LOGISTIC REGRESSION

Note:

- Since we already did all these in previous assignments and we can use the same from it.

In [0]:

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from tqdm import tqdm
```

In [2]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

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

In [3]:

```
! ls '/content/drive/My Drive/Applied AI/Datasets/New Donors/'
```

```
glove_vectors   Preprocessed_inc_others.csv  train_data.csv
PreProcessed.csv  resources.csv
```

In [4]:

```
data = pd.read_csv('/content/drive/My Drive/Applied AI/Datasets/New
Donors/Preprocessed_inc_others.csv')
data.head()
```

Out[4]:

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved
0	0	ca	mrs	grades_prek_2	53	1
1	1	ut	ms	grades_3_5	4	1

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved
2	2	ca	mrs	grades_prek_2	10	1
3	3	ga	mrs	grades_prek_2	2	1
4	4	wa	mrs	grades_3_5	2	1

In [5]:

```
data.describe()
```

Out[5]:

	Unnamed: 0	teacher_number_of_previously_posted_projects	project_is_approved	price	quantity	sentiment_sc
count	109248.000000	109248.000000	109248.000000	109248.000000	109248.000000	109248.000
mean	54623.500000	11.153165	0.848583	298.119343	16.965610	0.2100
std	31537.325441	27.777154	0.358456	367.498030	26.182942	0.0835
min	0.000000	0.000000	0.000000	0.660000	1.000000	-0.1897
25%	27311.750000	0.000000	1.000000	104.310000	4.000000	0.1543
50%	54623.500000	2.000000	1.000000	206.220000	9.000000	0.2082
75%	81935.250000	9.000000	1.000000	379.000000	21.000000	0.2643
max	109247.000000	451.000000	1.000000	9999.000000	930.000000	0.6633

In [6]:

```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(2)
```

Out[6]:

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	cl
0	0	ca	mrs	grades_prek_2	53	math_science	
1	1	ut	ms	grades_3_5	4	specialneeds	

In [7]:

```
y = y.reshape(-1,1)
print(y.shape)
```

```
(109248, 1)
```

Splitting the data

In [0]:

```
from sklearn.model_selection import train_test_split
data_train, data_test, label_train, label_test = train_test_split(X, y, test_size=0.33, stratify=y,
random_state=42)
```

In [9]:

```
print(data_train.shape)
print(data_test.shape)
print(label_train.shape)
print(label_test.shape)
```

```
(73196, 14)
(36052, 14)
(73196, 1)
(36052, 1)
```

In [0]:

```
X_train = data_train
X_test = data_test
y_train = label_train
y_test = label_test
```

In [0]:

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

```
(73196, 14)
(36052, 14)
(73196, 1)
(36052, 1)
```

1. Vectorizing all features

1.1 School State

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_1 = CountVectorizer(list(X_train['school_state'].values), lowercase=False, binary=True)
```

In [0]:

```
X_train_Sstate = vectorizer_1.fit_transform(X_train['school_state'].values)
X_test_Sstate = vectorizer_1.transform(X_test['school_state'].values)
```

In [13]:

```
print(X_train_Sstate.shape)
print(X_test_Sstate.shape)
```

```
(73196, 51)
(36052, 51)
```

1.2 Clean Categories

In [0]:

```
vectorizer_2 = CountVectorizer(list(X_train['clean_categories'].values), lowercase=False,
binary=True)
```

In [0]:

```
X_train_cat = vectorizer_2.fit_transform(X_train['clean_categories'].values)
X_test_cat = vectorizer_2.transform(X_test['clean_categories'].values)
```

In [16]:

```
print(X_train_cat.shape)
print(X_test_cat.shape)
```

```
(73196, 9)
(36052, 9)
```

1.3 Clean Sub categories

In [0]:

```
vectorizer_3 = CountVectorizer(list(X_train['clean_subcategories'].values), lowercase=False,
binary=True)
```

In [0]:

```
X_train_subcat = vectorizer_3.fit_transform(X_train['clean_subcategories'].values)
X_test_subcat = vectorizer_3.transform(X_test['clean_subcategories'].values)
```

In [19]:

```
print(X_train_subcat.shape)
print(X_test_subcat.shape)
```

```
(73196, 30)
(36052, 30)
```

1.4 Project Grade Category

In [0]:

```
vectorizer_4 = CountVectorizer(list(X_train['project_grade_category'].values), lowercase=False,
binary=True)
```

In [0]:

```
X_train_grade = vectorizer_4.fit_transform(X_train['project_grade_category'].values)
X_test_grade = vectorizer_4.transform(X_test['project_grade_category'].values)
```

In [22]:

```
print(X_train_grade.shape)
print(X_test_grade.shape)
```

```
(73196, 4)
(36052, 4)
```

1.5 Teacher Prefix

In [0]:

In [0]:

```
vectorizer_5 = CountVectorizer(list(X_train['teacher_prefix'].values), lowercase=False,
binary=True)
```

In [0]:

```
X_train_prefix = vectorizer_5.fit_transform(X_train['teacher_prefix'].values)
X_test_prefix = vectorizer_5.transform(X_test['teacher_prefix'].values)
```

In [25]:

```
print(X_train_prefix.shape)
print(X_test_prefix.shape)
```

```
(73196, 5)
(36052, 5)
```

1.6 Essay

1.6.1 BOW

In [0]:

```
vectorizer_6 = CountVectorizer(list(X_train['essay'].values), min_df=10,
max_features=5000,ngram_range=(2,2))
```

In [0]:

```
#We are considering the words which occur in atleat 10 documents and max_features=5000 because we
need only those important 5000 words
X_train_essay_bow = vectorizer_6.fit_transform(X_train['essay'].values)
X_test_essay_bow = vectorizer_6.transform(X_test['essay'].values)
```

In [28]:

```
print(X_train_essay_bow.shape)
print(X_test_essay_bow.shape)
```

```
(73196, 5000)
(36052, 5000)
```

1.6.2 TFIDF

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_7 = TfidfVectorizer(list(X_train['essay'].values), min_df=10, max_features=5000,
ngram_range=(2,2))
```

In [0]:

```
X_train_essay_tfidf = vectorizer_7.fit_transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer_7.transform(X_test['essay'].values)
```

In [31]:

```
print(X_train_essay_tfidf.shape)
print(X_test_essay_tfidf.shape)
```

```
(73196, 5000)
(36052, 5000)
```

1.6.3 AvgW2V

In [0]:

```
#Unpickling
import pickle
with open('/content/drive/My Drive/Applied AI/Datasets/New Donors/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words = set(model.keys())
```

In [33]:

```
X_train_essay_avg_w2v = []
X_test_essay_avg_w2v = []

for i in tqdm(X_train['essay']):
    vector = np.zeros(300)
    cnt_words = 0

    for word in i.split():
        if word in glove_words:
            vector += model[word]
            cnt_words +=1

    if cnt_words !=0:
            vector /= cnt_words

    X_train_essay_avg_w2v.append(vector)

for i in tqdm(X_test['essay']):
    vector = np.zeros(300)
    cnt_words = 0
    for word in i.split():
        if word in glove_words :
            vector += model[word]
            cnt_words += 1

    if cnt_words != 0:
        vector /= cnt_words
    X_test_essay_avg_w2v.append(vector)
```

```
100%|██████████| 73196/73196 [00:22<00:00, 3299.28it/s]
100%|██████████| 36052/36052 [00:11<00:00, 3228.37it/s]
```

In [34]:

```
print(len(X_train_essay_avg_w2v))
print(len(X_train_essay_avg_w2v[0]))
print('='*50)
print(len(X_test_essay_avg_w2v))
print(len(X_test_essay_avg_w2v[0]))
```

```
73196
300
==================================================
36052
300
```

1.6.4 TFIDF W2V

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
Tf_idf_model = TfidfVectorizer()
Tf_idf_model.fit(X_train['essay'])

dictionary = dict(zip(Tf_idf_model.get_feature_names(), Tf_idf_model.idf_))
tf_idf_words = set(Tf_idf_model.get_feature_names())
```

In [36]:

```
# TFIDF Word2Vec
# compute TFIDF word2vec for each review.
X_train_essay_tfidf_w2v = []; # 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
    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 tf_idf_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.append(vector)


X_test_essay_tfidf_w2v = []; # 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 tf_idf_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.append(vector)
```

```
100%|██████████| 73196/73196 [02:35<00:00, 472.09it/s]
100%|██████████| 36052/36052 [01:14<00:00, 467.14it/s]
```

In [37]:

```
print(len(X_train_essay_tfidf_w2v))
print(len(X_test_essay_tfidf_w2v))
```

```
73196
36052
```

1.7 Project Title

1.7.1 BOW

In [0]:

```
vectorizer_8 = CountVectorizer(list(X_train['title'].values), min_df=10)
```

In [0]:

```
X_train_title_bow = vectorizer_8.fit_transform(X_train['title'].values)
X_test_title_bow = vectorizer_8.transform(X_test['title'].values)
```

In [40]:

```
print(X_train_title_bow.shape)
print(X_test_title_bow.shape)
```

```
(73196, 2617)
```

```
(36052, 2617)
```

1.7.2 TFIDF

In [0]:

```
vectorizer_9 = TfidfVectorizer(list(X_train['title'].values), min_df=10)
```

In [0]:

```
X_train_title_tfidf = vectorizer_9.fit_transform(X_train['title'].values)
X_test_title_tfidf = vectorizer_9.transform(X_test['title'].values)
```

In [43]:

```
print(X_train_title_tfidf.shape)
print(X_test_title_tfidf.shape)
```

```
(73196, 2617)
(36052, 2617)
```

1.7.3 AvgW2V

In [44]:

```
#Avgw2V vector for preprcessed essay

X_train_title_avg_w2v = []
X_test_title_avg_w2v = []

for i in tqdm(X_train['title']):
    vector = np.zeros(300)
    cnt_words = 0

    for word in i.split():
        if word in glove_words:
            vector += model[word]
            cnt_words +=1

    if cnt_words !=0:
            vector /= cnt_words

    X_train_title_avg_w2v.append(vector)

for i in tqdm(X_test['title']):
    vector = np.zeros(300)
    cnt_words = 0

    for word in i.split():
        if word in glove_words:
            vector += model[word]
            cnt_words +=1

    if cnt_words !=0:
            vector /= cnt_words

    X_test_title_avg_w2v.append(vector)
```

```
100%|██████████| 73196/73196 [00:01<00:00, 58663.26it/s]
100%|██████████| 36052/36052 [00:00<00:00, 58402.45it/s]
```

In [45]:

```
print(len(X_train_title_avg_w2v))
print(len(X_train_title_avg_w2v[0]))
print('='*50)
print(len(X_test_title_avg_w2v))
print(len(X_test_title_avg_w2v[0]))
```

```
73196
300
==================================================
36052
300
```

1.7.4 TFIDFW2V

In [0]:

```
#Training
Tf_idf_model = TfidfVectorizer()
Tf_idf_model.fit(X_train['title'])

dictionary = dict(zip(Tf_idf_model.get_feature_names(), Tf_idf_model.idf_))
tf_idf_words = set(Tf_idf_model.get_feature_names())
```

In [47]:

```
# TFIDF Word2Vec
# compute TFIDF word2vec for each review.
X_train_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['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 tf_idf_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.append(vector)


X_test_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['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 tf_idf_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.append(vector)
```

```
100%|██████████| 73196/73196 [00:02<00:00, 27822.45it/s]
100%|██████████| 36052/36052 [00:01<00:00, 27338.38it/s]
```

In [48]:

```
print(len(X_train_title_tfidf_w2v))
print(len(X_test_title_tfidf_w2v))
```

```
73196
36052
```

1.8 Price

1.8.1 Price Unstandardised

In [0]:

```
X_train_price_unstandardized = X_train['price'].values.reshape(-1,1)
X_test_price_unstandardized = X_test['price'].values.reshape(-1,1)
```

In [50]:

```
print(X_train_price_unstandardized.shape)
print(X_test_price_unstandardized.shape)
```

```
(73196, 1)
(36052, 1)
```

1.8.2 Price Standardized

In [0]:

```
from sklearn.preprocessing import StandardScaler
sc_price = StandardScaler()
X_train_price = sc_price.fit_transform(X_train['price'].values.reshape(-1,1))
X_test_price = sc_price.transform(X_test['price'].values.reshape(-1,1))
```

In [52]:

```
print(X_train_price.shape)
print(X_test_price.shape)
```

```
(73196, 1)
(36052, 1)
```

1.9 Previously posted Projects

1.9.1 Unstandardized

In [0]:

```
X_train_previous_unstandardized = X_train['teacher_number_of_previously_posted_projects'].values.r
eshape(-1,1)
X_test_previous_unstandardized =
X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
```

In [54]:

```
print(X_train_previous_unstandardized.shape)
print(X_test_previous_unstandardized.shape)
```

```
(73196, 1)
(36052, 1)
```

1.9.2 Standardized

In [0]:

```
from sklearn.preprocessing import StandardScaler
sc_previous = StandardScaler()
X_train_previous =
sc_previous.fit_transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-
1,1))
X_test_previous =
sc_previous.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
```

In [57]:

```
print(X_train_previous.shape)
print(X_test_previous.shape)
```

```
(73196, 1)
(36052, 1)
```

1.10 Quantity

1.10.1 Unstandardized

In [0]:

```
X_train_quantity_unstandardized = X_train['quantity'].values.reshape(-1,1)
X_test_quantity_unstandardized = X_test['quantity'].values.reshape(-1,1)
```

In [59]:

```
print(X_train_quantity_unstandardized.shape)
print(X_test_quantity_unstandardized.shape)
```

```
(73196, 1)
(36052, 1)
```

1.10.2 Standardized

In [0]:

```
sc_quantity = StandardScaler()
X_train_quantity = sc_quantity.fit_transform(X_train['quantity'].values.reshape(-1,1))
X_test_quantity = sc_quantity.transform(X_test['quantity'].values.reshape(-1,1))
```

In [61]:

```
print(X_train_quantity.shape)
print(X_test_quantity.shape)
```

```
(73196, 1)
(36052, 1)
```

1.11 Sentiment Score

In [0]:

```
X_train_sentiment = X_train['sentiment_score'].values.reshape(-1,1)
X_test_sentiment = X_test['sentiment_score'].values.reshape(-1,1)
```

In [63]:

```
print(X_train_sentiment.shape)
print(X_test_sentiment.shape)
```

```
(73196, 1)
(36052, 1)
```

1.12 Number of Words in the title

1.12.1 Unstandardized

In [0]:

```
X_train_words_title_unstandardized = X_train['No of words in project title'].values.reshape(-1,1)
X_test_words_title_unstandardized = X_test['No of words in project title'].values.reshape(-1,1)
```

In [65]:

```
print(X_train_words_title_unstandardized.shape)
print(X_test_words_title_unstandardized.shape)
```

```
(73196, 1)
(36052, 1)
```

1.12.2 Standardized

In [0]:

```
sc_title_words = StandardScaler()
X_train_words_title = sc_title_words.fit_transform(X_train['No of words in project title'].values.r
eshape(-1,1))
X_test_words_title = sc_title_words.transform(X_test['No of words in project title'].values.reshape
(-1,1))
```

In [68]:

```
print(X_train_words_title.shape)
print(X_test_words_title.shape)
```

```
(73196, 1)
(36052, 1)
```

1.13 Number of Words in the essay

1.13.1 Unstandardized

In [0]:

```
X_train_words_essay_unstandardized = X_train['No of words in essay'].values.reshape(-1,1)
X_test_words_essay_unstandardized = X_test['No of words in essay'].values.reshape(-1,1)
```

In [0]:

```
print(X_train_words_essay_unstandardized.shape)
print(X_test_words_essay_unstandardized.shape)
```

```
(73196, 1)
(36052, 1)
```

1.13.2 Standardized

In [0]:

```
sc_essay_words = StandardScaler()
X_train_words_essay = sc_essay_words.fit_transform( X_train['No of words in essay'].values.reshape(
-1,1))
X_test_words_essay = sc_essay_words.transform(X_test['No of words in essay'].values.reshape(-1,1))
```

In [70]:

```
print(X_train_words_essay.shape)
print(X_test_words_essay.shape)
```

```
print(X_test_words_essay.shape)
```

```
(73196, 1)
(36052, 1)
```

In [71]:

```
print(X_train_Sstate.shape)
print(X_train_cat.shape)
print(X_train_subcat.shape)
print(X_train_grade.shape)
print(X_train_prefix.shape)
print(X_train_price.shape)
print(X_train_previous.shape)

print(X_train_essay_bow.shape)
print(X_train_essay_tfidf.shape)
print(len(X_train_essay_avg_w2v))
print(len(X_train_essay_tfidf_w2v))

print(X_train_title_bow.shape)
print(X_train_title_tfidf.shape)
print(len(X_train_title_avg_w2v))

print(X_train_quantity.shape)
print(X_train_sentiment.shape)
print(X_train_words_essay.shape)
print(X_train_words_title.shape)

print('='*50)
print(X_test_Sstate.shape)
print(X_test_cat.shape)
print(X_test_subcat.shape)
print(X_test_grade.shape)
print(X_test_prefix.shape)
print(X_test_price.shape)
print(X_test_previous.shape)

print(X_test_essay_bow.shape)
print(X_test_essay_tfidf.shape)
print(len(X_test_essay_avg_w2v))
print(len(X_test_essay_tfidf_w2v))

print(X_test_title_bow.shape)
print(X_test_title_tfidf.shape)
print(len(X_test_title_avg_w2v))
print(len(X_test_title_tfidf_w2v))

print(X_test_quantity.shape)
print(X_test_sentiment.shape)
print(X_test_words_essay.shape)
print(X_test_words_title.shape)
```

```
(73196, 51)
(73196, 9)
(73196, 30)
(73196, 4)
(73196, 5)
(73196, 1)
(73196, 1)
(73196, 5000)
(73196, 5000)
73196
73196
(73196, 2617)
(73196, 2617)
73196
(73196, 1)
(73196, 1)
(73196, 1)
(73196, 1)
==================================================
(36052, 51)
(36052, 9)
(36052, 30)
```

```
(36052, 4)
(36052, 5)
(36052, 1)
(36052, 1)
(36052, 5000)
(36052, 5000)
36052
36052
(36052, 2617)
(36052, 2617)
36052
36052
(36052, 1)
(36052, 1)
(36052, 1)
(36052, 1)
```

2. Set - 1

2.1 Merging the above features required for Set-1

In [0]:

```
from scipy.sparse import hstack
X_train_1 = hstack((X_train_Sstate, X_train_cat, X_train_subcat, X_train_grade, X_train_prefix, X_train_essay_bow, X_train_title_bow, X_train_previous, X_train_price)).tocsr()
X_test_1 = hstack((X_test_Sstate, X_test_cat, X_test_subcat, X_test_grade, X_test_prefix, X_test_essay_bow, X_test_title_bow, X_test_previous, X_test_price)).tocsr()
```

In [75]:

```
print(X_train_1.shape)
print(X_test_1.shape)
```

```
(73196, 7718)
(36052, 7718)
```

2.2 Grid Search CV

In [0]:

```
from sklearn.linear_model import LogisticRegression
classifier_1 = LogisticRegression()
```

In [0]:

```
from sklearn.model_selection import GridSearchCV
parameters = [
                {
                    'C' : [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000],
                    'penalty' : ['l1', 'l2']
                }
            ]
gridsearch_1 = GridSearchCV(estimator=classifier_1, param_grid=parameters, scoring='roc_auc', cv=2, n_jobs=-1, return_train_score=True)
```

In [0]:

```
gridsearch_1 = gridsearch_1.fit(X_train_1, y_train)
```

In [82]:

```
results = pd.DataFrame.from_dict(gridsearch_1.cv_results_)
results = results.sort_values(['param_C'])
results.head()
```

Out[82]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_penalty	params	split0_test_score	split1_test_score
0	0.028398	0.000469	0.000000	0.000000	1e-05	l1	{'C': 1e-05, 'penalty': 'l1'}	NaN	NaN
1	0.297417	0.006274	0.032599	0.004696	1e-05	l2	{'C': 1e-05, 'penalty': 'l2'}	0.635354	0.64212
2	0.022233	0.001241	0.000000	0.000000	0.0001	l1	{'C': 0.0001, 'penalty': 'l1'}	NaN	NaN
3	0.329449	0.010884	0.032196	0.006696	0.0001	l2	{'C': 0.0001, 'penalty': 'l2'}	0.645100	0.65041
4	0.021930	0.001393	0.000000	0.000000	0.001	l1	{'C': 0.001, 'penalty': 'l1'}	NaN	NaN

In [83]:

```
best_C_1 = gridsearch_1.best_params_['C']
best_penalty_1 = gridsearch_1.best_params_['penalty']
print('Best C:', best_C_1)
print('Best Penalty:', best_penalty_1)
```

```
Best C: 0.01
Best Penalty: l2
```

Note:

- We can see that the best C is 0.01 and Regulariser is 'L2'

2.3 AUC vs Hyperparameter

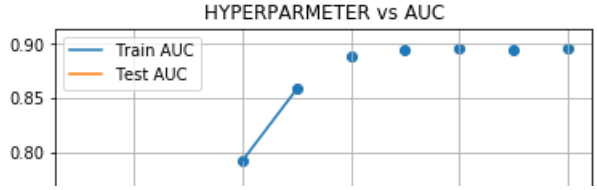
In [0]:

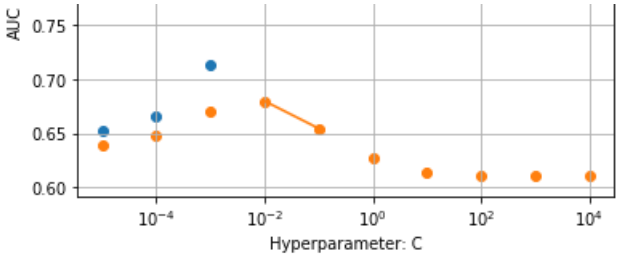
```
train_auc_1 = results['mean_train_score']
test_auc_1 = results['mean_test_score']
C_1 = results['param_C']
```

In [91]:

```
plt.plot(C_1, train_auc_1, label='Train AUC')
plt.plot(C_1, test_auc_1, label='Test AUC')
plt.scatter(C_1, train_auc_1)
plt.scatter(C_1, test_auc_1)

plt.title('HYPERPARMETER vs AUC')
plt.xlabel('Hyperparameter: C')
plt.ylabel('AUC')
plt.xscale('log')
plt.legend()
plt.grid()
plt.show()
```





2.4 Modelling with best C and penalty to find best AUC

In [92]:

```
classifier_withParam_1 = LogisticRegression(penalty=best_penalty_1, C=best_C_1)
classifier_withParam_1.fit(X_train_1, y_train)
```

Out[92]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

2.5 Cross Validation

In [0]:

```
from sklearn.model_selection import cross_val_score
cv_1 = cross_val_score(estimator=classifier_withParam_1, X=X_train_1, y=y_train, cv=5, scoring='roc
_auc')
```

In [94]:

```
best_auc_1 = cv_1.mean()
print('Best AUC: %4f' %best_auc_1)
```

```
Best AUC: 0.688205
```

2.6 ROC curve on train and test data

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

```
y_train_pred_1 = batch_predict(classifier_withParam_1, X_train_1)
y_test_pred_1 = batch_predict(classifier_withParam_1, X_test_1)
```

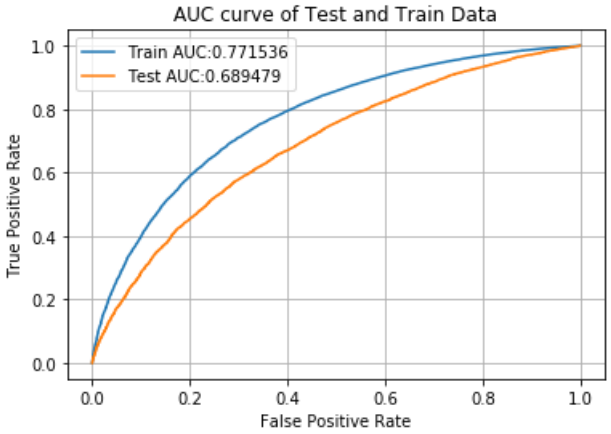
In [0]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_1, train_tpr_1, train_thresh_1 = roc_curve(y_train, y_train_pred_1)
test_fpr_1, test_tpr_1, test_thresh_1 = roc_curve(y_test, y_test_pred_1)
```

In [98]:

```
plt.plot(train_fpr_1, train_tpr_1, label='Train AUC:%4f'%auc(train_fpr_1, train_tpr_1))
plt.plot(test_fpr_1, test_tpr_1, label='Test AUC:%4f'%auc(test_fpr_1, test_tpr_1))

plt.title('AUC curve of Test and Train Data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid()
plt.show()
```



2.7 Finding Confusion Matrix

In [0]:

```
def find_best_threshold( fpr, tpr, threshold):
    t = threshold[np.argmax(tpr*(1-fpr))]
    print('the maximum tpr*(1-fpr) is :', max(tpr*(1-fpr)), 'for threshold', np.round(t,3))
    return t
```

In [100]:

```
best_t = find_best_threshold(train_fpr_1, train_tpr_1, train_thresh_1)
```

```
the maximum tpr*(1-fpr) is : 0.4977922006408444 for threshold 0.833
```

In [0]:

```
def predict_with_threshold(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)

    return predictions
```

In [0]:

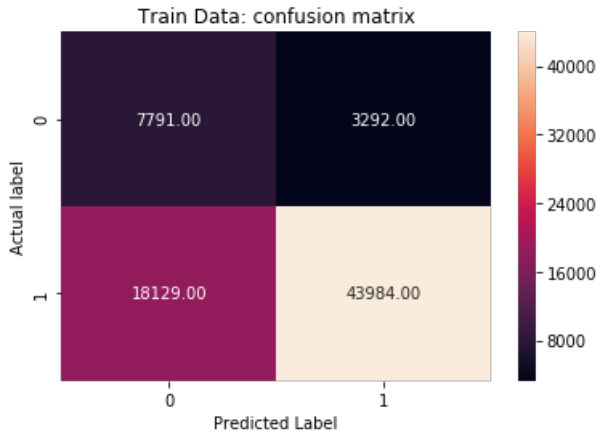
```
from sklearn.metrics import confusion_matrix

cm_train_1 = confusion_matrix(y_train, predict_with_threshold(y_train_pred_1, best_t))
cm_test_1 = confusion_matrix(y_test, predict_with_threshold(y_test_pred_1, best_t))
```

In [103]:

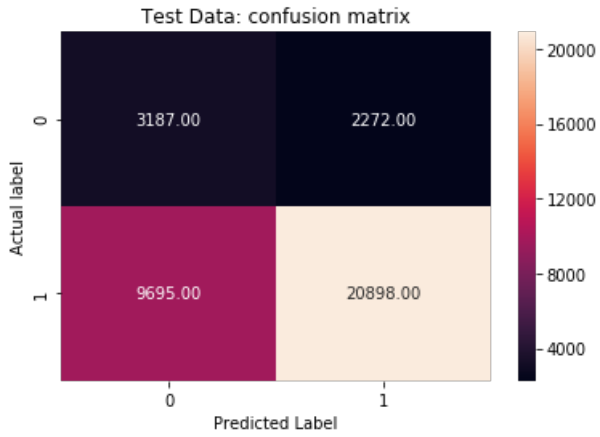
```
sns.heatmap(cm_train_1, annot=True, fmt='.2f')
plt.title('Train Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
```

```
plt.show()
```



In [104]:

```
sns.heatmap(cm_test_1, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



3. Set-2:

In [0]:

```
from scipy.sparse import hstack
X_train_2 = hstack((X_train_Sstate, X_train_cat, X_train_subcat, X_train_grade, X_train_prefix, X_train_essay_tfidf, X_train_title_tfidf, X_train_previous, X_train_price)).tocsr()
X_test_2 = hstack((X_test_Sstate, X_test_cat, X_test_subcat, X_test_grade, X_test_prefix, X_test_essay_tfidf, X_test_title_tfidf, X_test_previous, X_test_price)).tocsr()
```

In [106]:

```
print(X_train_2.shape)
print(X_test_2.shape)
```

```
(73196, 7718)
(36052, 7718)
```

In [107]:

```
print(y_train.shape)
print(y_test.shape)
```

```
(73196, 1)
```

(36052, 1)

2.2.2 Grid Search

In [0]:

```
classifier_2 = LogisticRegression()
```

In [0]:

```
from sklearn.model_selection import GridSearchCV
parameters = [
                {
                    'C' : [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000],
                    'penalty' : ['l1', 'l2']
                }
            ]
gridsearch_2 = GridSearchCV(estimator=classifier_2, param_grid=parameters, scoring='roc_auc', cv=2
, n_jobs=-1, return_train_score=True)
```

In [0]:

```
gridsearch_2 = gridsearch_2.fit(X_train_2 ,y_train)
```

In [112]:

```
best_C_2 = gridsearch_2.best_params_['C']
best_penalty_2 = gridsearch_2.best_params_['penalty']
print('Best C :', best_C_2)
print('Best Penalty:', best_penalty_2)
```

```
Best C : 0.1
Best Penalty: l2
```

Summary:

- it shows that the best alpha= 0.1 and best regularizer = 'l2'

In [113]:

```
results = pd.DataFrame.from_dict(gridsearch_2.cv_results_)
results = results.sort_values(['param_C'])
results.head()
```

Out[113]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_penalty	params	split0_test_score	split1_test_score
0	0.032552	0.002066	0.000000	0.000000	1e-05	l1	{'C': 1e-05, 'penalty': 'l1'}	NaN	NaN
1	0.321316	0.000256	0.027203	0.000969	1e-05	l2	{'C': 1e-05, 'penalty': 'l2'}	0.614768	0.615486
2	0.023181	0.001165	0.000000	0.000000	0.0001	l1	{'C': 0.0001, 'penalty': 'l1'}	NaN	NaN
3	0.270342	0.001667	0.027768	0.000972	0.0001	l2	{'C': 0.0001, 'penalty': 'l2'}	0.615178	0.615050
4	0.022770	0.000911	0.000000	0.000000	0.001	l1	{'C': 0.001,	NaN	NaN

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_penalty	params	split0_test_score	split1_test_score
4	0.022770	0.000911	0.000000	0.000000	0.001	l1	{'penalty': 'l1'}	NaN	NaN

2.2.3 AUC vs Hyperparameter

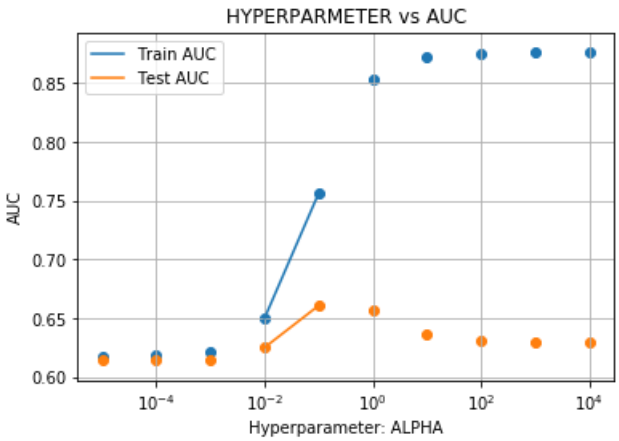
In [0]:

```
train_auc_2 = results['mean_train_score']
test_auc_2 = results['mean_test_score']
C_2 = results['param_C']
```

In [116]:

```
plt.plot(C_2, train_auc_2, label='Train AUC')
plt.plot(C_2, test_auc_2, label='Test AUC')
plt.scatter(C_2, train_auc_2)
plt.scatter(C_2, test_auc_2)

plt.title('HYPERPARMETER vs AUC')
plt.xlabel('Hyperparameter: ALPHA')
plt.ylabel('AUC')
plt.xscale('log')
plt.legend()
plt.grid()
plt.show()
```



2.2.3 Modelling with Parameters

In [117]:

```
classifier_withParam_2 = LogisticRegression(penalty=best_penalty_2, C=best_C_2)
classifier_withParam_2.fit(X_train_2, y_train)
```

Out[117]:

```
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

2.2.4 Cross Validation

In [0]:

```
from sklearn.model_selection import cross_val_score
cv_2 = cross_val_score(estimator=classifier_withParam_2, X=X_train_2, y=y_train,cv=5, scoring='roc_
auc')
```

In [119]:

```
best_auc_2 = cv_1.mean()
print('Best AUC:%4f' %best_auc_2)
```

```
Best AUC:0.688205
```

2.2.5 ROC curve on Train and Test Data

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

```
y_train_pred_2 = batch_predict(classifier_withParam_2, X_train_2)
y_test_pred_2 = batch_predict(classifier_withParam_2, X_test_2)
```

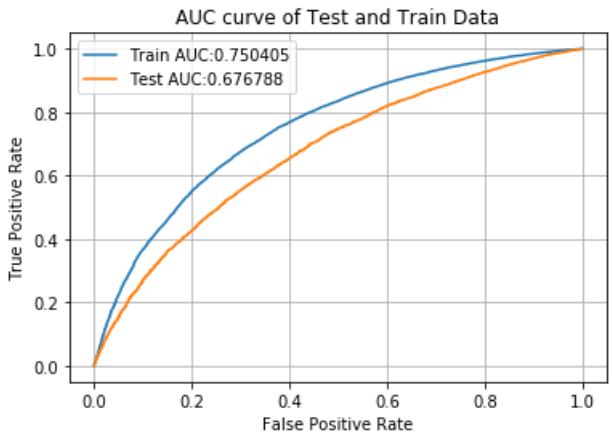
In [0]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_2, train_tpr_2, train_thresh_2 = roc_curve(y_train, y_train_pred_2)
test_fpr_2, test_tpr_2, test_thresh_2 = roc_curve(y_test, y_test_pred_2)
```

In [123]:

```
plt.plot(train_fpr_2, train_tpr_2, label='Train AUC:%4f'%auc(train_fpr_2, train_tpr_2))
plt.plot(test_fpr_2, test_tpr_2, label='Test AUC:%4f'%auc(test_fpr_2, test_tpr_2))

plt.title('AUC curve of Test and Train Data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid()
plt.show()
```



2.2.6 Confusion Matrix

In [0]:

```
def find_best_threshold(fpr, tpr, threshold):
    t = threshold[np.argmax(tpr*(1-fpr))]
    print('the maximum tpr*(1-fpr) is :', max(tpr*(1-fpr)), 'for threshold', np.round(t,3))
```

```
    return t
```

In [125]:

```
best_t = find_best_threshold(train_fpr_2, train_tpr_2, train_thresh_2)
```

```
the maximum tpr*(1-fpr) is : 0.47363944330264035 for threshold 0.837
```

In [0]:

```
def predict_with_threshold(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)

    return predictions
```

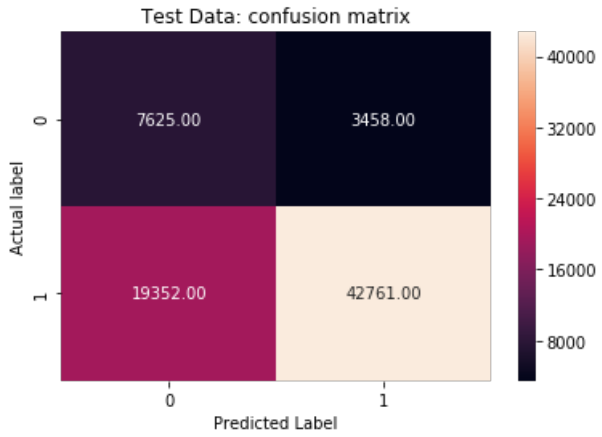
In [0]:

```
from sklearn.metrics import confusion_matrix

cm_train_2 = confusion_matrix(y_train, predict_with_threshold(y_train_pred_2, best_t))
cm_test_2 = confusion_matrix(y_test, predict_with_threshold(y_test_pred_2, best_t))
```

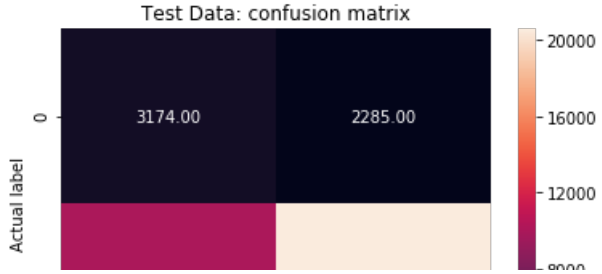
In [128]:

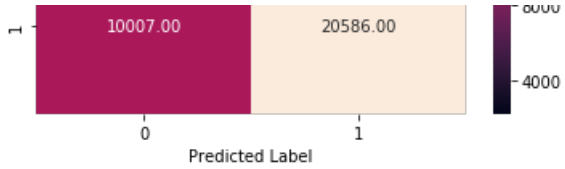
```
sns.heatmap(cm_train_2, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [129]:

```
sns.heatmap(cm_test_2, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```





3.1 Set - 3

In [0]:

```
from scipy.sparse import hstack
X_train_3 = hstack((X_train_Sstate, X_train_cat, X_train_subcat, X_train_grade, X_train_prefix, X_t
rain_previous, X_train_price, X_train_essay_avg_w2v, X_train_title_avg_w2v)).tocsr()
X_test_3 = hstack((X_test_Sstate, X_test_cat, X_test_subcat, X_test_grade, X_test_prefix, X_test_pr
evious, X_test_price, X_test_essay_avg_w2v, X_test_title_avg_w2v)).tocsr()
```

In [131]:

```
print(X_train_3.shape)
print(X_test_3.shape)
```

```
(73196, 701)
(36052, 701)
```

In [132]:

```
print(y_train.shape)
print(y_test.shape)
```

```
(73196, 1)
(36052, 1)
```

3.1.3 Grid Search

In [0]:

```
classifier_3 = LogisticRegression()
```

In [0]:

```
from sklearn.model_selection import GridSearchCV
parameters = [
                {
                    'C' : [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000],
                    'penalty' : ['l1' , 'l2']
                }
            ]
gridsearch_3 = GridSearchCV(estimator=classifier_3, param_grid=parameters, scoring='roc_auc', cv=2
, n_jobs=-1, return_train_score=True)
```

In [0]:

```
gridsearch_3 = gridsearch_3.fit(X_train_3 ,y_train)
```

In [138]:

```
best_C_3 = gridsearch_3.best_params_['C']
best_penalty_3 = gridsearch_3.best_params_['penalty']
print('Best C :', best_C_3)
print('Best Penalty:', best_penalty_3)
```

```
Best C : 10
Best Penalty: l2
```

Summary:

- it shows that the best C = 10 and best regularizer is 'l2'.

3.1.4 HyperParameter vs AUC

In [139]:

```
results = pd.DataFrame.from_dict(gridsearch_3.cv_results_)
results = results.sort_values(['param_C'])
results.head()
```

Out[139]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_penalty	params	split0_test_score	split1_test_score
0	0.396038	0.000789	0.000000	0.000000	1e-05	l1	{'C': 1e-05, 'penalty': 'l1'}	NaN	NaN
1	2.390292	0.005105	0.110998	0.000490	1e-05	l2	{'C': 1e-05, 'penalty': 'l2'}	0.621127	0.61934
2	0.287748	0.004986	0.000000	0.000000	0.0001	l1	{'C': 0.0001, 'penalty': 'l1'}	NaN	NaN
3	2.800356	0.076358	0.095690	0.001003	0.0001	l2	{'C': 0.0001, 'penalty': 'l2'}	0.624016	0.62084
4	0.280813	0.000818	0.000000	0.000000	0.001	l1	{'C': 0.001, 'penalty': 'l1'}	NaN	NaN

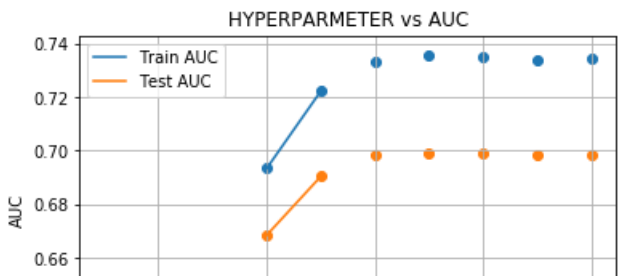
In [0]:

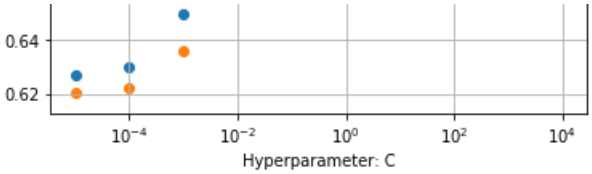
```
train_auc_3 = results['mean_train_score']
test_auc_3 = results['mean_test_score']
C_3 = results['param_C']
```

In [141]:

```
plt.plot(C_3, train_auc_3, label='Train AUC')
plt.plot(C_3, test_auc_3, label='Test AUC')
plt.scatter(C_3, train_auc_3)
plt.scatter(C_3, test_auc_3)

plt.title('HYPERPARMETER vs AUC')
plt.xlabel('Hyperparameter: C')
plt.ylabel('AUC')
plt.xscale('log')
plt.legend()
plt.grid()
plt.show()
```





3.1.5 Modelling with Parameters

In [142]:

```
classifier_withParam_3 = LogisticRegression(penalty=best_penalty_3, C=best_C_3)
classifier_withParam_3.fit(X_train_3, y_train)
```

Out[142]:

```
LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

3.1.6 Cross Validation

In [0]:

```
from sklearn.model_selection import cross_val_score
cv_3 = cross_val_score(estimator=classifier_withParam_3, X=X_train_3, y=y_train,cv=5, scoring='roc_
auc')
```

In [144]:

```
best_auc_3 = cv_3.mean()
print('Best AUC:%4f' %best_auc_3)
```

```
Best AUC:0.702954
```

3.1.6 ROC curve on Train and Test Data

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

```
y_train_pred_3 = batch_predict(classifier_withParam_3, X_train_3)
y_test_pred_3 = batch_predict(classifier_withParam_3, X_test_3)
```

In [0]:

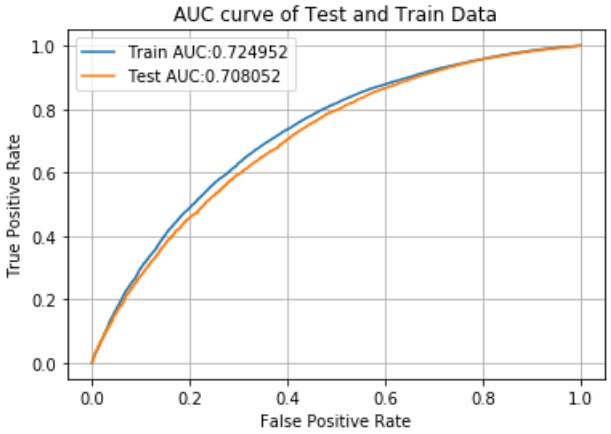
```
from sklearn.metrics import roc_curve, auc
train_fpr_3, train_tpr_3, train_thresh_3 = roc_curve(y_train, y_train_pred_3)
test_fpr_3, test_tpr_3, test_thresh_3 = roc_curve(y_test, y_test_pred_3)
```

In [148]:

```
plt.plot(train_fpr_3, train_tpr_3, label='Train AUC:%4f'%auc(train_fpr_3, train_tpr_3))
```

```
plt.plot(test_fpr_3, test_tpr_3, label='Test AUC:%4f'%auc(test_fpr_3, test_tpr_3))

plt.title('AUC curve of Test and Train Data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid()
plt.show()
```



3.1.7 Confusion Matrix

In [0]:

```
def find_best_threshold(fpr, tpr, threshold):
    t = threshold[np.argmax(tpr*(1-fpr))]
    print('the maximum tpr*(1-fpr) is :', max(tpr*(1-fpr)), 'for threshold', np.round(t,3))
    return t
```

In [150]:

```
best_t = find_best_threshold(train_fpr_3, train_tpr_3, train_thresh_3)
```

```
the maximum tpr*(1-fpr) is : 0.4473110591098588 for threshold 0.844
```

In [0]:

```
def predict_with_threshold(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)

    return predictions
```

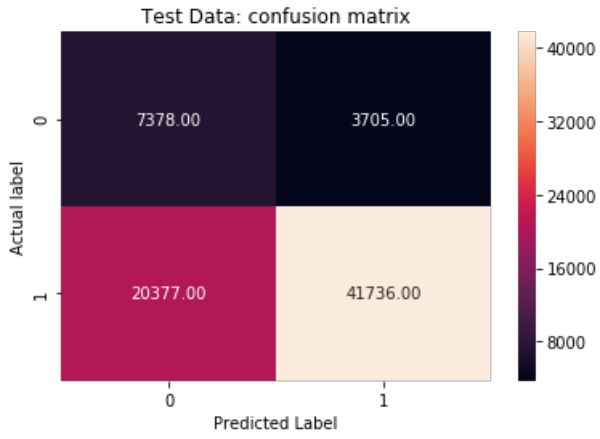
In [0]:

```
from sklearn.metrics import confusion_matrix

cm_train_3 = confusion_matrix(y_train, predict_with_threshold(y_train_pred_3, best_t))
cm_test_3 = confusion_matrix(y_test, predict_with_threshold(y_test_pred_3, best_t))
```

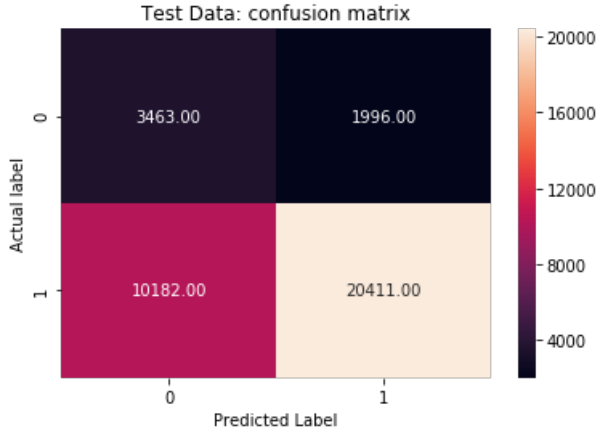
In [153]:

```
sns.heatmap(cm_train_3, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [154]:

```
sns.heatmap(cm_test_3, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



4. Set -4

4.1 Merging all the columns

In [0]:

```
from scipy.sparse import hstack
X_train_4 = hstack((X_train_Sstate, X_train_cat, X_train_subcat, X_train_grade, X_train_prefix, X_t
rain_previous, X_train_price, X_train_essay_tfidf_w2v, X_train_title_tfidf_w2v)).tocsr()
X_test_4 = hstack((X_test_Sstate, X_test_cat, X_test_subcat, X_test_grade, X_test_prefix, X_test_pr
evious, X_test_price, X_test_essay_tfidf_w2v, X_test_title_tfidf_w2v)).tocsr()
```

In [156]:

```
print(X_train_4.shape)
print(X_test_4.shape)
```

```
(73196, 701)
(36052, 701)
```

4.2 Grid Search CV

In [0]:

```
classifier_4 = LogisticRegression()
```

In [0]:

```
parameters = [
                {
                    'C': [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000],
                    'penalty' : ['l1', 'l2']
                }
            ]
grid_search_4 = GridSearchCV(estimator=classifier_4, param_grid=parameters, scoring='roc_auc', cv=
2, n_jobs=-1, return_train_score=True)
```

In [0]:

```
grid_search_4 = grid_search_4.fit(X=X_train_4, y=y_train)
```

In [161]:

```
best_C_4 = grid_search_4.best_params_['C']
best_penalty_4 = grid_search_4.best_params_['penalty']
print('Best C:', best_C_4)
print('Best Regularizer:', best_penalty_4)
```

```
Best C: 1
Best Regularizer: 12
```

4.3 Hyperparameter vs AUC

In [163]:

```
results = pd.DataFrame.from_dict(grid_search_4.cv_results_)
results = results.sort_values(['param_C'])
results.head()
```

Out[163]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_penalty	params	split0_test_score	split1_test_score
0	0.369625	0.020564	0.000000	0.000000	1e-05	l1	{'C': 1e-05, 'penalty': 'l1'}	NaN	NaN
1	2.462661	0.070438	0.099510	0.000067	1e-05	l2	{'C': 1e-05, 'penalty': 'l2'}	0.635116	0.63247
2	0.226950	0.004219	0.000000	0.000000	0.0001	l1	{'C': 0.0001, 'penalty': 'l1'}	NaN	NaN
3	2.590081	0.353885	0.115953	0.020777	0.0001	l2	{'C': 0.0001, 'penalty': 'l2'}	0.641217	0.63688
4	0.212311	0.008434	0.000000	0.000000	0.001	l1	{'C': 0.001, 'penalty': 'l1'}	NaN	NaN

In [0]:

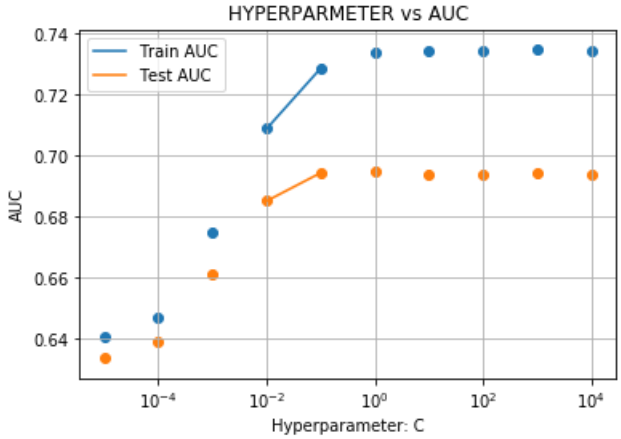
```
train_auc_4 = results['mean_train_score']
test_auc_4 = results['mean_test_score']
C_4 = results['param_C']
```

In [165]:

```
plt.plot(C_4, train_auc_4, label='Train AUC')
```

```
plt.plot(C_4, test_auc_4, label='Test AUC')
plt.scatter(C_4, train_auc_4)
plt.scatter(C_4, test_auc_4)

plt.title('HYPERPARMETER vs AUC')
plt.xlabel('Hyperparameter: C')
plt.ylabel('AUC')
plt.xscale('log')
plt.legend()
plt.grid()
plt.show()
```



4.4 Modelling with Parameters

In [166]:

```
classifier_withParam_4 = LogisticRegression(penalty=best_penalty_4, C=best_C_4)
classifier_withParam_4.fit(X_train_4, y_train)
```

Out[166]:

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

4.5 Cross validate

In [0]:

```
from sklearn.model_selection import cross_val_score
cv_4 = cross_val_score(estimator=classifier_withParam_4, X=X_train_4, y=y_train,cv=5, scoring='roc_
auc')
```

In [168]:

```
best_auc_4 = cv_4.mean()
print('Best AUC: %4f' %best_auc_4)
```

```
Best AUC: 0.700331
```

4.6 AUC CURVE

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop,1000):
```

```
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
    return y_data_pred
```

In [0]:

```
y_train_pred_4 = batch_predict(classifier_withParam_4, X_train_4)
y_test_pred_4 = batch_predict(classifier_withParam_4, X_test_4)
```

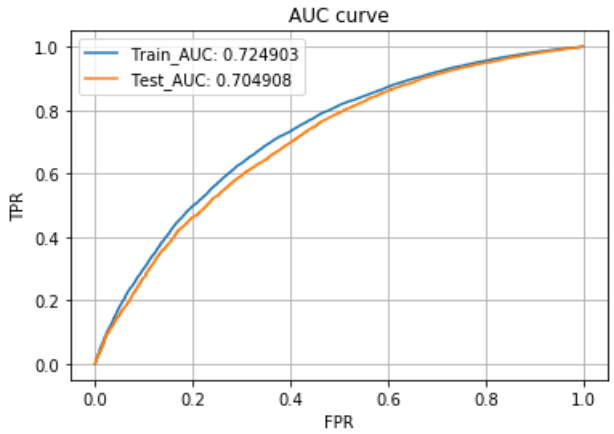
In [0]:

```
train_fpr_4, train_tpr_4, train_threshold_4 = roc_curve(y_train, y_train_pred_4)
test_fpr_4, test_tpr_4, train_threshold_4 = roc_curve(y_test, y_test_pred_4)
```

In [172]:

```
plt.plot(train_fpr_4, train_tpr_4, label='Train_AUC: %4f' %auc(train_fpr_4, train_tpr_4))
plt.plot(test_fpr_4, test_tpr_4, label='Test_AUC: %4f' %auc(test_fpr_4, test_tpr_4))

plt.title('AUC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.grid()
plt.show()
```



4.1.10 Confusion matrix

In [0]:

```
def find_best_threshold( fpr, tpr, threshold):
    t = threshold[np.argmax(tpr*(1-fpr))]
    print('the maximum tpr*(1-fpr) is :', max(tpr*(1-fpr)), 'for threshold', np.round(t,3))
    return t
```

In [174]:

```
best_t = find_best_threshold(train_fpr_4, train_tpr_4, train_threshold_4)
```

```
the maximum tpr*(1-fpr) is : 0.44857823524886603 for threshold 0.695
```

In [0]:

```
def predict_with_threshold(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
```

```
    return predictions
```

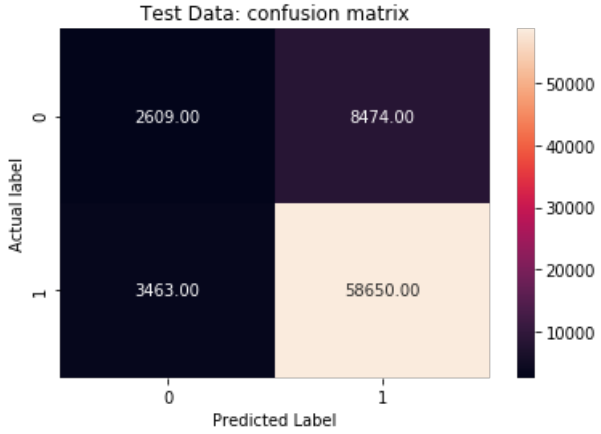
In [0]:

```
from sklearn.metrics import confusion_matrix

cm_train_4 = confusion_matrix(y_train, predict_with_threshold(y_train_pred_4, best_t))
cm_test_4 = confusion_matrix(y_test, predict_with_threshold(y_test_pred_4, best_t))
```

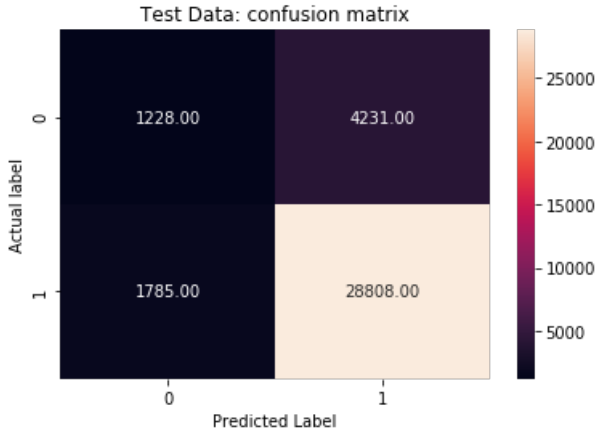
In [177]:

```
sns.heatmap(cm_train_4, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [178]:

```
sns.heatmap(cm_test_4, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



5. Set-5

5.1 Merging all the columns

In [0]:

```
from scipy.sparse import hstack
X_train_5 = hstack((X_train_Sstate, X_train_cat, X_train_subcat, X_train_grade, X_train_prefix, X_t
```

```
rain_previous, X_train_price, X_train_quantity, X_train_words_title, X_train_words_essay, X_train_s
entiment)).tocsr()
X_test_5 = hstack((X_test_Sstate, X_test_cat, X_test_subcat, X_test_grade, X_test_prefix, X_test_pr
evious, X_test_price, X_test_quantity, X_test_words_title, X_test_words_essay, X_test_sentiment)).t
ocsr()
```

In [183]:

```
print(X_train_5.shape)
print(X_test_5.shape)
```

```
(73196, 105)
(36052, 105)
```

5.2 Grid Search CV

In [0]:

```
classifier_5 = LogisticRegression()
```

In [0]:

```
from sklearn.model_selection import GridSearchCV
parameters = [
                {
                    'C': [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000],
                    'penalty' : ['l1', 'l2']
                }
            ]
grid_search_5 = GridSearchCV(estimator=classifier_5, param_grid=parameters, scoring='roc_auc', cv=
2, n_jobs=-1, return_train_score=True)
```

In [0]:

```
grid_search_5 = grid_search_5.fit(X=X_train_5, y=y_train)
```

In [194]:

```
best_C_5 = grid_search_5.best_params_['C']
best_penalty_5 = grid_search_5.best_params_['penalty']
print('Best_C %4f:' %best_C_5)
print('Best Regularizer:', best_penalty_5)
```

```
Best_C 0.010000:
Best Regularizer: l2
```

5.3 Hyperparameter vs AUC

In [190]:

```
results = pd.DataFrame.from_dict(grid_search_5.cv_results_)
results = results.sort_values(['param_C'])
results.head()
```

Out[190]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_penalty	params	split0_test_score	split1_test_score
0	0.014149	0.002166	0.000000	0.000000	1e-05	l1	{'C': 1e-05, 'penalty': 'l1'}	NaN	NaN
1	0.128292	0.000049	0.021569	0.000334	1e-05	l2	{'C': 1e-05, 'penalty': 'l2'}	0.627224	0.623436

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_penalty	params	split0_test_score	split1_test_score
2	0.008750	0.000076	0.000000	0.000000	0.0001	l1	{'C': 0.0001, 'penalty': 'l1'}	NaN	NaN
3	0.107303	0.000799	0.022804	0.000679	0.0001	l2	{'C': 0.0001, 'penalty': 'l2'}	0.627125	0.623905
4	0.009601	0.000405	0.000000	0.000000	0.001	l1	{'C': 0.001, 'penalty': 'l1'}	NaN	NaN

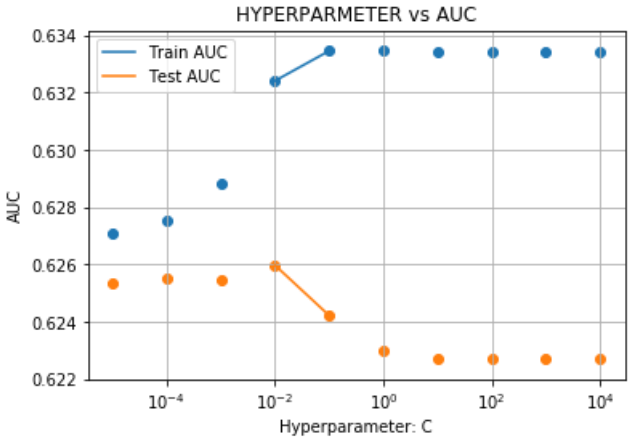
In [0]:

```
train_auc_5 = results['mean_train_score']
test_auc_5 = results['mean_test_score']
C_5 = results['param_C']
```

In [192]:

```
plt.plot(C_5, train_auc_5, label='Train AUC')
plt.plot(C_5, test_auc_5, label='Test AUC')
plt.scatter(C_5, train_auc_5)
plt.scatter(C_5, test_auc_5)

plt.title('HYPERPARMETER vs AUC')
plt.xlabel('Hyperparameter: C')
plt.ylabel('AUC')
plt.xscale('log')
plt.legend()
plt.grid()
plt.show()
```



5.4 Modelling with Parameters

In [195]:

```
classifier_withParam_5 = LogisticRegression(penalty=best_penalty_5, C=best_C_5)
classifier_withParam_5.fit(X_train_5, y_train)
```

Out[195]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

5.5 Cross validate

In [0]:

```
from sklearn.model_selection import cross_val_score
cv_5 = cross_val_score(estimator=classifier_withParam_5, X=X_train_5, y=y_train,cv=10,
scoring='roc_auc')
```

In [197]:

```
best_auc_5 = cv_5.mean()
print('Best AUC:%4f' %best_auc_5)
```

```
Best AUC:0.626290
```

5.6 AUC CURVE

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop,1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
    return y_data_pred
```

In [0]:

```
y_train_pred_5 = batch_predict(classifier_withParam_5, X_train_5)
y_test_pred_5 = batch_predict(classifier_withParam_5, X_test_5)
```

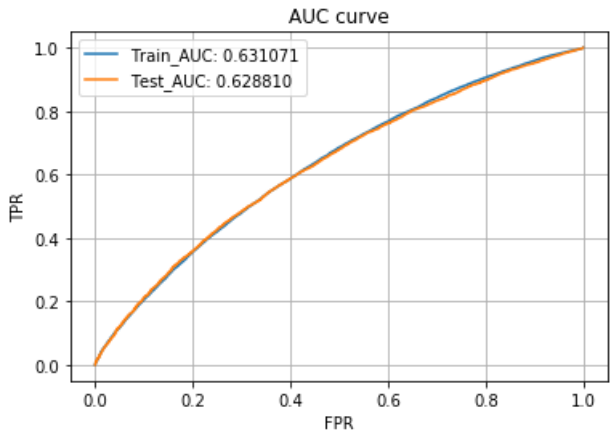
In [0]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_5, train_tpr_5, train_threshold_5 = roc_curve(y_train, y_train_pred_5)
test_fpr_5, test_tpr_5, train_threshold_5 = roc_curve(y_test, y_test_pred_5)
```

In [201]:

```
plt.plot(train_fpr_5, train_tpr_5, label='Train_AUC: %4f' %auc(train_fpr_5, train_tpr_5))
plt.plot(test_fpr_5, test_tpr_5, label='Test_AUC: %4f' %auc(test_fpr_5, test_tpr_5))

plt.title('AUC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.grid()
plt.show()
```



4.1.10 Confusion matrix

In [0]:

In [0]:

```
def find_best_threshold( fpr, tpr, threshold):
    t = threshold[np.argmax(tpr*(1-fpr))]
    print('the maximum tpr*(1-fpr) is :', max(tpr*(1-fpr)), 'for threshold', np.round(t,3))
    return t
```

In [203]:

```
best_t = find_best_threshold(train_fpr_5, train_tpr_5, train_threshold_5)
```

```
the maximum tpr*(1-fpr) is : 0.35281059254208386 for threshold 0.764
```

In [0]:

```
def predict_with_threshold(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)

    return predictions
```

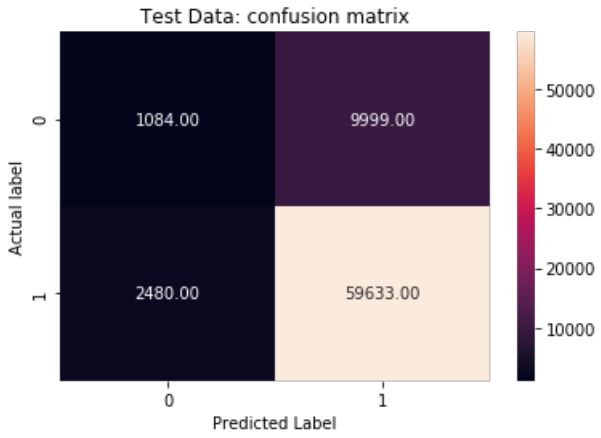
In [0]:

```
from sklearn.metrics import confusion_matrix

cm_train_5 = confusion_matrix(y_train, predict_with_threshold(y_train_pred_5, best_t))
cm_test_5 = confusion_matrix(y_test, predict_with_threshold(y_test_pred_5, best_t))
```

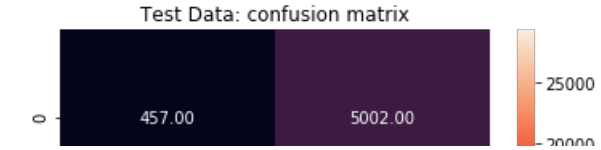
In [206]:

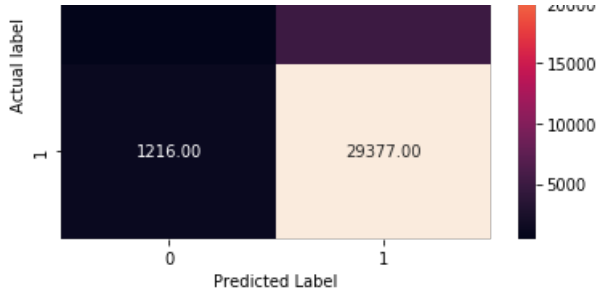
```
sns.heatmap(cm_train_5, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [207]:

```
sns.heatmap(cm_test_5, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```





Summary:

In [211]:

```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ['Set Number', 'Vectorizer', 'Model', 'Best Hyperparameter : C', 'Best Hyperparamet
er : Penalty ', 'Best AUC']
x.add_row(['Set-1', 'BOW', 'Logistic Regression', str(best_C_1), str(best_penalty_1),
str(best_auc_1)])
x.add_row(['Set-2', 'TFIDF', 'Logistic Regression', str(best_C_2), str(best_penalty_2),
str(best_auc_2)])
x.add_row(['Set-3', 'AvgW2V', 'Logistic Regression', str(best_C_3), str(best_penalty_3), str(best_a
uc_3)])
x.add_row(['Set-4', 'TFIDF-W2V', 'Logistic Regression', str(best_C_3), str(best_penalty_4),
str(best_auc_4)])
x.add_row(['Set-5', 'Numerical', 'Logistic Regression', str(best_C_5), str(best_penalty_5), str(bes
t_auc_5)])

print(x)
```

```
+------------+------------+---------------------+-------------------------+-----------------------
-----+--------------------+
| Set Number | Vectorizer |        Model        | Best Hyperparameter : C | Best Hyperparameter : P
enalty  |      Best AUC      |
+------------+------------+---------------------+-------------------------+-----------------------
-----+--------------------+
|   Set-1    |    BOW     | Logistic Regression |           0.01          |               12
| 0.6882048488016882 |
|   Set-2    |   TFIDF    | Logistic Regression |           0.1           |               12
| 0.6882048488016882 |
|   Set-3    |   AvgW2V   | Logistic Regression |            10           |               12
| 0.7029536480659971 |
|   Set-4    | TFIDF-W2V  | Logistic Regression |            10           |               12
| 0.7003306307052629 |
|   Set-5    | Numerical  | Logistic Regression |           0.01          |               12
| 0.6262896722497586 |
+------------+------------+---------------------+-------------------------+-----------------------
-----+--------------------+
```

That's the end of the code