape

```

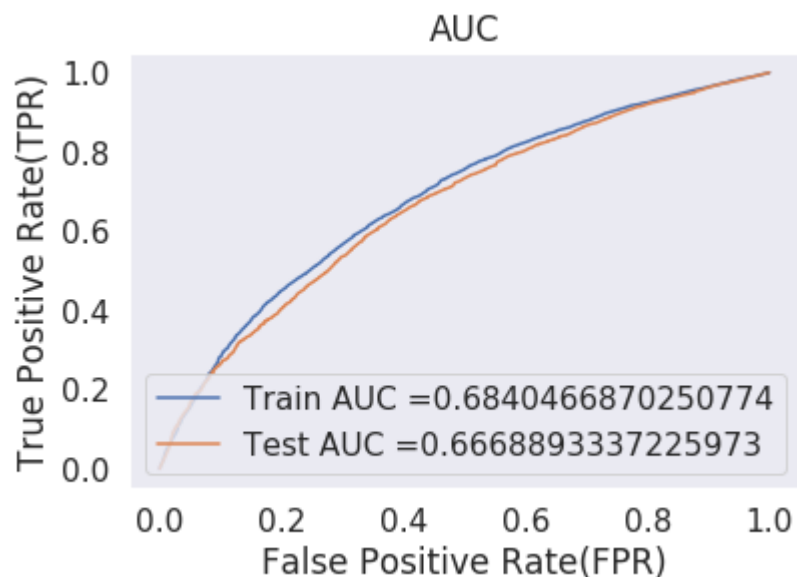


```

1 # using the L2 regularization
2 from sklearn.calibration import CalibratedClassifierCV
3 from sklearn.metrics import roc_curve, auc
4 Classifier_bow = SGDClassifier(loss = 'hinge', class_weight = 'balanced', penalty = 'l1', alpha = 10**-3)
5 Classifier_bow.fit(X_set2_train, y_train)
6
7 clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3, method='isotonic')
8 clfcalibrated.fit(X_set2_train, y_train)
9
10 y_train_pred = clfcalibrated.predict_proba(X_set2_train)[:, 1]
11 y_test_pred = clfcalibrated.predict_proba(X_set2_test)[:, 1]
12 train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
13 test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
14
15 plt.plot(train_fpr, train_tpr, label="Train AUC =" + str(auc(train_fpr, train_tpr)))
16 plt.plot(test_fpr, test_tpr, label="Test AUC =" + str(auc(test_fpr, test_tpr)))
17 plt.legend()
18 plt.ylabel("True Positive Rate(TPR)")
19 plt.xlabel("False Positive Rate(FPR)")
20 plt.title("AUC")
21 plt.grid()
22 plt.show()
23 # print(y_train_pred.shape)
24 # y_test_pred.shape

```





```

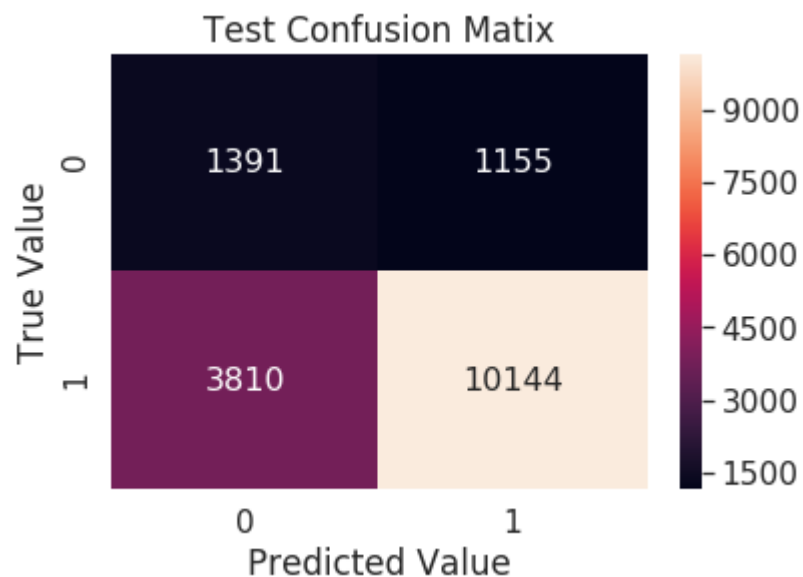
1 #CONFUSION MATRIX as per L2 Regularization
2 import seaborn as sea
3 test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred3, te_thresholds, test_fpr, test_fpr)), range(2), range(2))
4 sea.set(font_scale=1.4)
5 sea.heatmap(test_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
6 plt.xlabel("Predicted Value")
7 plt.ylabel("True Value")
8 plt.title("Test Confusion Matix")

```



the maximum value of $\text{tpr} \cdot (1 - \text{fpr})$ 0.25 for threshold 0.81

Text(0.5, 1.0, 'Test Confusion Matix')



```

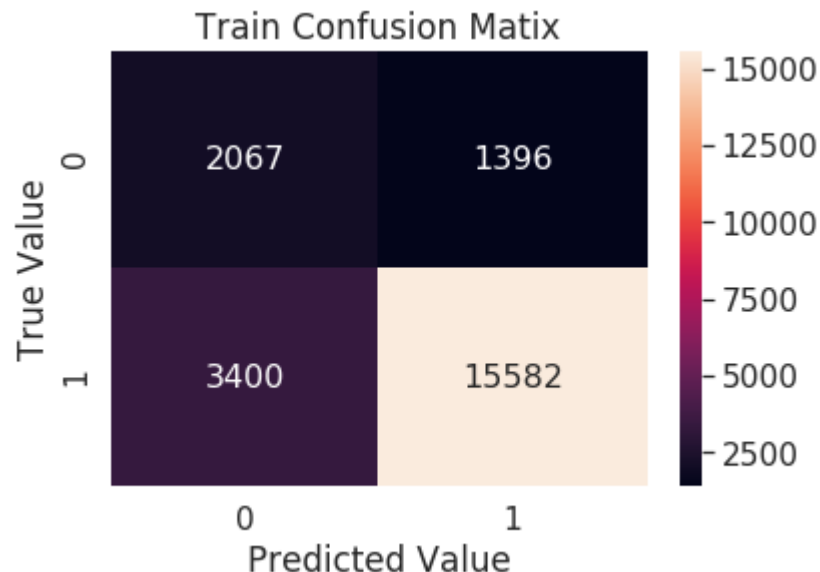
1 #CONFUSION MATRIX as per L2 Regularization
2 train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred3,te_thresholds,train_fpr,train_fpr)), range(2,
3 sea.set(font_scale=1.4)
4 sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
5 plt.xlabel("Predicted Value")
6 plt.ylabel("True Value")
7 plt.title("Train Confusion Matix")

```



the maximum value of $\text{tpr} \times (1 - \text{fpr})$ 0.25 for threshold 0.79

Text(0.5, 1.0, 'Train Confusion Matix')



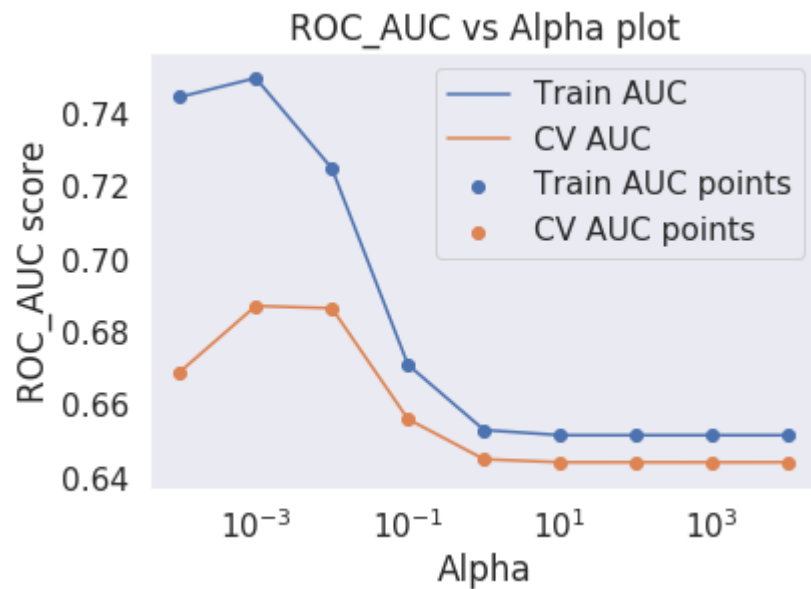
▼ Applying the SGDClassifier on SET:4

```

1 #BY USING "l2" REGULARISER
2
3 parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
4 SV = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',)
5
6 classifier = GridSearchCV(SV, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
7 classifier.fit(X_set4_train, y_train)
8
9 train_auc= classifier.cv_results_['mean_train_score']
10 cv_auc = classifier.cv_results_['mean_test_score']
11
12 plt.plot(parameters['alpha'], train_auc, label='Train AUC')
13 plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
14 plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
15 plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
16 plt.legend()
17 plt.xscale('log')
18 plt.xlabel("Alpha")
19 plt.ylabel("ROC_AUC score")
20 plt.title("ROC_AUC vs Alpha plot")
21 plt.grid()

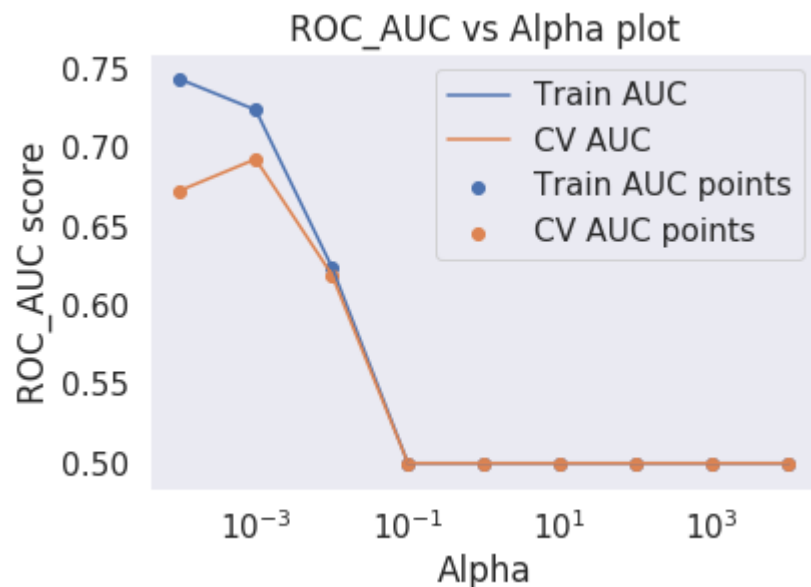
```

```
22 plt.show()
```



```
1 """#BY USING "L1" REGULARIZER
2
3 parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
4 SV = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
5
6 classifier = GridSearchCV(SV, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
7 classifier.fit(X_set4_train, y_train)
8
9 train_auc= classifier.cv_results_['mean_train_score']
10 cv_auc = classifier.cv_results_['mean_test_score']
11
12 plt.plot(parameters['alpha'], train_auc, label='Train AUC')
13 plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
14 plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
15 plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
16 plt.legend()
17 plt.xscale('log')
18 plt.xlabel("Alpha")
19 plt.ylabel("ROC_AUC score")
20 plt.title("ROC_AUC vs Alpha plot")
21 plt.grid()
22 plt.show()
```

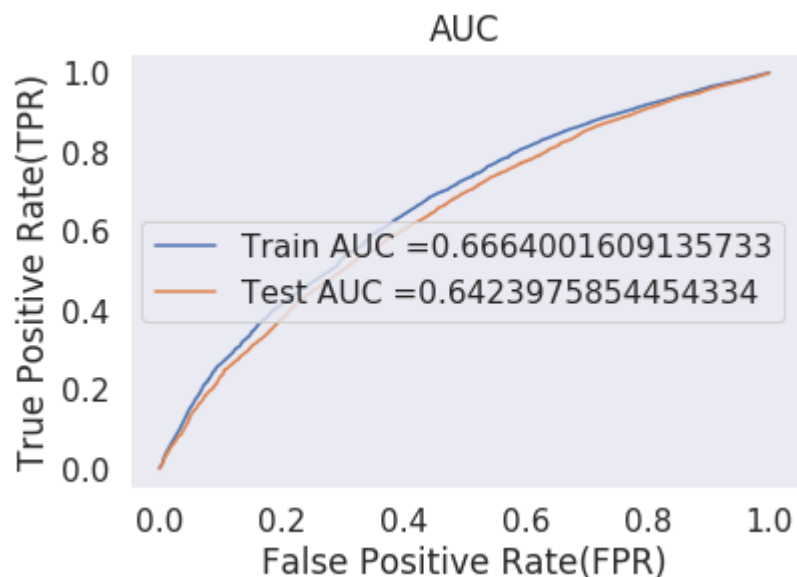




```

1 # using the L2 regularization
2 from sklearn.calibration import CalibratedClassifierCV
3 from sklearn.metrics import roc_curve, auc
4 Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced', alpha = 10**-2)
5 Classifier_bow.fit(X_set2_train, y_train)
6
7 clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3, method='isotonic')
8 clfcalibrated.fit(X_set2_train, y_train)
9
10 y_train_pred4 = clfcalibrated.predict_proba(X_set2_train)[: , 1]
11 y_test_pred4 = clfcalibrated.predict_proba(X_set2_test)[: , 1]
12 train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred4)
13 test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred4)
14
15 plt.plot(train_fpr, train_tpr, label="Train AUC =" + str(auc(train_fpr, train_tpr)))
16 plt.plot(test_fpr, test_tpr, label="Test AUC =" + str(auc(test_fpr, test_tpr)))
17 plt.legend()
18 plt.ylabel("True Positive Rate(TPR)")
19 plt.xlabel("False Positive Rate(FPR)")
20 plt.title("AUC")
21 plt.grid()
22 plt.show()
23 # print(y_train_pred.shape)
24 # y_test_pred.shape

```

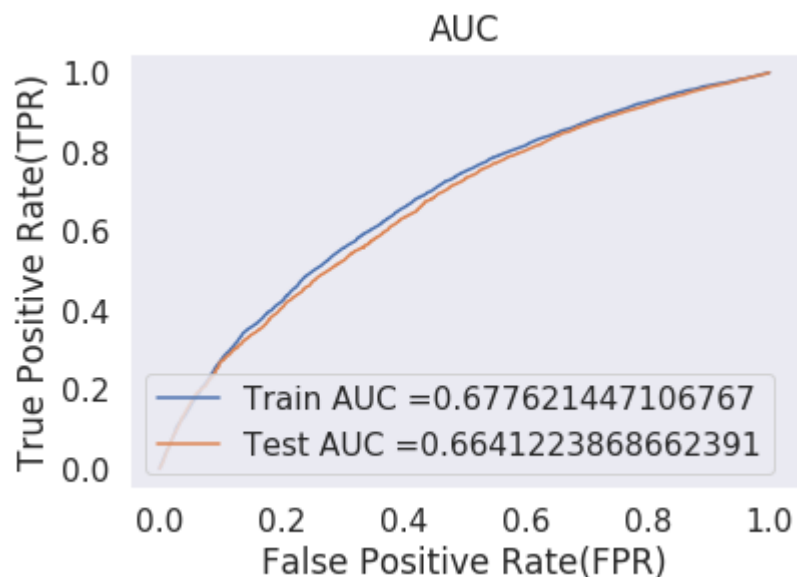


```

1 # using the L2 regularization
2 from sklearn.calibration import CalibratedClassifierCV
3 from sklearn.metrics import roc_curve, auc
4 Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced' ,alpha = 10**-3)
5 Classifier_bow.fit(X_set2_train ,y_train)
6
7 clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3 , method='isotonic')
8 clfcalibrated.fit(X_set2_train, y_train)
9
10 y_train_pred = clfcalibrated.predict_proba(X_set2_train)[: , 1]
11 y_test_pred = clfcalibrated.predict_proba(X_set2_test)[: , 1]
12 train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
13 test_fpr, test_tpr, te_thresholds = roc_curve(y_test , y_test_pred)
14
15 plt.plot(train_fpr, train_tpr, label="Train AUC =" +str(auc(train_fpr, train_tpr)))
16 plt.plot(test_fpr, test_tpr, label="Test AUC =" +str(auc(test_fpr, test_tpr)))
17 plt.legend()
18 plt.ylabel("True Positive Rate(TPR)")
19 plt.xlabel("False Positive Rate(FPR)").
20 plt.title("AUC")
21 plt.grid()
22 plt.show()
23 # print(y_train_pred.shape)
24 # y_test_pred.shape

```





```

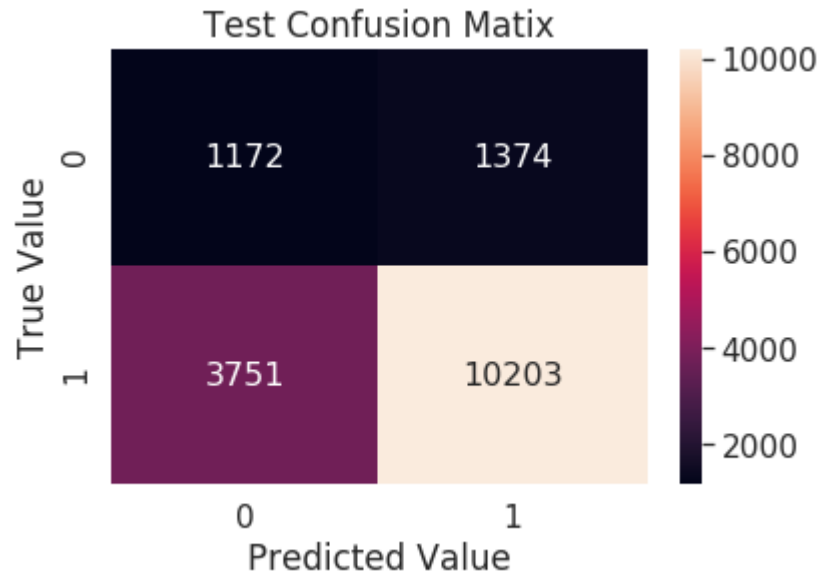
1 #CONFUSION MATRIX
2 import seaborn as sea
3 test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred4, te_thresholds, test_fpr, test_fpr)), range(2), range(2))
4 sea.set(font_scale=1.4)
5 sea.heatmap(test_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
6 plt.xlabel("Predicted Value")
7 plt.ylabel("True Value")
8 plt.title("Test Confusion Matix")
9

```



the maximum value of $\text{tpr} \cdot (1 - \text{fpr})$ 0.25 for threshold 0.82

Text(0.5, 1.0, 'Test Confusion Matix')



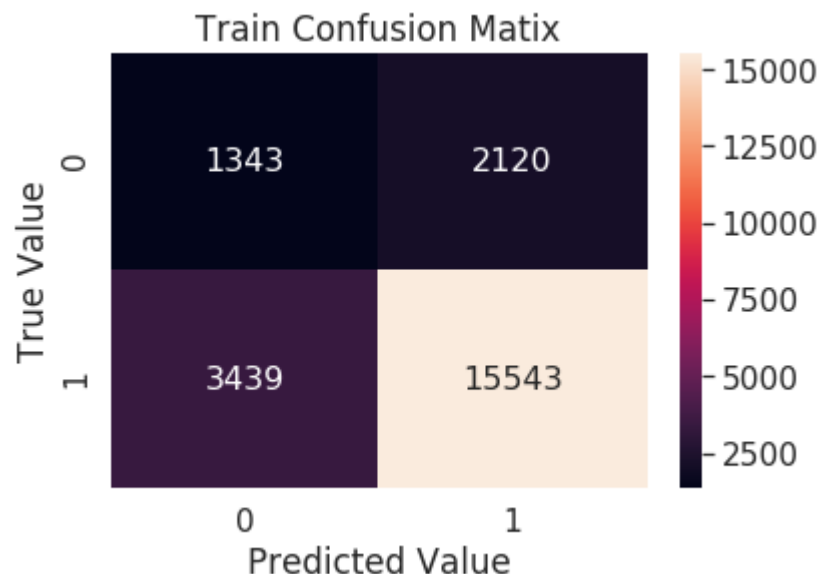
```

1 #CONFUSION MATRIX
2 import seaborn as sea
3 train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred4,te_thresholds,train_fpr,train_fpr)), range(2)
4 sea.set(font_scale=1.4)
5 sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
6 plt.xlabel("Predicted Value")
7 plt.ylabel("True Value")
8 plt.title("Train Confusion Matix")
9

```



the maximum value of $\text{tpr} \times (1 - \text{fpr})$ 0.25 for threshold 0.79
 Text(0.5, 1.0, 'Train Confusion Matix')



▼ Applying the SGDClassifier on SET:5

1. school_state : categorical data
2. clean_categories : categorical data
3. clean_subcategories : categorical data
4. project_grade_category : categorical data
5. teacher_prefix : categorical data
6. quantity : numerical data
7. teacher_number_of_previously_posted_projects : numerical data
8. price : numerical data
9. sentiment score's of each of the essay : numerical data
10. number of words in the title : numerical data
11. number of words in the combine essays : numerical data
12. Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method : numerical data

```
1 print(X_train.shape)
```

```
2 print(X_test.shape)
3 print(X_cv.shape)
```

```
(22445, 19)
(16500, 19)
(11055, 19)
```

▼ Numeber of words in title feature number 10 in above descritpion

```
1 # For train data
2 title_length_train=[]
3 for i in range(0,22445):
4     title_length_train.append(len(X_train["project_title"][i].split()))
5
6 title_length_train=np.array(title_length_train)
7
8 #for test data titles
9 title_length_test=[]
10 for i in range(0,16500):
11     title_length_test.append(len(X_test["project_title"][i].split()))
12
13 title_length_test=np.array(title_length_test)
14 #for cv data titles
15 title_length_cv=[]
16 for i in range(0,11055):
17     title_length_cv.append(len(X_cv["project_title"][i].split()))
18
19 title_length_cv=np.array(title_length_cv)
```

▼ Number of words in combined essays

```
1 essay_length_test=[]
2 for i in range(0,16500):
3     essay_length_test.append(len(X_test["essay"][i].split()))
4
5 essay_length_test=np.array(essay_length_test)
6
7 #for cv data essay
8
9 essay_length_cv=[]
10 for i in range(0,11055):
11     essay_length_cv.append(len(X_cv["essay"][i].split()))
12
13 essay_length_cv=np.array(essay_length_cv)
```



```

14
15
16 #for train data essay
17
18 essay_length_train=[]
19 for i in range(0,22445):
20     essay_length_train.append(len(X_train["essay"][i].split()))
21
22 essay_length_train=np.array(essay_length_train).

```

▼ Sentiment scores of each combined essay's

```

1 """
2 The code is taken from
3 https://www.nltk.org/_modules/nltk/sentiment/vader.html
4 https://www.programcreek.com/python/example/100005/nltk.sentiment.vader.SentimentIntensityAnalyzer
5 https://www.nltk.org/api/nltk.sentiment.html
6
7 VADER sentiment analysis tools:
8
9 Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for
10 Sentiment Analysis of Social Media Text. Eighth International Conference on
11 Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.
12 """
13 import nltk
14 from nltk.sentiment.vader import SentimentIntensityAnalyzer
15 nltk.download('vader_lexicon')
16 #https://www.programcreek.com/python/example/100005/nltk.sentiment.vader.SentimentIntensityAnalyzer
17 def analyze_sentiment(df):
18     sentiments = []
19     sid = SentimentIntensityAnalyzer()
20     for i in tqdm(range(df.shape[0])):
21         line = df['essay'][i] # take one essay
22         sentiment = sid.polarity_scores(line)# calculate the sentiment
23         sentiments.append([sentiment['neg'], sentiment['pos'],
24                             sentiment['neu'], sentiment['compound']])# list of lists
25     df[['neg', 'pos', 'neu', 'compound']] = pd.DataFrame(sentiments)
26     df['Negative'] = df['compound'] < -0.1
27     df['Positive'] = df['compound'] > 0.1
28     return df

```

↳ [nltk_data] Downloading package vader_lexicon to /root/nltk_data...

```

1 X_train=analyze_sentiment(X_train)
2 X_test=analyze_sentiment(X_test)

```

```
3 X_cv=analyze_sentiment(X_cv)
```

```
100%|██████████| 22445/22445 [01:00<00:00, 371.76it/s]
100%|██████████| 16500/16500 [00:43<00:00, 378.97it/s]
100%|██████████| 11055/11055 [00:29<00:00, 374.51it/s]
```

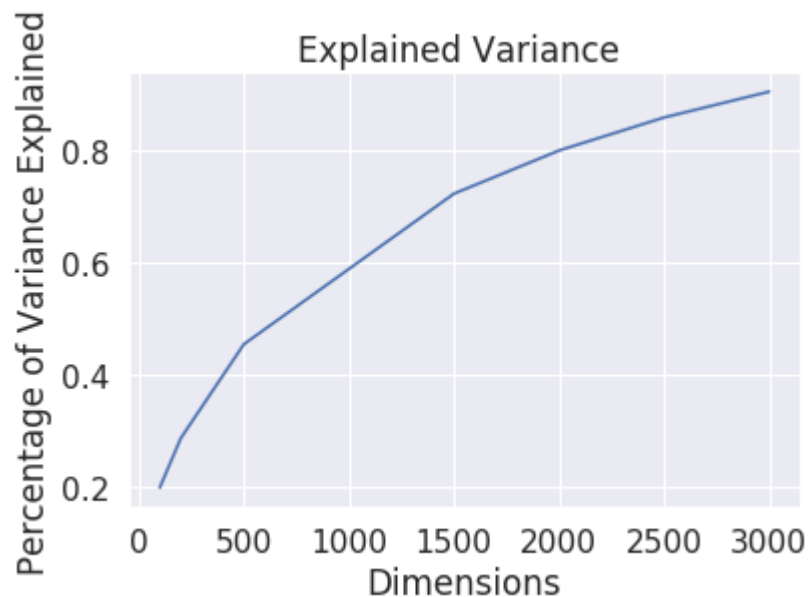
▼ TruncatedSVD on TfidfVectorizer of essay text

```
1 # considering only 5000 Points
2 X_train_tf_essay=X_train_tf_essay[:,0:5000]
3 X_cv_tf_essay=X_cv_tf_essay[:,0:5000]
4 X_test_tf_essay=X_test_tf_essay[:,0:5000]
5 from sklearn.decomposition import TruncatedSVD
6 #https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
7 Dimensions = [100,200,500,1500,2000,2500,3000]
8 Variance_Explained = []
9 for i in tqdm(Dimensions):
10     svd = TruncatedSVD(n_components = i, random_state = 42)
11     svd.fit(X_train_tf_essay)
12     Variance_Explained.append(svd.explained_variance_ratio_.sum())
```

```
100%|██████████| 7/7 [11:33<00:00, 146.09s/it]
```

```
1 plt.xlabel("Dimensions")
2 plt.ylabel("Percentage of Variance Explained")
3 plt.title("Explained Variance ")
4 plt.plot(Dimensions,Variance_Explained)
5 plt.show()
```

```
↳
```



```

1 # from the above graph we can see that 90 % of the varince is explained by 3000 diemnsions
2 svd = TruncatedSVD(n_components= 3000)
3 svd.fit(X_train_tf_essay)
4 #Transforms:
5 #Train SVD
6 X_train_tf_essay= svd.transform(X_train_tf_essay )
7 #Test SVD
8 X_test_tf_essay = svd.transform(X_test_tf_essay )
9 #CV SVD
10 X_cv_tf_essay = svd.transform(X_cv_tf_essay )

```

```

1 #for train
2 pos=list(X_train['pos'])
3 pos=np.array(pos)
4 neg=list(X_train['neg'])
5 neg=np.array(neg)
6 com=list(X_train['compound'])
7 com=np.array(com)
8 # combine all
9 from scipy.sparse import hstack
10 # with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
11 X_set5_train = hstack((X_train_teacher_prefix,X_train_cat,X_train_subcat,X_train_project_grade_category,X_train_school_state,train_
12                        essay_length_train.reshape(-1,1),title_length_train.reshape(-1,1),
13                        pos.reshape(-1,1),neg.reshape(-1,1),com.reshape(-1,1)))

```

14

```

1 #for test
2 pos=list(X_test['pos'])
3 pos=np.array(pos)
4 neg=list(X_test['neg'])
5 neg=np.array(neg)
6 com=list(X_test['compound'])
7 com=np.array(com)
8 # combine all
9 from scipy.sparse import hstack
10 # with the same hstack function we are concatenating a sparse matrix and a dense matrix :)
11 X_set5_test = hstack((X_test_teacher_prefix,X_test_cat,X_test_subcat ,X_test_project_grade_category,X_test_school_state,
12                      test_qnty_standar,test_price_standar,test_prev_proj_standar,
13                      essay_length_test.reshape(-1,1),title_length_test.reshape(-1,1),
14                      pos.reshape(-1,1),neg.reshape(-1,1),com.reshape(-1,1),))

```

▸ Applying SGDClassifier on SET 5

```

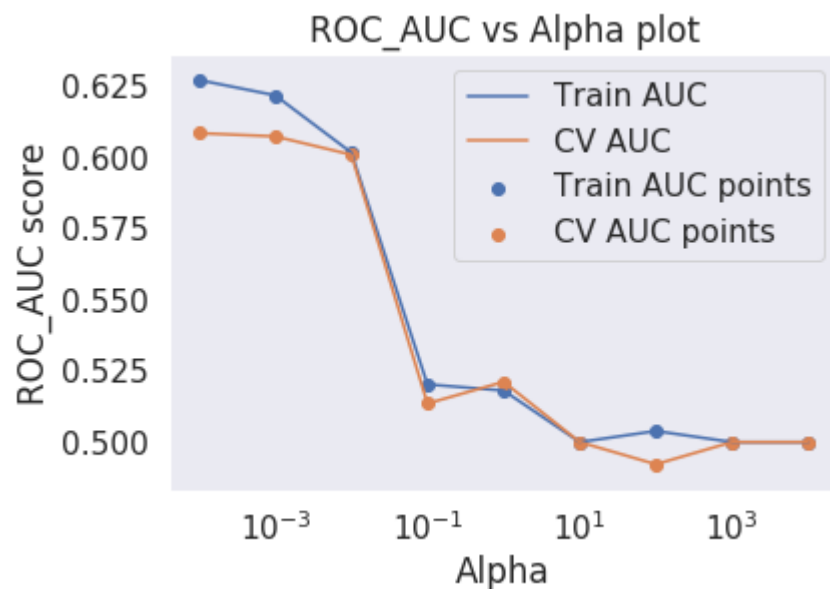
1 #BY USING L1 REGULARISER
2 from sklearn.metrics import roc_auc_score
3 import matplotlib.pyplot as plt
4 from sklearn.model_selection import train_test_split
5 from sklearn.model_selection import GridSearchCV
6 #from sklearn.datasets import *
7 from sklearn import linear_model
8 from sklearn.linear_model import SGDClassifier
9 from sklearn import svm
10 # hyperparameter tuning with l2 reg
11 """#we are using L1 Regularizer
12 parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
13 SV = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced',)
14 classifier = GridSearchCV(SV, parameters, cv= 3, scoring='roc_auc',return_train_score=True)
15 classifier.fit(X_set5_train, y_train)
16
17 train_auc= classifier.cv_results_['mean_train_score']
18 cv_auc = classifier.cv_results_['mean_test_score']
19
20 plt.plot(parameters['alpha'], train_auc, label='Train AUC')
21 plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
22
23 plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
24 plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
25 plt.legend()
26 plt.xscale("log")

```

```

27 plt.xlabel("Alpha")
28 plt.ylabel("ROC_AUC score")
29 plt.title("ROC_AUC vs Alpha plot")
30 plt.grid()
31 plt.show()

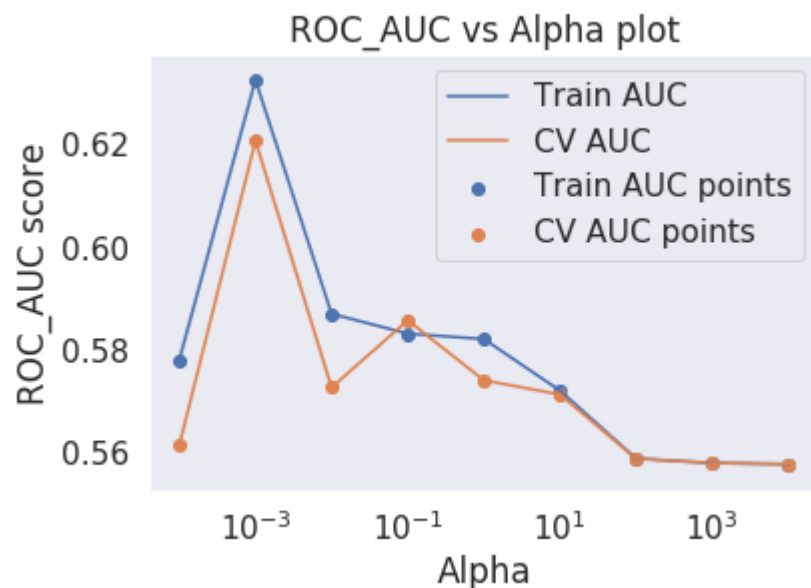
```



```

1 #BY USING L2 REGULARISER
2
3 parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
4 SV = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',)
5 classifier = GridSearchCV(SV, parameters, cv= 3, scoring='roc_auc',return_train_score=True)
6 classifier.fit(X_set5_train, y_train)
7
8 train_auc= classifier.cv_results_['mean_train_score']
9 cv_auc = classifier.cv_results_['mean_test_score']
10
11 plt.plot(parameters['alpha'], train_auc, label='Train AUC')
12 plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
13 plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
14 plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
15 plt.legend()
16 plt.xscale("log")
17 plt.xlabel("Alpha")
18 plt.ylabel("ROC_AUC score")
19 plt.title("ROC_AUC vs Alpha plot")
20 plt.grid()
21 plt.show()

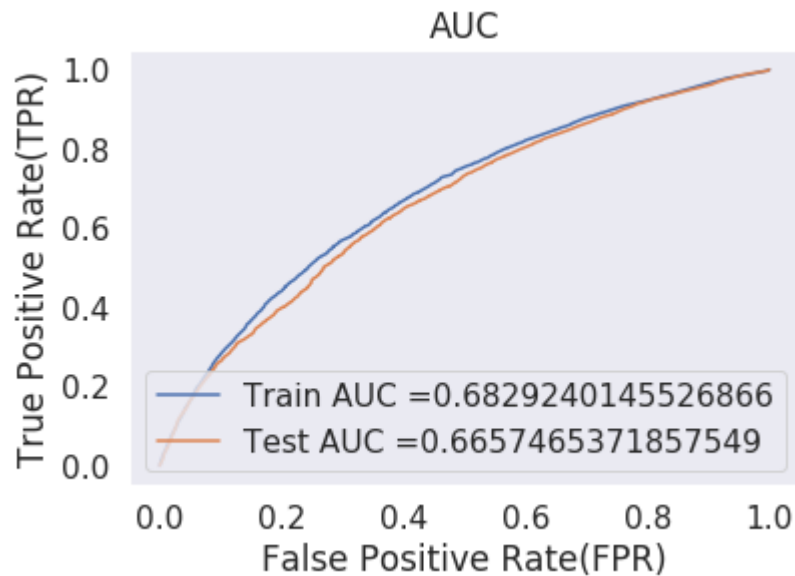
```



```

1 # using the L2 regularization
2 from sklearn.calibration import CalibratedClassifierCV
3 from sklearn.metrics import roc_curve, auc
4 Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced', alpha = 10**-3)
5 Classifier_bow.fit(X_set2_train, y_train)
6
7 clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3, method='isotonic')
8 clfcalibrated.fit(X_set2_train, y_train)
9
10 y_train_pred = clfcalibrated.predict_proba(X_set2_train)[:, 1]
11 y_test_pred = clfcalibrated.predict_proba(X_set2_test)[:, 1]
12 train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
13 test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
14
15 plt.plot(train_fpr, train_tpr, label="Train AUC =" + str(auc(train_fpr, train_tpr)))
16 plt.plot(test_fpr, test_tpr, label="Test AUC =" + str(auc(test_fpr, test_tpr)))
17 plt.legend()
18 plt.ylabel("True Positive Rate(TPR)")
19 plt.xlabel("False Positive Rate(FPR)")
20 plt.title("AUC")
21 plt.grid()
22 plt.show()
23 # print(y_train_pred.shape)
24 # y_test_pred.shape

```

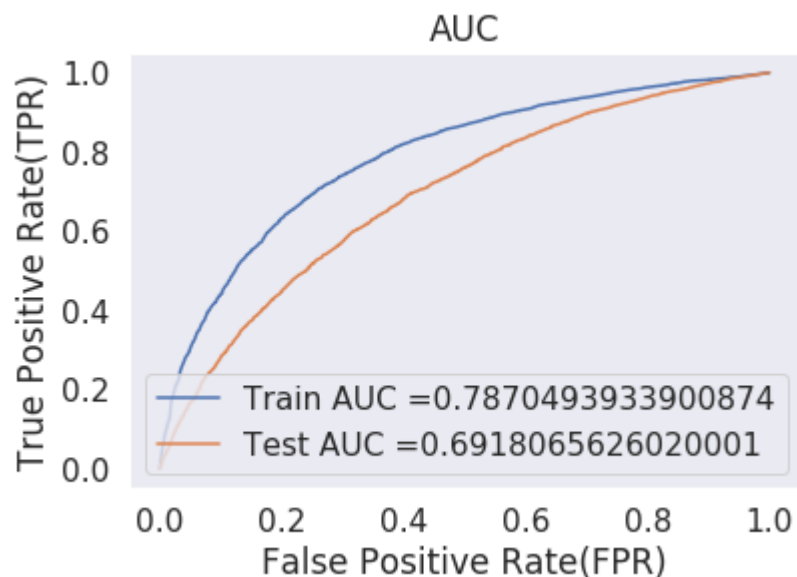


```

1 # using the L2 regularization
2 from sklearn.calibration import CalibratedClassifierCV
3 from sklearn.metrics import roc_curve, auc
4 Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced', alpha = 10**-3)
5 Classifier_bow.fit(X_set2_train, y_train)
6
7 clfcalibrated = CalibratedClassifierCV(Classifier_bow, cv=3, method='isotonic')
8 clfcalibrated.fit(X_set2_train, y_train)
9
10 y_train_pred6 = clfcalibrated.predict_proba(X_set2_train)[: , 1]
11 y_test_pred6 = clfcalibrated.predict_proba(X_set2_test)[: , 1]
12 train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred6)
13 test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred6)
14
15 plt.plot(train_fpr, train_tpr, label="Train AUC =" + str(auc(train_fpr, train_tpr)))
16 plt.plot(test_fpr, test_tpr, label="Test AUC =" + str(auc(test_fpr, test_tpr)))
17 plt.legend()
18 plt.ylabel("True Positive Rate(TPR)")
19 plt.xlabel("False Positive Rate(FPR)")
20 plt.title("AUC")
21 plt.grid()
22 plt.show()
23 # print(y_train_pred.shape)
24 # y_test_pred.shape

```





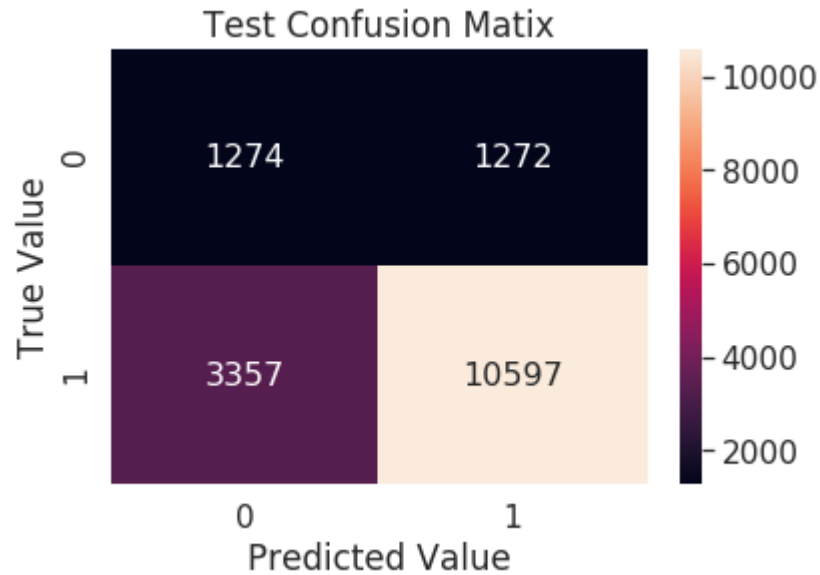
```

1 #CONFUSION MATRIX
2 import seaborn as sea
3 test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred6, te_thresholds, test_fpr, test_fpr)), range(2), range(2))
4 sea.set(font_scale=1.4)
5 sea.heatmap(test_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
6 plt.xlabel("Predicted Value")
7 plt.ylabel("True Value")
8 plt.title("Test Confusion Matix")

```



the maximum value of $tpr \cdot (1 - fpr)$ 0.25 for threshold 0.8
 Text(0.5, 1.0, 'Test Confusion Matix')



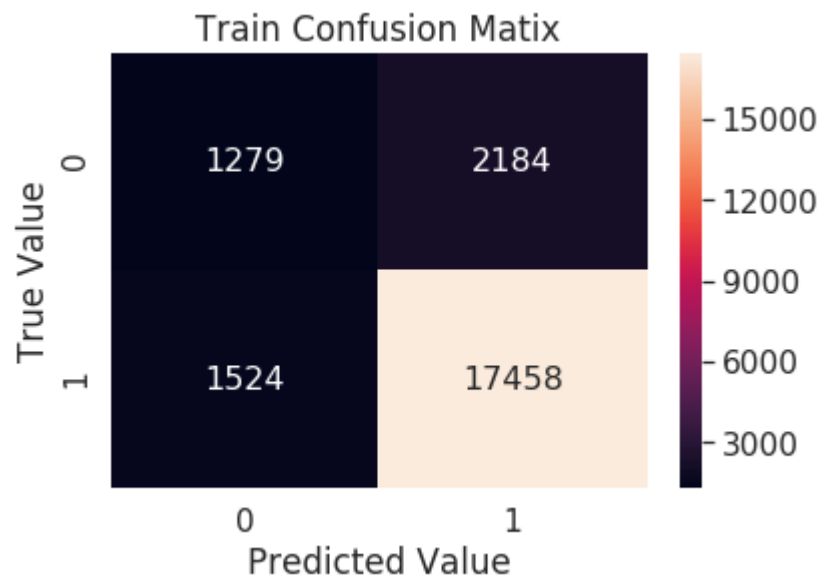
```

1 #CONFUSION MATRIX
2 import seaborn as sea
3 train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred6, te_thresholds, train_fpr, train_fpr)), range(2,
4 sea.set(font_scale=1.4)
5 sea.heatmap(train_confusion_matrix, annot = True, annot_kws={"size":16}, fmt = 'd')
6 plt.xlabel("Predicted Value")
7 plt.ylabel("True Value")
8 plt.title("Train Confusion Matix")
  
```



the maximum value of $\text{tpr} \times (1 - \text{fpr})$ 0.25 for threshold 0.73


Text(0.5, 1.0, 'Train Confusion Matix')



```

1 # Please compare all your models using Prettytable library
2 #how to use pretty table http://zetcode.com/python/prettytable/
3 from prettytable import PrettyTable
4 tb = PrettyTable()
5 tb.field_names= ("Vectorizer", " L1 Alpha ", " L2 Alpha " , " L1 AUC ", " L2 AUC ")
6 tb.add_row(["BOW ", 10**-3,10**-2, 67 ,70])
7 tb.add_row(["Tf - Idf ", 10**-3,10**-3, 66 ,70])
8 tb.add_row(["AVG - W2V", 10**-3,10**-3, 66 ,69])
9 tb.add_row(["AVG - Tf - Idf",10**-3,10**-2, 66 ,64])
10 tb.add_row(["SVD-Top 3000 Features", 10**-3,10**-3, 66 ,69])
11 print(tb.get_string(titles = "Observations for diffrent data metrix "))

```



Vectorizer	L1 Alpha	L2 Alpha	L1 AUC	L2 AUC
BOW	0.001	0.01	67	70
Tf - Idf	0.001	0.001	66	70
AVG - W2V	0.001	0.001	66	69
AVG - Tf - Idf	0.001	0.01	66	64
SVD-Top 3000 Features	0.001	0.001	66	69

