In [23]:

```
""" Title: Sentiment Analysis using Python
Author: SasikalaAlaguvel
Date: 2019.11.20
Availability: https://www.kaggle.com/sasikala11/sentiment-analysis-using-python """
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import plotly.graph_objs as go
import plotly.offline as py
color = sns.color_palette()
#py.init_notenotebook_mode(connected=True)
import plotly.tools as tlsdata
import nltk
from nltk.stem.porter import *
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.cluster import KMeans
from gensim.models import word2vec

from sklearn.manifold import TSNE
from sklearn import metrics
import sklearn
from sklearn.metrics import jaccard_similarity_score
cv = CountVectorizer()
from nltk.corpus import stopwords
from sklearn.metrics.pairwise import cosine_similarity
stop = set(stopwords.words("english"))

import warnings
warnings.filterwarnings('ignore')
import os
os.listdir("../input")

#input data files are available in the "../input" directory
#for example, running this will list the files in the input directory
data = pd.read_csv('../input/1429_1.csv', encoding="ISO-8859-1")
#keeping only the neceessary columns
#print(data.head())

#any results you write to the current directory are saved as output.

#--------------------------------------------------
```

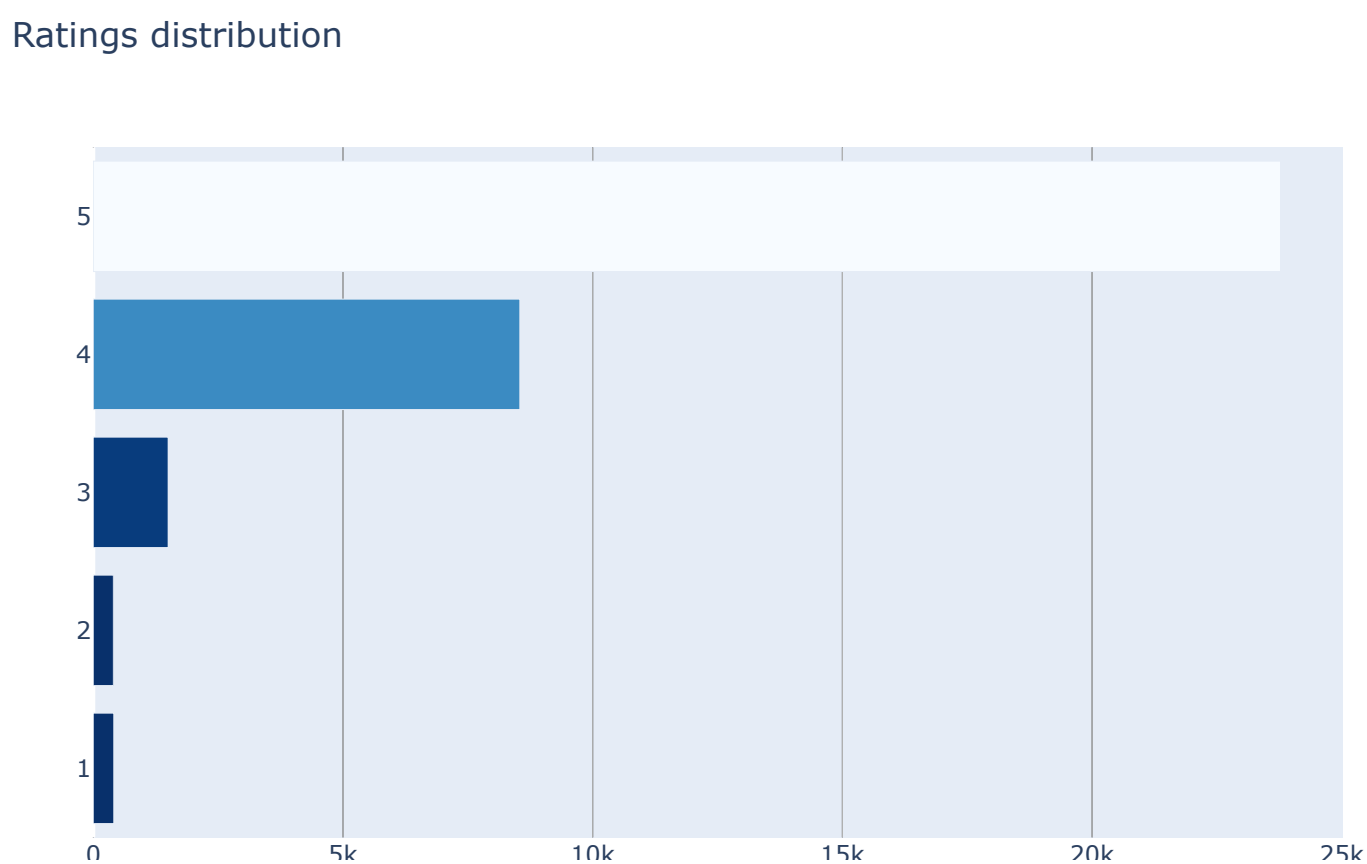
```
43
44 print(data.shape)
45 print(data.dtypes)
46 print(data.isnull().sum())
47 data = data.dropna(subset=['reviews.text'])
48
49 from wordcloud import WordCloud, STOPWORDS
50 stopwords = set(STOPWORDS)
51
52 def show_wordcloud(data, title = None):
53     wordcloud = WordCloud(
54         background_color='black',
55         stopwords=stopwords,
56         max_words=200,
57         max_font_size=40,
58         scale=3,
59         random_state=1
60         ).generate(str(data))
61
62     fig = plt.figure(1, figsize=(15,15))
63     plt.axis('off')
64     if title:
65         fig.suptitle(title, fontsize=20)
66         fig.subplot_adjust(top=2.3)
67
68     plt.imshow(wordcloud)
69     plt.show()
70
71
72 show_wordcloud(data['reviews.text'])
```

```
(34660, 21)
id                      object
name                    object
asins                   object
brand                   object
categories              object
keys                    object
manufacturer            object
reviews.date            object
reviews.dateAdded       object
reviews.dateSeen        object
```

```
reviews.didPurchase      object
reviews.doRecommend      object
reviews.id              float64
reviews.numHelpful      float64
reviews.rating          float64
reviews.sourceURLs       object
reviews.text             object
reviews.title            object
reviews.userCity        float64
reviews.userProvince    float64
reviews.username         object
dtype: object
id                          0
name                     6760
asins                       2
brand                       0
categories                  0
keys                        0
manufacturer                0
reviews.date               39
reviews.dateAdded       10621
reviews.dateSeen            0
reviews.didPurchase     34659
reviews.doRecommend       594
reviews.id              34659
reviews.numHelpful        529
reviews.rating             33
reviews.sourceURLs          0
reviews.text                1
reviews.title               5
reviews.userCity        34660
reviews.userProvince    34660
reviews.username            2
dtype: int64
```


Kindle
tablet
cover
Fire
love
Great
charger
Amazon
reading
Excellent
easy
quality
perfect
product
books
use
works
nice
Wanted
include
fun
app
really
need
screen
bought
HD
read
surprised
faster
Christmas
already
disappointed
size
Got
nothing
user
first
gave
given
ads
traveling
cease
father
Preloaded
present
happy
weeks
now
reader
beginner
learn
purchased
schooled
youAnd
graphics
experienced
everything
grandchildren
grand
Thank
wish
light
far
learned
inch
video
almost
ordering
easiest
recently
Inexpensive
lots
elderly
inlaws
two
want
large
rubber
daughter
Simply
person
downloading
several
handsome
gift
replacement
device
utilize
well
carry
quite
scr
graphite
home
si
DX2
go
Boug
th
Ki
Amaz
last
cove

In [2]:
```
cnt_srs = data['reviews.rating'].value_counts().head()
trace = go.Bar(
    y=cnt_srs.index[::-1],
    x=cnt_srs.values[::-1],
    orientation='h',
    marker=dict(
        color=cnt_srs.values[::-1],
        colorscale='Blues',
        reversescale=True),
    )
layout = dict(title='Ratings distribution')
data1 = [trace]
fig = go.Figure(data=data1, layout=layout)
py.iplot(fig, filename='Ratings')
```

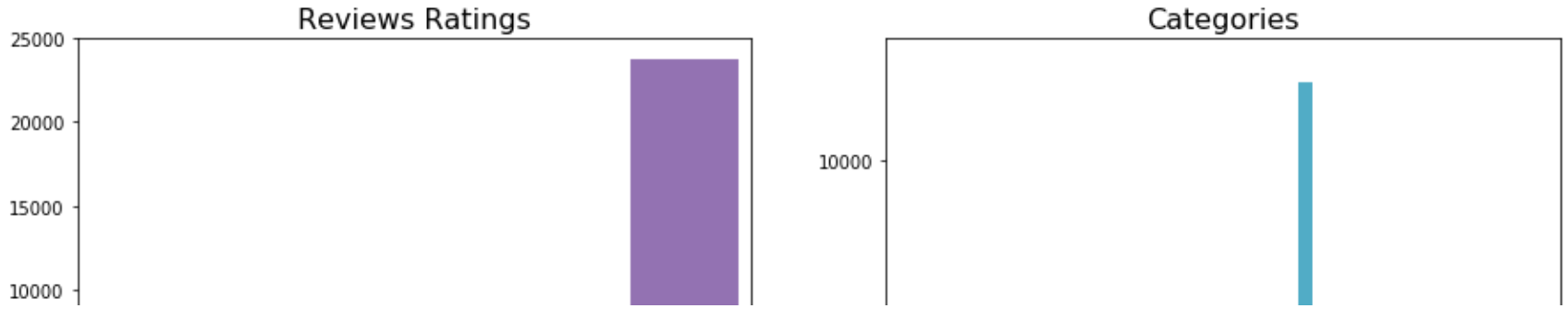


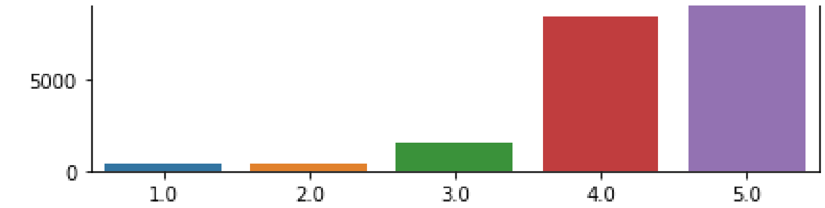
0
5K
10K
15K
20K
25K

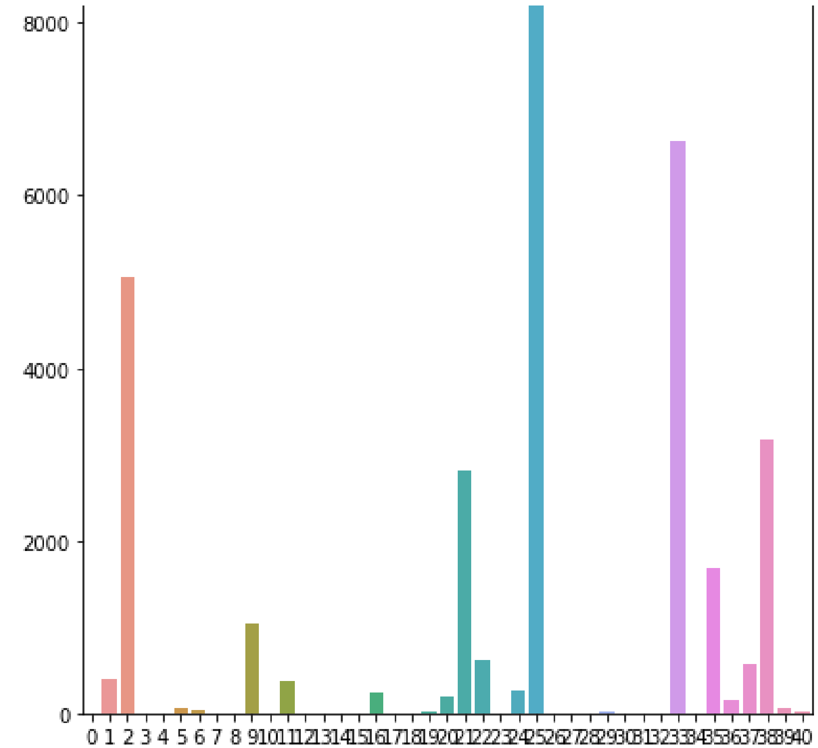
In [3]:

```
from subprocess import check_output
#print(check_output(["ls", "../input"]).decode("utf8"))
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import proj3d

from IPython.display import HTML
cat_hist = data.groupby('categories', as_index=False).count()
HTML(pd.DataFrame(cat_hist['categories']).to_html())
import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
import re
import string
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
from sklearn.metrics import f1_score
from sklearn.naive_bayes import GaussianNB
def removePunctuation(x):
    x = x.lower()
    x = re.sub(r'[^\x00-\x7f]',r' ', x) # replace the char that is not in the ASCII table
    return re.sub('['+string.punctuation+']', " ", x)

stops = set(stopwords.words("english"))
def removeStopwords(x):
    filterd_words = [word for word in x.split() if word not in stops]
    return "".join (filtered_words)
'''
When we deal with text problem in Natural Language Processing,
stop words removal process is a one of the important step to have a
better input for any models. Stop words means that it is a very
common words in a language (e.g. a, an, the in English. 的, 了 in
Chinese. え, も in Japanese). It does not help on most of NLP problem
such as semantic analysis, classification etc.
'''

def removeAmzString(x):
    return re.sub(r'[0-9]+ people found this helpful\. Was this review helpful to you Yes No', "", x)
# remove the amazon fixed sentence.

```

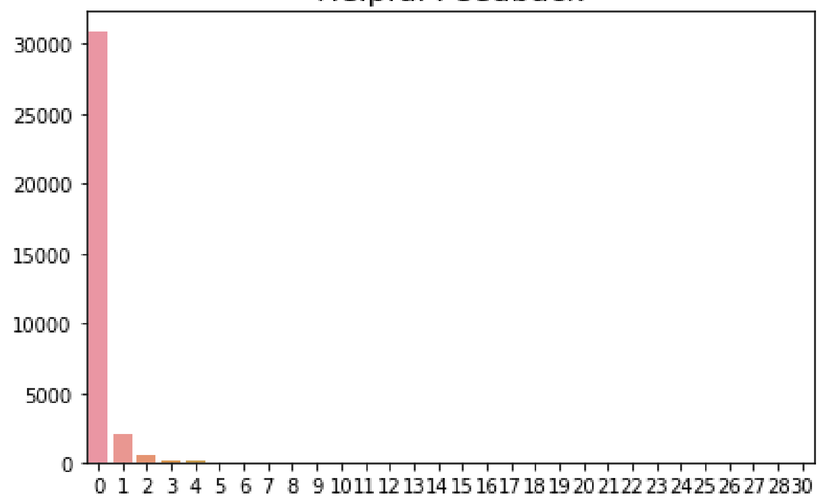
In [4]:
```
reviews =  [sent if type(sent)==str else "" for sent in data['reviews.title'].values]
reviews = [removeAmzString(sent) for sent in reviews]

reviews = [removePunctuation(sent) for sent in reviews]

stopwords = set(STOPWORDS)
wordcloud = WordCloud(background_color='white', stopwords=stopwords, max_words=200,
                      max_font_size=40,random_state=42).generate(str(reviews))
plt.figure(figsize=(15,20))
ax1 = plt.subplot2grid((4, 2), (0, 0))
ax2 = plt.subplot2grid((4, 2), (1, 0))
ax3 = plt.subplot2grid((4, 2), (0, 1), rowspan=2)
ax4 = plt.subplot2grid((4, 2), (2, 0), colspan=2,rowspan=2)

rat_hist = data.groupby('reviews.rating',as_index=False).count()
sns.barplot(x=rat_hist['reviews.rating'].values,y=rat_hist['id'].values,ax=ax1)

cat_hist = cat_hist.sort_values(by='id')
sns.barplot(x=cat_hist['categories'].index,y=cat_hist['id'].values,ax=ax3)

hf_hist = data.groupby('reviews.numHelpful',as_index=False).count()[0:30]
sns.barplot(x=hf_hist['reviews.numHelpful'].values.astype(int),y=hf_hist['id'].values,ax=ax2)

ax1.set_title("Reviews Ratings",fontsize=16)
ax3.set_title("Categories",fontsize=16)
ax2.set_title("Helpful Feedback",fontsize=16)
ax4.set_title("Words Cloud",fontsize=16)
ax4.imshow(wordcloud)
ax4.axis('off')
plt.show()

```







Helpful Feedback



Words Cloud


good tablet
great kids'
alexa
works great"
great gift
it' great
streaming device'
great product'
bought
great device
good
great price'
tablet great
great price
kids'
great
great value'

In [36]:

```
import nltk.stem as ns
from spellchecker import SpellChecker
def cleaning(s):
    s = str(s)
    s = s.lower()
    s = re.sub('\s\W',' ',s)
    s = re.sub('\W,\s',' ',s)
    s = re.sub(r'[^\w]', ' ', s)
    s = re.sub("\d+", "", s)
    s = re.sub('\s+',' ',s)
    s = re.sub('[!@#$_]', '', s)
    s = s.replace("co","")
    s = s.replace("https","")
    s = s.replace(",","")
    s = s.replace("[\w*"," ")
    s = s.replace(' s ', ' is ')
    s = s.replace('don t', 'do not')
    s = s.replace('doesn t', 'does not')
    s = s.replace('can t', 'cannot')
    s = s.replace('isn t', 'is not')
    s = s.replace('uldn t', 'uld not')
    s = s.replace('aren t', 'are not')
    s = s.replace('wasn t', 'was not')
    s = s.replace('weren t ', 'were not')
    s = s.replace('haven t', 'have not')
    s = s.replace('hasn t', 'has not')
    s = s.replace(' ve ', ' have ')
    s = s.replace(' wa ', ' was ')
    s = s.replace(' s ','')
    s = s.replace(' t ','')
    s = s.replace(' ntact', ' contact')
    s = s.replace(' nnect', ' connect')
    s = s.replace(' wasnt ', ' was not ')
    words = s.split()
    lemmatizer = ns.WordNetLemmatizer()
    for i in range(len(words)):
        words[i] = lemmatizer.lemmatize(words[i],'n')
    s = " ".join(words)
    return s
data['reviews.text'] = [cleaning(s) for s in data['reviews.text']]
data['revies.title'] = [cleaning(s) for s in data['reviews.title']]

```

In [37]:
```
from pathlib import Path
data = data.reset_index(drop=True)
outfile = Path('reviews.txt')
with outfile.open('w',encoding='utf-8') as w:
    for k in range(data['reviews.text'].shape[0]):
        w.write(data['reviews.text'][k]+'\n')
```

In [38]:
```
from sklearn.model_selection import train_test_split
from collections import  Counter
review = data['reviews.text'].to_numpy()
rate = data['reviews.rating'].to_numpy()
idx = np.argwhere(np.isnan(rate))
review_miss_rate = review[idx]
rate_miss = rate[idx]
review1 = np.delete(review, idx.T, axis=0)
rate1 = np.delete(rate, idx.T, axis=0)
review_train, review_test, rate_train, rate_test = train_test_split(review1,
                        rate1,test_size=0.2,random_state=10,stratify=rate1)
pd.DataFrame({'review':review_train}).to_csv('train_review.csv')
pd.DataFrame({'review':review_test}).to_csv('test_review.csv')
pd.DataFrame({'rate':rate_train}).to_csv('train_rate.csv')
pd.DataFrame({'rate':rate_test}).to_csv('test_rate.csv')
pd.DataFrame({'review':review_miss_rate.reshape(33)}).to_csv('miss_review_rate.csv')
pd.DataFrame({'rate':rate_miss.reshape(33)}).to_csv('miss_rate.csv')

```

In [39]:
```
outfile = Path('train_review.txt')
with outfile.open('w',encoding='utf-8') as w:
    for k in range(review_train.shape[0]):
        w.write(review_train[k]+'\n')

outfile = Path('test_review.txt')
with outfile.open('w',encoding='utf-8') as w:
    for k in range(review_test.shape[0]):
        w.write(review_test[k]+'\n')
```

In [ ]:
```
from sklearn.model_selection import train_test_split
df_train = pd.read_csv('review_train.csv')
df_train = df_train.dropna(axis=0,how='any')
df_train = df_train.reset_index(drop=True)
x = df_train['review'].to_numpy()
y = pd.read_csv('rate_train.csv')
y = y['rate'].to_numpy()
y = np.delete(y,12675,axis=0)# we don't have comment on idx = 12675
# whole training set
#train, val, y_train, y_val = train_test_split(x,
#                         y,test_size=0.2,random_state=10,stratify=y)
```

In [ ]:
```
from pathlib import Path
outfile = Path('../input/train.txt')
with outfile.open('w',encoding='utf-8') as w:
    for k in range(train.shape[0]):
        w.write(train[k]+'\n')
pd.DataFrame({'review':train}).to_csv('../input/train.csv')
```

In [60]:
```
from pathlib import Path
outfile = Path('../input/val.txt')
with outfile.open('w',encoding='utf-8') as w:
    for k in range(val.shape[0]):
        w.write(val[k]+'\n')
pd.DataFrame({'review':val}).to_csv('../input/val.csv')
```

In [ ]:
```
pd.DataFrame({'rate':y_train}).to_csv('../input/y_train.csv')
pd.DataFrame({'rate':y_val}).to_csv('../input/y_val.csv')
```

In [ ]:

```
""" Title: Word2Vec Model, Software Framework for Topic Modelling with Large Corpora
Author: Radim et ah. and Petr Sojka
Date: 2010.5.22
Availability: https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html#sphx-glr-download-auto-examples-
#generate a doc vector by averaging the word2vec vector
#use gensim.utils.simple_preprocess function
import pandas as pd
import numpy as np
df_train = pd.read_csv('../input/train.csv')
train = df_train['review'].to_numpy()
df_y_train = pd.read_csv('../input/y_train.csv')
y_train = df_y_train['rate'].to_numpy()
df_val = pd.read_csv('../input/val.csv')
val = df_val['review'].to_numpy()
df_y_val = pd.read_csv('../input/y_val.csv')
y_val = df_y_val['rate'].to_numpy()



# train word2vec model
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
from gensim.test.utils import datapath
from gensim import utils
from pathlib import Path

class MyCorpus(object):
    """An interator that yields sentences (lists of str)."""

    def __iter__(self):
        corpus_path = Path('../input/train.txt')
        for line in open(corpus_path):
            # assume there's one document per line, tokens separated by whitespace
            yield utils.simple_preprocess(line)

import gensim.models
sentences = MyCorpus()
model = gensim.models.Word2Vec(sentences=sentences, size=50, iter=5)
model.save('../embedding/word2vec.model')
model = gensim.models.Word2Vec.load('../embedding/word2vec.model')

# generate sentence vector for each sentence
```

```
word_vectors = model.wv
len(word_vectors.vocab)
for k, seq in enumerate(train):
    tokens = gensim.utils.simple_preprocess(seq)
    l = 0.
    for i in tokens:
        if i in word_vectors.vocab:
            ave_vec[k] += model.wv[i]
            l += 1
    ave_vec[k] /= l
print(ave_vec.shape)
```

In [15]:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from matplotlib import pyplot as plt
from sklearn.metrics import f1_score
from sklearn import linear_model
from sklearn.metrics import confusion_matrix
import numpy as np
from sklearn.externals import joblib
#import warnings
#warnings.filterwarnings("ignore")
df_train = pd.read_csv('../input/train.csv')
df_y_train = pd.read_csv('../input/y_train.csv')
df_val = pd.read_csv('../input/val.csv')
df_y_val = pd.read_csv('../input/y_val.csv')
train = df_train['review'].to_numpy()
y_train = df_y_train['rate'].to_numpy()
val = df_val['review'].to_numpy()
y_val = df_y_val['rate'].to_numpy()

# the data set is merge here for crossvalidation in the Bayesian Inference analysis.
'''
train_total = pd.DataFrame({'review':train.tolist()+val.tolist()})
train_y_total = pd.DataFrame({'rate':y_train.tolist()+y_val.tolist()})
train_total.to_csv('../input/train_total.csv')
train_y_total.to_csv('../input/train_y_total.csv')
'''

vectorizer = TfidfVectorizer(stop_words='english',use_idf=True)
model_tr = vectorizer.fit_transform(train)
model_val = vectorizer.transform(val)
```

In [14]:
```
from sklearn.decomposition import PCA
#from sklearn.preprocessing import normalize
#model_tr_normalize = normalize(model_tr, norm='l2', axis=0, copy=True, return_norm=False)
pca = PCA(n_components = 0.95)
pca.fit(model_tr.toarray())
reduced = pca.transform(model_tr.toarray())
print(reduced.shape)
```

```
(22159, 3192)
```

In [16]:
```
from sklearn.decomposition import PCA
model_tr1 = model_tr.copy()
model_val1 = model_val.copy()
#pca = joblib.load('../input/model/pca.pkl')
model_val = pca.transform(model_val.toarray())
model_tr = pca.transform(model_tr.toarray())

```

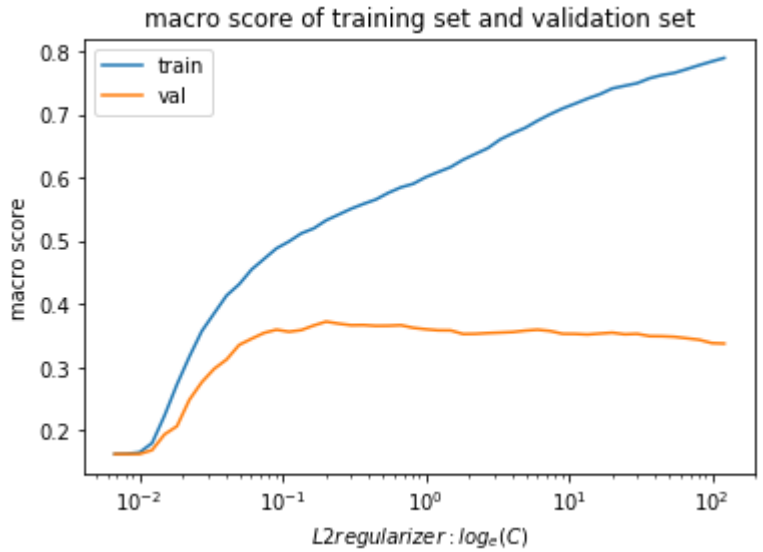
In [4]:

```
# Logistic regression, L2 regularization, weighted loss, hyperparameter selection use validation set, macro-weighted sc
# reason: we assume that all the variables are necessary for classification so we use a L2 regularizer.
# and we give more weight to the minor class to remedy for the class imblance.
import warnings
from matplotlib import pyplot as plt
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
warnings.filterwarnings("ignore")
from sklearn.linear_model import LogisticRegression
n_alphas = 76
#alphas = np.exp(np.linspace(-5,10,n_alphas))
C = np.exp(np.linspace(-5,10,n_alphas))
train_score = []
val_score = []
for i in C:
    m = LogisticRegression(C=i,class_weight='balanced').fit(model_tr, y_train)
    s = f1_score(y_val, m.predict(model_val), average='macro')
    train_score.append(f1_score(y_train, m.predict(model_tr), average='macro'))
    val_score.append(f1_score(y_val, m.predict(model_val), average='macro'))
    print ('Val macro score is', s)
#plt.plot(np.log10(C),train_score,label='train')
#plt.plot(np.log10(C),val_score,label='val')
#plt.title('macro score of training set and validation set')
#plt.legend()
#plt.xlabel('log10(C)')
#plt.ylabel('macro score')
#plt.show()
#------------------
```

...

In [5]:

```
from sklearn.metrics import confusion_matrix
ax = plt.gca()
# randomly choose 50 coefficients in the 9676 variables
# to plot figure of coefs vs regularizer.
ax.plot(C[:len(train_score)], train_score,label='train')
ax.plot(C[:len(train_score)], val_score,label='val')
ax.set_xscale('log')
#ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.xlabel('$L2 regularizer: log_e(C)$')
plt.ylabel('macro score')
plt.legend()
plt.title('macro score of training set and validation set')
plt.axis('tight')
plt.show()
C_opt = C[np.argmax(val_score)]
m_opt = LogisticRegression(C=C_opt,class_weight='balanced').fit(model_tr, y_train)
s = f1_score(y_val, m_opt.predict(model_val), average='macro')
mat = confusion_matrix(y_val, m_opt.predict(model_val))
print ('Best Val macro score is', s)
print('confusion matrix is: \n',mat)
print('recall of each class:\n',[round(mat[0,0]/mat[0,:].sum(),4), round(mat[1,1]/mat[1,:].sum(),4),
                                 round(mat[2,2]/mat[2,:].sum(),4),round(mat[3,3]/mat[3,:].sum(),4),
                                 round(mat[4,4]/mat[4,:].sum(),4)])
joblib.dump(m_opt,'../input/model/logistic_L2.pkl')
```

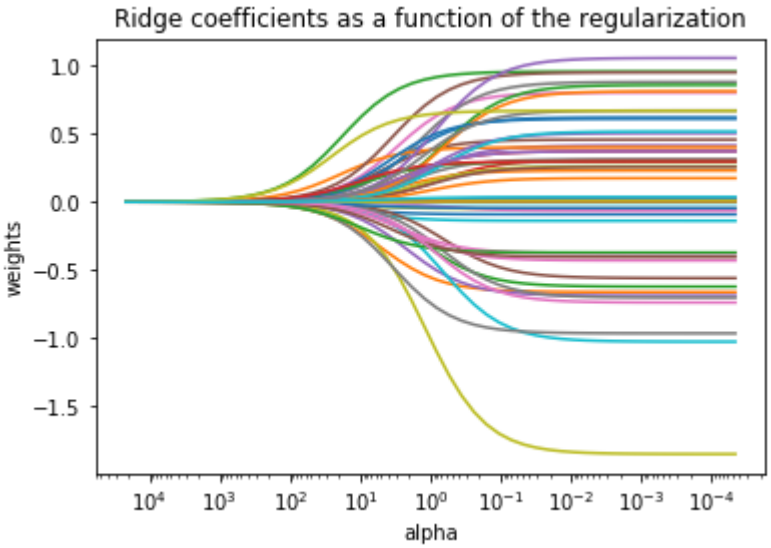



```
Best Val macro score is 0.3724644780023236
confusion matrix is:
 [[  31   12    8    2   12]
 [  18   12   13    6   15]
 [  22   23   73   39   83]
 [  25   25  121  299  897]
 [  33   37   78  262 3394]]
recall of each class:
 [0.4769, 0.1875, 0.3042, 0.2187, 0.8922]
```

Out[5]: ['../input/model/logistic_L2.pkl']

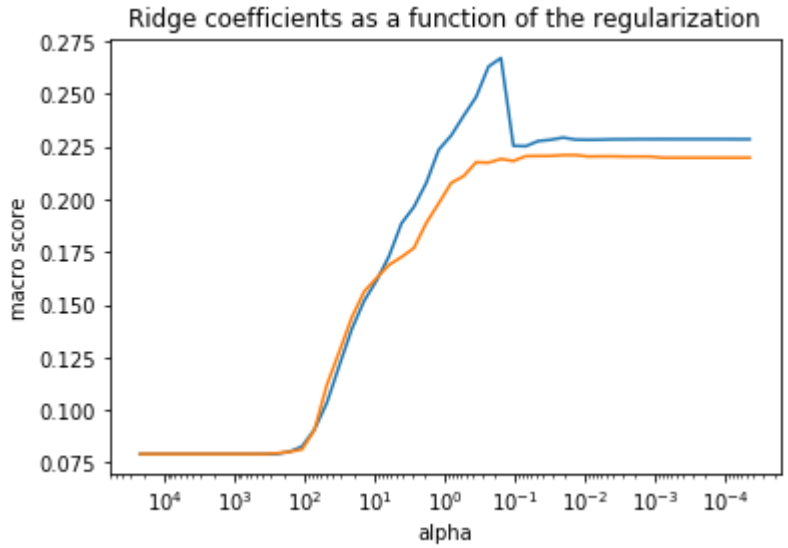
In [66]:

```
# ridge regression (L2 regularizer)
#
from sklearn import linear_model
n_alphas = 50
alphas = np.exp(np.linspace(-10,10,n_alphas))
coefs = []
train_score = []
val_score = []
for a in alphas:
    ridge = linear_model.Ridge(alpha=a, fit_intercept=True)
    ridge.fit(model_tr, y_train)
    train_pre = ridge.predict(model_tr).astype('int')
    train_pre[train_pre > 5] = np.int(5)
    train_pre[train_pre < 0] = np.int(1)
    val_pre = ridge.predict(model_val).astype('int')
    val_pre[val_pre > 5] = np.int(5)
    val_pre[val_pre < 1] = np.int(1)
    train_score.append(f1_score(y_train, train_pre, average='macro'))
    val_score.append(f1_score(y_val, val_pre, average='macro'))
    coefs.append(ridge.coef_)

cc = np.array(coefs)

ax = plt.gca()
# randomly choose 50 coefficients in the 9676 variables
# to plot figure of coefs vs regularizer.
ax.plot(alphas, cc[:,np.random.randint(0,3192,50)])
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Ridge coefficients as a function of the regularization')
plt.axis('tight')
plt.show()
```


Ridge coefficients as a function of the regularization
weights
alpha
1.0
0.5
0.0
−0.5
−1.0
−1.5
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}

In [67]:

```
train_score = np.array(train_score)
val_score = np.array(val_score)
ax1 = plt.gca()
# randomly choose 50 coefficients in the 9676 variables
# to plot figure of coefs vs regularizer.
ax1.plot(alphas, train_score)
ax1.plot(alphas, val_score)
ax1.set_xscale('log')
ax1.set_xlim(ax1.get_xlim()[::-1])  # reverse axis
plt.xlabel('alpha')
plt.ylabel('macro score')
plt.title('Ridge coefficients as a function of the regularization')
plt.axis('tight')
plt.show()

alphas_best = alphas[np.argmax(val_score)]
print('the best macro score on validation set:',np.max(val_score))
ridge = linear_model.Ridge(alpha=alphas_best, fit_intercept=True).fit(model_tr, y_train)
val_pre = ridge.predict(model_val).astype('int')
val_pre[val_pre > 5] = np.int(5)
val_pre[val_pre < 1] = np.int(1)
mat = confusion_matrix(y_val, val_pre)
print('confusion matrix is: \n',mat)
print('f1 score of each class:\n',f1_score(y_val, val_pre, average='macro'))
#print('the number of wi equal to 0 =',len(np.where(np.abs(m.coef_.flatten())==0)[0]))

```




```
the best macro score on validation set: 0.2209726787766575
confusion matrix is:
 [[   0   13   31   19    2]
 [   0   12   29   23    0]
 [   1   17   88  130    4]
 [   0   16  187 1067   97]
 [   0    5  123 3027  649]]
f1 score of each class:
 0.2209726787766575
```

In [68]:
```
joblib.dump(ridge,'../input/model/tf-idf_ridge.pkl')
```

Out[68]: ['../input/model/tf-idf_ridge.pkl']

Decision Tree

There are 7 hyperparameters in Decision Tree classifier. Since the dataset in this project has a large size and a large number of variables. Here I use some prior knowledge of this task to determine some hyperparameters and the order hyperparameter tuning. In the tuning process, I will use greedy search to find out the optimal hyperparameter.

1."criterion": It defines the criterion whether a node should be split and the two options are gini-index and entropy. When using entropy criterion, or saying the information gain between parent node and child node, the degree of impurity in a node can be better enlarged than using gini-index. But since our task has too many variables, using entropy criterion will result in more overfitting. So gini-index will be chose

to be the criterion.

2."splitter": we will choose "best" here so that our algorithm will choose the most important feature to split each time. Although we can use "random" and then merge the leave using some criterion. But because our task is too complecated, splitting the best node each can provide better stability.

3.parameters that I use val score to choose:

The parameters are ordered in their degree of importance to the model and they are choose according to validation set score.

3.1.max_depth: max_depth is the most important hyperparameter in decision, because it mainly decides the ability of the tree to fit the train set. Here I tune max_depth from 3 and fixed minimum of samples in leafs to 5, minmum of samples to split to 10. And give no constraint to maximum of feature number and minimum impurity decrease.

3.2.min_samples_leaf: min_samples_leaf can be use to smooth the boundary and with similar validation score a model with bigger min_samples_leaf can give a better boundary. I use the optimal max_length and the search for the best min_sample_leaf.

3.3.min_samples_split: This hyperparameter also can be use to make the boundary smoother.

In [3]:

```
from sklearn.tree import DecisionTreeClassifier
import warnings
warnings.filterwarnings("ignore")
clf = DecisionTreeClassifier(random_state=0,criterion='gini',
                             splitter='best' )

depth = np.linspace(3,183,61).astype('int')
train_score1 = []
val_score1 = []
for i in depth:
    m = DecisionTreeClassifier(random_state=0,
                               criterion='gini',
                               splitter='best',
                               max_depth=i,
                               min_samples_leaf=5,
                               min_samples_split=10,
                               class_weight='balanced'
                              )
    m.fit(model_tr, y_train)
    s = f1_score(y_val, m.predict(model_val), average='macro')
    train_score1.append(f1_score(y_train, m.predict(model_tr), average='macro'))
    val_score1.append(f1_score(y_val, m.predict(model_val), average='macro'))
    #print ('Val macro-weighted score is', s)

train_score1 = np.array(train_score1)
val_score1 = np.array(val_score1)

min_leaf = np.linspace(1,101,51).astype('int')
train_score2 = []
val_score2 = []
for i in min_leaf:
    m = DecisionTreeClassifier(random_state=0,
                               criterion='gini',
                               splitter='best',
                               max_depth=depth[np.argmax(val_score1)],
                               min_samples_leaf=i,
                               min_samples_split=10,
                               class_weight='balanced'
                              )
    m.fit(model_tr, y_train)
    s = f1_score(y_val, m.predict(model_val), average='macro')
    train_score2.append(f1_score(y_train, m.predict(model_tr), average='macro'))
```

```
    val_score2.append(f1_score(y_val, m.predict(model_val), average='macro'))

min_samples_spl = np.linspace(5,50,17).astype('int')
train_score3 = []
val_score3 = []
for i in min_samples_spl:
    m = DecisionTreeClassifier(random_state=0,
                               criterion='gini',
                               splitter='best',
                               max_depth=depth[np.argmax(val_score1)],
                               min_samples_leaf=min_leaf[np.argmax(val_score2)],
                               min_samples_split=i,
                               class_weight='balanced'
                              )
    m.fit(model_tr, y_train)
    s = f1_score(y_val, m.predict(model_val), average='macro')
    train_score3.append(f1_score(y_train, m.predict(model_tr), average='macro'))
    val_score3.append(f1_score(y_val, m.predict(model_val), average='macro'))

m_opt = DecisionTreeClassifier(random_state=0,
                               criterion='gini',
                               splitter='best',
                               max_depth=depth[np.argmax(val_score1)],
                               min_samples_leaf=min_leaf[np.argmax(val_score2)],
                               min_samples_split=min_samples_spl[np.argmax(val_score3)],
                               class_weight='balanced'
                              )
m_opt.fit(model_tr, y_train)
mat = confusion_matrix(y_val, m_opt.predict(model_val))
print('optimal score',f1_score(y_val, m_opt.predict(model_val), average='macro'))
print('confusion matrix is: \n',mat)
print('accuracy of each class:\n',[round(mat[0,0]/mat[0,:].sum(),4), round(mat[1,1]/mat[1,:].sum(),4),
                                   round(mat[2,2]/mat[2,:].sum(),4),round(mat[3,3]/mat[3,:].sum(),4),
                                   round(mat[4,4]/mat[4,:].sum(),4)])


```

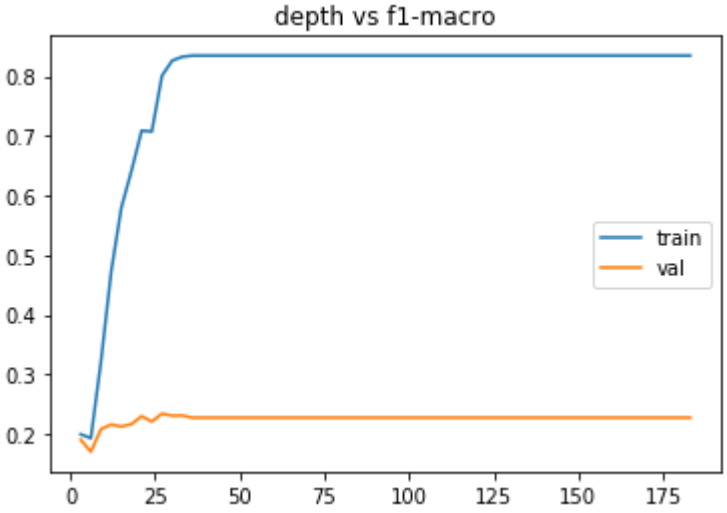
```
optimal score 0.23399102471368582
confusion matrix is:
 [[   4    5    8   19   29]
 [   3    7    2   25   27]
 [   5   16   38   73  108]
```

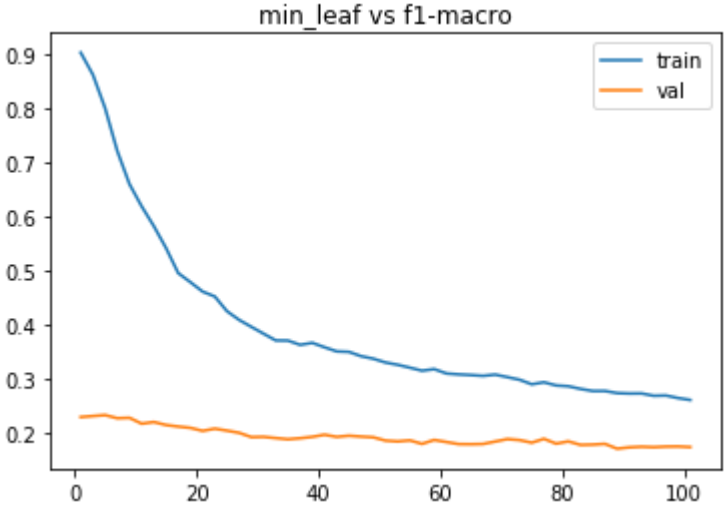
```
 [  29   36  124  433  745]
 [  75   74  283 1067 2305]]
accuracy of each class:
 [0.0615, 0.1094, 0.1583, 0.3168, 0.6059]
```

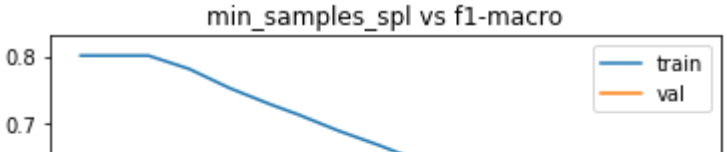
In [4]:

```
plt.figure()
plt.plot(depth,train_score1,label='train')
plt.plot(depth,val_score1,label='val')
plt.legend()
plt.title('depth vs f1-macro')
print(depth[np.argmax(val_score1)])
print(np.max(val_score1))
pass
plt.figure()
plt.plot(min_leaf,train_score2,label='train')
plt.plot(min_leaf,val_score2,label='val')
plt.legend()
plt.title('min_leaf vs f1-macro')
print(min_leaf[np.argmax(val_score2)])
print(np.max(val_score2))
pass
plt.figure()
plt.plot(min_samples_spl,train_score3,label='train')
plt.plot(min_samples_spl,val_score3,label='val')
plt.legend()
plt.title('min_samples_spl vs f1-macro')
print(min_samples_spl[np.argmax(val_score3)])
print(np.max(val_score3))
pass
```

```
27
0.23370315113393864
5
0.23370315113393864
13
0.23399102471368582
```


depth vs f1-macro
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
25
50
75
100
125
150
175
train
val


min_leaf vs f1-macro
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
train
val

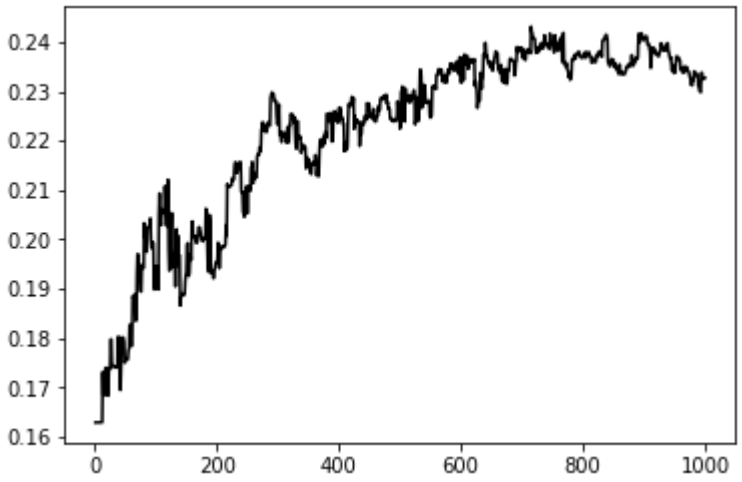


In [5]:
```
1 joblib.dump(m_opt,'../input/decision_tree.pkl')
```

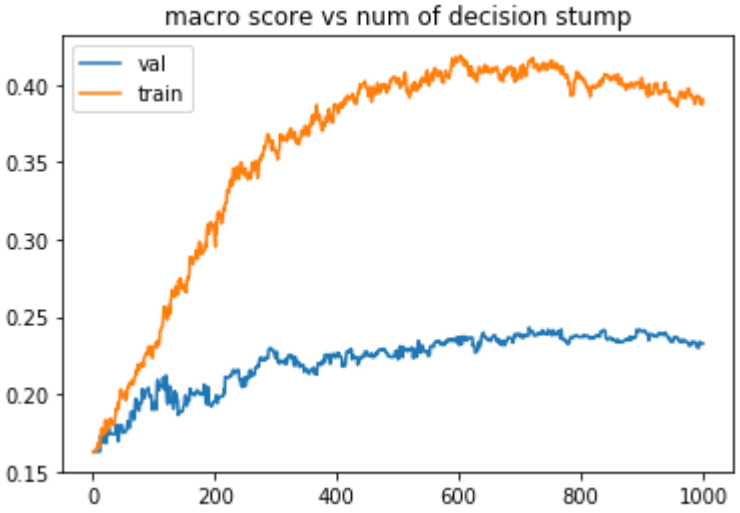
Out[5]: ['../input/decision_tree.pkl']

In [6]:
```
# Adaboost with tree method
#import warnings
#warnings.filterwarnings("ignore")
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
clf = AdaBoostClassifier(n_estimators=1000, random_state=1)
clf.fit(model_tr, y_train)
real_val_macro = [0]
for real_test_predict in clf.staged_predict(model_val):
    if f1_score(y_val, real_test_predict, average='macro')> np.max(real_val_macro) :
        pred_opt = real_test_predict
    real_val_macro.append(
        f1_score(y_val, real_test_predict, average='macro'))



n_trees_real = len(clf)
plt.figure()
plt.plot(range(1, n_trees_real + 1),
         real_val_macro[1:], c='black',
         label='SAMME.R')


```

Out[6]: [<matplotlib.lines.Line2D at 0x7f6d9887b6d8>]



In [9]:
```
real_val_macro = [0]
train_macro = []
for real_test_predict in clf.staged_predict(model_val):
    if f1_score(y_val, real_test_predict, average='macro')> np.max(real_val_macro) :
        pred_opt = f1_score(y_val, real_test_predict, average='macro')
    real_val_macro.append(
        f1_score(y_val, real_test_predict, average='macro'))
for real_train_predict in clf.staged_predict(model_tr):
    train_macro.append(f1_score(y_train, real_train_predict, average='macro'))


plt.figure()
plt.plot(range(1, n_trees_real + 1),
         real_val_macro[1:],
         label='val')
plt.plot(range(1, n_trees_real + 1),
         train_macro,
         label='train')
plt.legend()
plt.title('macro score vs num of decision stump')
adaboost_opt = AdaBoostClassifier(n_estimators=np.argmax(real_val_macro[1:])+1,
                                  random_state=1)
```



In [32]:
```
1 np.max(real_val_macro)
```

Out[32]: 0.24310635913853074

In [29]:
```
1 clf_ada_pca = AdaBoostClassifier(n_estimators=715, random_state=1).fit(model_tr,y_train)
```

In [64]:
```
1 joblib.dump(clf_ada_pca,'../input/model/adaboost1.pkl')
```

Out[64]: ['../input/model/adaboost1.pkl']

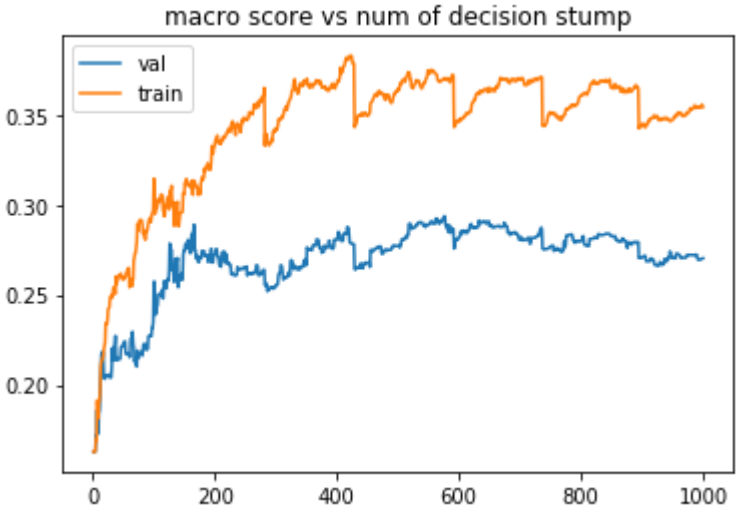
In [31]:
```
1 w2v_adaboost = clf_ada_pca
2 print('TF-IDF feature: Adaboost, best F1-macro score on validation set:',f1_score(y_val, w2v_adaboost.predict(model_val)
3 mat = confusion_matrix(y_val, w2v_adaboost.predict(model_val))
4 print('confusion matrix on validation set is: \n',mat)
5 print('F1 score of each class:\n',f1_score(y_val, w2v_adaboost.predict(model_val), average=None))
6 print('accuracy on validation set:', w2v_adaboost.score(model_val,y_val),'\n\n')
```

```
TF-IDF feature: Adaboost, best F1-macro score on validation set: 0.24310635913853074
confusion matrix on validation set is:
 [[   3    0    5   15   42]
 [   1    0    9   17   37]
 [   0    0   21   67  152]
 [   2    0   49  300 1016]
 [   3   11   61  574 3155]]
F1 score of each class:
 [0.08108108 0.         0.10909091 0.25641026 0.76894955]
accuracy on validation set: 0.6279783393501805
```

In [20]:

```
# Adaboost with tree method with out PCA
#import warnings
#warnings.filterwarnings("ignore")
from imblearn.over_sampling import SMOTE # doctest: +NORMALIZE_WHITESPACE
from collections import Counter

from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
clf_no_pca = AdaBoostClassifier(n_estimators=1000, random_state=1)
clf_no_pca.fit(model_tr1, y_train)
real_val_macro1 = [0]
train_macro1 = []
for real_test_predict in clf_no_pca.staged_predict(model_val1):
    if f1_score(y_val, real_test_predict, average='macro')> np.max(real_val_macro1) :
        pred_opt1 = real_test_predict
    real_val_macro1.append(
        f1_score(y_val, real_test_predict, average='macro'))
for real_train_predict in clf_no_pca.staged_predict(model_tr1):
    train_macro1.append(f1_score(y_train, real_train_predict, average='macro'))
```

In [22]:
```
plt.figure()
plt.plot(range(1, n_trees_real + 1),
         real_val_macro1[1:],
         label='val')
plt.plot(range(1, n_trees_real + 1),
         train_macro1,
         label='train')
plt.legend()
plt.title('macro score vs num of decision stump')
adaboost_opt = AdaBoostClassifier(n_estimators=np.argmax(real_val_macro[1:])+1,
                                  random_state=1)

```



In [39]:
```
np.argmax(real_val_macro1)
```

Out[39]: 577

In [40]:
```
clf_no_pca_opt = AdaBoostClassifier(n_estimators=577, random_state=1)
clf_no_pca_opt.fit(model_tr1, y_train)
```

Out[40]:
```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=577, random_state=1)
```

In [42]:
```
w2v_adaboost = clf_no_pca_opt
print('TF-IDF feature: Adaboost, best F1-macro score on validation set:',f1_score(y_val, w2v_adaboost.predict(model_val1
mat = confusion_matrix(y_val, w2v_adaboost.predict(model_val1))
print('confusion matrix on validation set is: \n',mat)
print('F1 score of each class:\n',f1_score(y_val, w2v_adaboost.predict(model_val1), average=None))
print('accuracy on validation set:', w2v_adaboost.score(model_val1,y_val),'\n\n')
```

```
TF-IDF feature: Adaboost, best F1-macro score on validation set: 0.29365991902671673
confusion matrix on validation set is:
 [[   6    4    8   16   31]
 [   2    8    4   20   30]
 [   7    8   28   70  127]
 [  12    7   22  251 1075]
 [  28   11   39  329 3397]]
F1 score of each class:
 [0.1        0.15686275 0.16422287 0.24452021 0.80269376]
accuracy on validation set: 0.6660649819494585
```

In [25]:
```
joblib.dump(adaboost_opt,'../input/model/adaboost_nopca.pkl')
```

Out[25]:
```
['../input/model/adaboost_nopca.pkl']
```

In [45]:
```
from sklearn.linear_model import LogisticRegression
n_alphas = 41
#alphas = np.exp(np.linspace(-5,10,n_alphas))
C = np.exp(np.linspace(-5,10,n_alphas))
a = np.linspace(0.2,0.7,6)
train_score = [[],[],[],[],[],[]]
val_score = [[],[],[],[],[],[]]
for n,u in enumerate(a):
    for j,i in enumerate(C):
        m = LogisticRegression(penalty='elasticnet',C=i,class_weight='balanced',n_jobs=-1, l1_ratio=u,solver='saga').f
        train_pre = m.predict(model_tr)
        val_pre = m.predict(model_val)
        train_score[n].append(f1_score(y_train, train_pre, average='macro'))
        val_score[n].append(f1_score(y_val, val_pre, average='macro'))

```

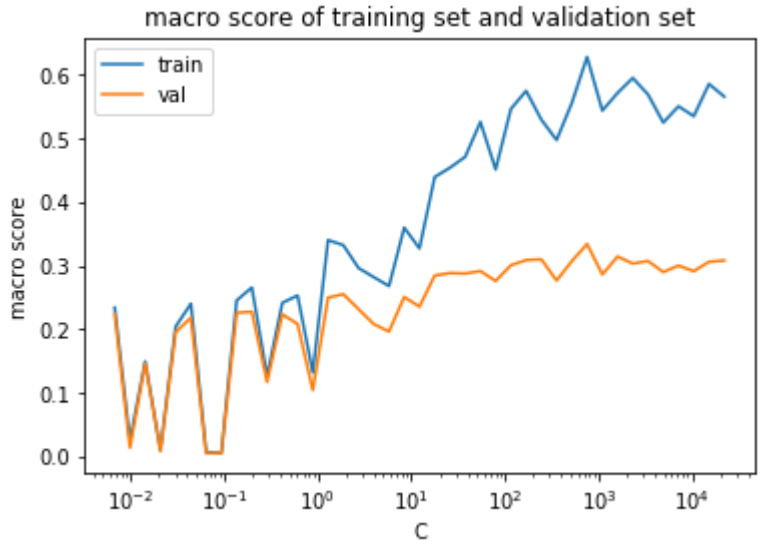
In [60]:
```
elastic_opt = LogisticRegression(penalty='elasticnet',
                                 C=C[(np.argmax(val_score))%41],
                                 class_weight='balanced',n_jobs=-1,
                                 l1_ratio=a[(np.argmax(val_score))//41],
                                 solver='saga')
elastic_opt.fit(model_tr,y_train)
print('the best macro score on validation set:',f1_score(y_val, elastic_opt.predict(model_val), average='macro'))
mat = confusion_matrix(y_val, elastic_opt.predict(model_val))
print('confusion matrix is: \n',mat)
print('accuracy of each class:\n',f1_score(y_val, elastic_opt.predict(model_val), average=None))
```

```
the best macro score on validation set: 0.31207363766143204
confusion matrix is:
 [[  30   14    4    5   12]
 [  20   18    6   11    9]
 [  32   34   39   92   43]
 [  50   76   99  566  576]
 [ 102  113   95 1019 2475]]
accuracy of each class:
 [0.2006689  0.11285266 0.16149068 0.36993464 0.7154213 ]
```

In [62]:
```
joblib.dump(elastic_opt,'../input/model/logistic_elsticnet.pkl')
```

Out[62]: ['../input/model/logistic_elsticnet.pkl']

In [63]:
```
train_score = np.array(train_score)
val_score = np.array(val_score)
ax1 = plt.gca()
# randomly choose 50 coefficients in the 9676 variables
# to plot figure of coefs vs regularizer.
ax1.plot(C, train_score[(np.argmax(val_score))//41], label='train')
ax1.plot(C, val_score[(np.argmax(val_score))//41],label='val')
ax1.set_xscale('log')
#ax1.set_xlim(ax1.get_xlim()[::-1])  # reverse axis
plt.xlabel('C')
plt.ylabel('macro score')
plt.title('macro score of training set and validation set')
plt.axis('tight')
plt.legend()
plt.show()
```



In [1]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from matplotlib import pyplot as plt
from sklearn.metrics import f1_score
from sklearn import linear_model
from sklearn.metrics import confusion_matrix
import numpy as np
from sklearn.externals import joblib
#import warnings
#warnings.filterwarnings("ignore")
df_train = pd.read_csv('../input/train.csv')
df_y_train = pd.read_csv('../input/y_train.csv')
df_val = pd.read_csv('../input/val.csv')
df_y_val = pd.read_csv('../input/y_val.csv')
train = df_train['review'].to_numpy()
y_train = df_y_train['rate'].to_numpy()
val = df_val['review'].to_numpy()
y_val = df_y_val['rate'].to_numpy()

# the data set is merge here for crossvalidation in the Bayesian Inference analysis.
'''
train_total = pd.DataFrame({'review':train.tolist()+val.tolist()})
train_y_total = pd.DataFrame({'rate':y_train.tolist()+y_val.tolist()})
train_total.to_csv('../input/train_total.csv')
train_y_total.to_csv('../input/train_y_total.csv')
'''

vectorizer = TfidfVectorizer(stop_words='english',use_idf=True)
model_tr = vectorizer.fit_transform(train)
model_val = vectorizer.transform(val)
```

```
/home/zixi/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/__init__.py:15: DeprecationWarning: sklearn.externals.joblib is deprecated in 0.21 and will be removed in 0.23. Please import this functionality directly from joblib, which can be installed with: pip install joblib. If this warning is raised when loading pickled models, you may need to re-serialize those models with scikit-learn 0.21+.
  warnings.warn(msg, category=DeprecationWarning)
```

In [14]:
```
from sklearn.decomposition import PCA
#from sklearn.preprocessing import normalize
#model_tr_normalize = normalize(model_tr, norm='l2', axis=0, copy=True, return_norm=False)
pca = PCA(n_components = 0.95)
pca.fit(model_tr.toarray())
reduced = pca.transform(model_tr.toarray())
print(reduced.shape)
```

```
(22159, 3192)
```

In [2]:
```
from sklearn.decomposition import PCA
model_tr1 = model_tr
model_val1 = model_val
#pca = joblib.load('../input/model/pca.pkl')
model_val = pca.transform(model_val.toarray())
model_tr = pca.transform(model_tr.toarray())

```

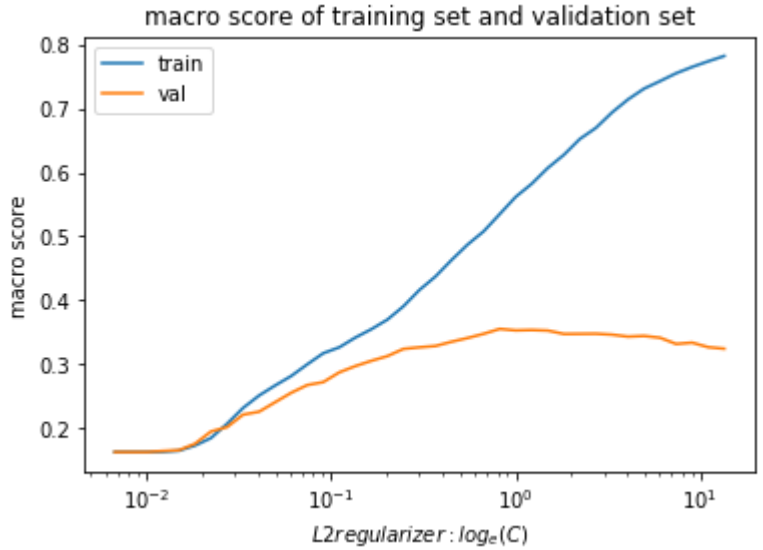
In [5]:

```
# Logistic regression, L1 regularization, weighted loss, hyperparameter selection use validation set, macro-weighted sc
# reason: we assume that some of the variable which is corresponding to the word contribute little to emotion(rating)
# and we give more weight to the minor class to remedy for the class imblance.
import warnings
from matplotlib import pyplot as plt
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
warnings.filterwarnings("ignore")
n_alphas = 76
#alphas = np.exp(np.linspace(-5,10,n_alphas))
C = np.exp(np.linspace(-5,10,n_alphas))
train_score = []
val_score = []
for i in C:
    m = LogisticRegression(penalty='l1',C=i,class_weight='balanced').fit(model_tr, y_train)
    s = f1_score(y_val, m.predict(model_val), average='macro')
    train_score.append(f1_score(y_train, m.predict(model_tr), average='macro'))
    val_score.append(f1_score(y_val, m.predict(model_val), average='macro'))
    print ('Val macro score is', s)
#plt.plot(np.log10(C),train_score,label='train')
#plt.plot(np.log10(C),val_score,label='val')
#plt.title('macro score of training set and validation set')
#plt.legend()
#plt.xlabel('log10(C)')
#plt.ylabel('macro score')
#plt.show()
#-------------------
```

...

In [6]:

```
from sklearn.metrics import confusion_matrix
ax = plt.gca()
# randomly choose 50 coefficients in the 9676 variables
# to plot figure of coefs vs regularizer.
ax.plot(C[:len(train_score)], train_score,label='train')
ax.plot(C[:len(train_score)], val_score,label='val')
ax.set_xscale('log')
#ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.xlabel('$L2 regularizer: log_e(C)$')
plt.ylabel('macro score')
plt.legend()
plt.title('macro score of training set and validation set')
plt.axis('tight')
plt.show()
C_opt = C[np.argmax(val_score)]
m_opt = LogisticRegression(C=C_opt,class_weight='balanced').fit(model_tr, y_train)
s = f1_score(y_val, m_opt.predict(model_val), average='macro')
mat = confusion_matrix(y_val, m_opt.predict(model_val))
print ('Best Val macro score is', s)
print('confusion matrix is: \n',mat)
print('recall of each class:\n',[round(mat[0,0]/mat[0,:].sum(),4), round(mat[1,1]/mat[1,:].sum(),4),
                                 round(mat[2,2]/mat[2,:].sum(),4),round(mat[3,3]/mat[3,:].sum(),4),
                                 round(mat[4,4]/mat[4,:].sum(),4)])
joblib.dump(m_opt,'../input/model/logistic_L1.pkl')
```




```
Best Val macro score is 0.3626043565870517
confusion matrix is:
 [[  26    8   16    3   12]
 [  16   14   18    8    8]
 [  22   22   73   54   69]
 [  23   29  159  369  787]
 [  37   42  120  400 3205]]
recall of each class:
 [0.4, 0.2188, 0.3042, 0.2699, 0.8425]
```

Out[6]: ['../input/model/logistic_L2.pkl']

In [60]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score

df_train = pd.read_csv('../input/train.csv')
train = df_train['review'].to_numpy()
df_y_train = pd.read_csv('../input/y_train.csv')
y_train = df_y_train['rate'].to_numpy()
df_val = pd.read_csv('../input/val.csv')
val = df_val['review'].to_numpy()
df_y_val = pd.read_csv('../input/y_val.csv')
y_val = df_y_val['rate'].to_numpy()

import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
from gensim.test.utils import datapath
from gensim import utils
from pathlib import Path
import gensim
model = gensim.models.Word2Vec.load('../embedding/word2vec.model')
ave_vec = np.zeros((train.shape[0],50),dtype='float')
word_vectors = model.wv
len(word_vectors.vocab)
for k, seq in enumerate(train):
    tokens = gensim.utils.simple_preprocess(seq)
    for i in tokens:
        if i in word_vectors.vocab:
            ave_vec[k] += model.wv[i]
    ave_vec[k] /= len(tokens)
print(ave_vec.shape)
```

```
(22159, 50)
```

In [61]:
```
word_vectors = model.wv
len(word_vectors.vocab)
ave_vec_val = np.zeros((val.shape[0],50),dtype='float')
for k, seq in enumerate(val):
    tokens = gensim.utils.simple_preprocess(seq)
    for i in tokens:
        if i in word_vectors.vocab:
            ave_vec_val[k] += model.wv[i]
    ave_vec_val[k] /= np.array(len(tokens))
print(ave_vec_val.shape)
```

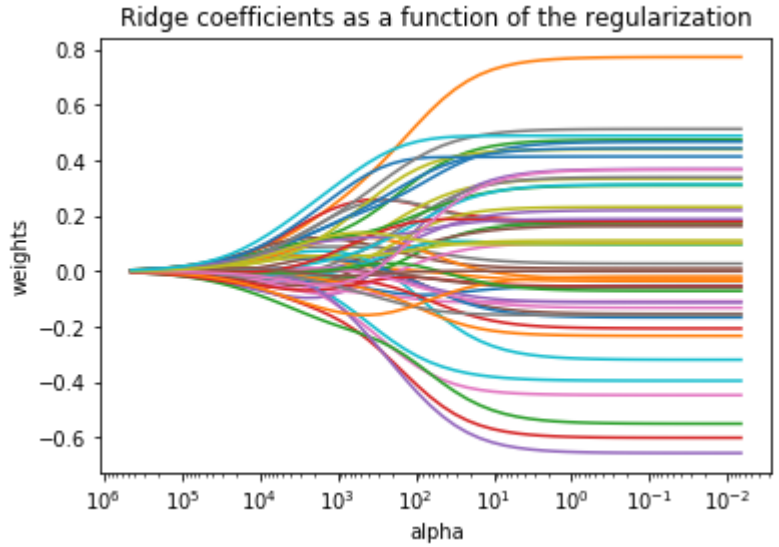
```
(5540, 50)
```

In [97]:

```
# ridge regression (L2 regularizer)
#
print(Counter(y_train))
print(Counter(y_val))
stra = {5.0:15214,
        4.0:10000,
        3.0:2000,
        2.0:1000,
        1.0:1000}
sm = SMOTE(random_state=42, sampling_strategy=stra)
X_res, y_res = sm.fit_resample(ave_vec, y_train)
print(Counter(y_res))
from sklearn import linear_model
n_alphas = 50
alphas = np.exp(np.linspace(-5,13,n_alphas))
coefs = []
train_score = []
val_score = []
for a in alphas:
    ridge = linear_model.Ridge(alpha=a, fit_intercept=True)
    ridge.fit(X_res, y_res)
    train_pre = ridge.predict(ave_vec).astype('int')
    train_pre[train_pre > 5] = np.int(5)
    train_pre[train_pre < 0] = np.int(1)
    val_pre = ridge.predict(ave_vec_val).astype('int')
    val_pre[val_pre > 5] = np.int(5)
    val_pre[val_pre < 1] = np.int(1)
    train_score.append(f1_score(y_train, train_pre, average='macro'))
    val_score.append(f1_score(y_val, val_pre, average='macro'))
    coefs.append(ridge.coef_)

cc = np.array(coefs)

ax = plt.gca()
# randomly choose 50 coefficients in the 9676 variables
# to plot figure of coefs vs regularizer.
ax.plot(alphas, cc)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Ridge coefficients as a function of the regularization')
```

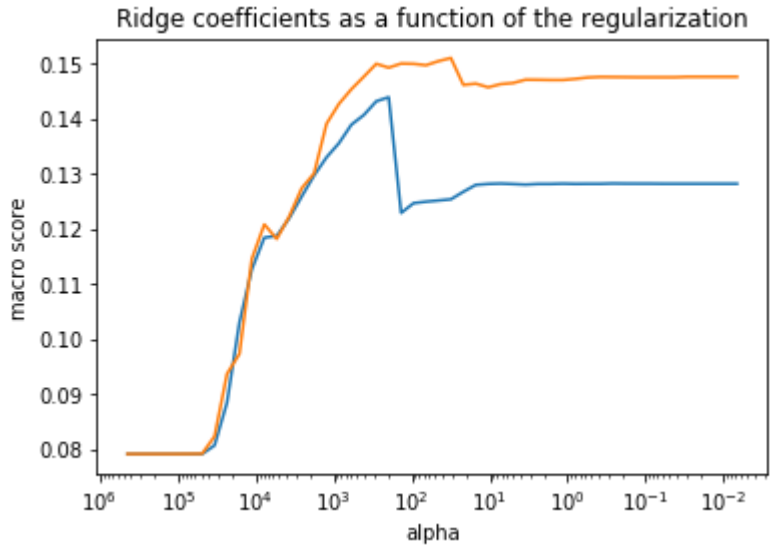
```
plt.axis('tight')
plt.show()
```

```
Counter({5.0: 15214, 4.0: 5466, 3.0: 959, 1.0: 263, 2.0: 257})
Counter({5.0: 3804, 4.0: 1367, 3.0: 240, 1.0: 65, 2.0: 64})
Counter({5.0: 15214, 4.0: 10000, 3.0: 2000, 2.0: 1000, 1.0: 1000})
```



In [98]:

```
train_score = np.array(train_score)
val_score = np.array(val_score)
ax1 = plt.gca()
# randomly choose 50 coefficients in the 9676 variables
# to plot figure of coefs vs regularizer.
ax1.plot(alphas, train_score)
ax1.plot(alphas, val_score)
ax1.set_xscale('log')
ax1.set_xlim(ax1.get_xlim()[::-1])  # reverse axis
plt.xlabel('alpha')
plt.ylabel('macro score')
plt.title('Ridge coefficients as a function of the regularization')
plt.axis('tight')
plt.show()

alphas_best = alphas[np.argmax(val_score)]
print('the best macro score on validation set:',np.max(val_score))
ridge = linear_model.Ridge(alpha=alphas_best, fit_intercept=True).fit(X_res, y_res)
val_pre = ridge.predict(ave_vec_val).astype('int')
val_pre[val_pre > 5] = np.int(5)
val_pre[val_pre < 1] = np.int(1)
mat = confusion_matrix(y_val, val_pre)
print('confusion matrix is: \n',mat)
print('accuracy of each class:\n',[round(mat[0,0]/mat[0,:].sum(),4), round(mat[1,1]/mat[1,:].sum(),4),
                                   round(mat[2,2]/mat[2,:].sum(),4),round(mat[3,3]/mat[3,:].sum(),4),
                                   round(mat[4,4]/mat[4,:].sum(),4)])
print('the number of wi equal to 0 =',len(np.where(np.abs(ridge.coef_.flatten())==0)[0]))

```




```
the best macro score on validation set: 0.15098002820153628
confusion matrix is:
 [[   0    1   49   15    0]
 [   0    1   54    9    0]
 [   0    6  157   74    3]
 [   0    4  424  878   61]
 [   0    6  497 2927  374]]
accuracy of each class:
 [0.0, 0.0156, 0.6542, 0.6423, 0.0983]
the number of wi equal to 0 = 0
```

In [99]:
```
1 joblib.dump(ridge,'../input/model_w2v/w2v_ridge.pkl')
```

Out[99]:
```
['../input/model_w2v/w2v_ridge.pkl']
```

In [80]:

```
from sklearn.tree import DecisionTreeClassifier
import warnings
warnings.filterwarnings("ignore")
clf = DecisionTreeClassifier(random_state=0,criterion='entropy',
                             splitter='best' )

depth = np.linspace(3,183,61).astype('int')
train_score1 = []
val_score1 = []
for i in depth:
    m = DecisionTreeClassifier(random_state=0,
                               criterion='entropy',
                               splitter='best',
                               max_depth=i,
                               min_samples_leaf=5,
                               min_samples_split=10,
                               class_weight='balanced'
                              )
    m.fit(X_res, y_res)
    s = f1_score(y_val, m.predict(ave_vec_val), average='macro')
    train_score1.append(f1_score(y_train, m.predict(ave_vec), average='macro'))
    val_score1.append(f1_score(y_val, m.predict(ave_vec_val), average='macro'))
    #print ('Val macro-weighted score is', s)

train_score1 = np.array(train_score1)
val_score1 = np.array(val_score1)

min_leaf = np.linspace(1,101,51).astype('int')
train_score2 = []
val_score2 = []
for i in min_leaf:
    m = DecisionTreeClassifier(random_state=0,
                               criterion='entropy',
                               splitter='best',
                               max_depth=depth[np.argmax(val_score1)],
                               min_samples_leaf=i,
                               min_samples_split=10,
                               class_weight='balanced'
                              )
    m.fit(X_res, y_res)
    s = f1_score(y_val, m.predict(ave_vec_val), average='macro')
    train_score2.append(f1_score(y_train, m.predict(ave_vec), average='macro'))
```

```
    val_score2.append(f1_score(y_val, m.predict(ave_vec_val), average='macro'))

min_samples_spl = np.linspace(5,50,17).astype('int')
train_score3 = []
val_score3 = []
for i in min_samples_spl:
    m = DecisionTreeClassifier(random_state=0,
                               criterion='entropy',
                               splitter='best',
                               max_depth=depth[np.argmax(val_score1)],
                               min_samples_leaf=min_leaf[np.argmax(val_score2)],
                               min_samples_split=i,
                               class_weight='balanced'
                              )
    m.fit(X_res, y_res)
    s = f1_score(y_val, m.predict(ave_vec_val), average='macro')
    train_score3.append(f1_score(y_train, m.predict(ave_vec), average='macro'))
    val_score3.append(f1_score(y_val, m.predict(ave_vec_val), average='macro'))

m_opt = DecisionTreeClassifier(random_state=0,
                               criterion='entropy',
                               splitter='best',
                               max_depth=depth[np.argmax(val_score1)],
                               min_samples_leaf=min_leaf[np.argmax(val_score2)],
                               min_samples_split=min_samples_spl[np.argmax(val_score3)],
                               class_weight='balanced'
                              )
m_opt.fit(X_res, y_res)
mat = confusion_matrix(y_val, m_opt.predict(ave_vec_val))
print('optimal score',f1_score(y_val, m_opt.predict(ave_vec_val), average='macro'))
print('confusion matrix is: \n',mat)
print('accuracy of each class:\n',[round(mat[0,0]/mat[0,:].sum(),4), round(mat[1,1]/mat[1,:].sum(),4),
                                   round(mat[2,2]/mat[2,:].sum(),4),round(mat[3,3]/mat[3,:].sum(),4),
                                   round(mat[4,4]/mat[4,:].sum(),4)])
```

```
optimal score 0.23430500637813695
confusion matrix is:
 [[   4    3    8   21   29]
 [   7    4   21   13   19]
 [  11   14   39   94   82]
 [  48   48  133  477  661]
```

```
 [  73   66  237 1118 2310]]
accuracy of each class:
 [0.0615, 0.0625, 0.1625, 0.3489, 0.6073]
```

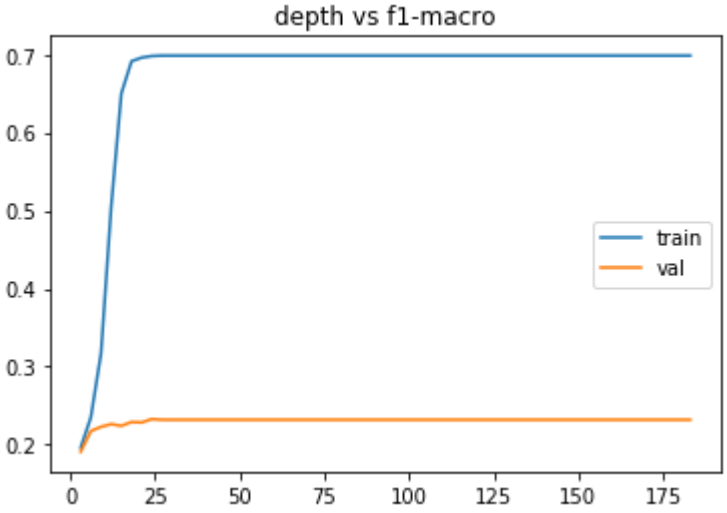
In [81]:

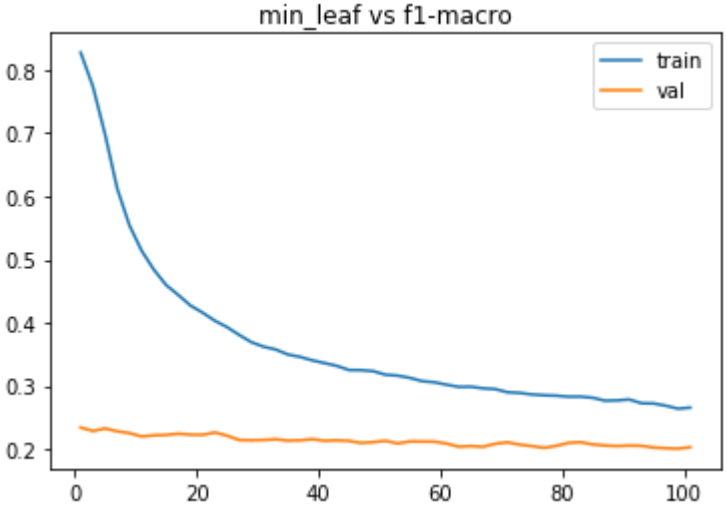
```
mat = confusion_matrix(y_val, m_opt.predict(ave_vec_val))
print('optimal score',f1_score(y_val, m_opt.predict(ave_vec_val), average='macro'))
print('confusion matrix is: \n',mat)
print('f1 of each class:\n',f1_score(y_val, m_opt.predict(ave_vec_val), average=None))

```

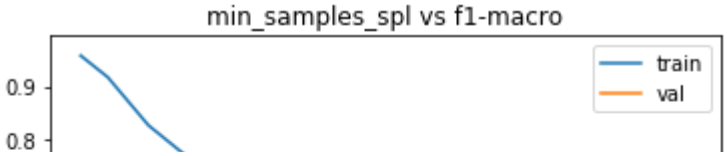
```
optimal score 0.23430500637813695
confusion matrix is:
 [[   4    3    8   21   29]
 [   7    4   21   13   19]
 [  11   14   39   94   82]
 [  48   48  133  477  661]
 [  73   66  237 1118 2310]]
f1 of each class:
 [0.03846154 0.04020101 0.11504425 0.30873786 0.66908038]
```

In [82]:
```
plt.figure()
plt.plot(depth,train_score1,label='train')
plt.plot(depth,val_score1,label='val')
plt.legend()
plt.title('depth vs f1-macro')
print(depth[np.argmax(val_score1)])
print(np.max(val_score1))
pass
plt.figure()
plt.plot(min_leaf,train_score2,label='train')
plt.plot(min_leaf,val_score2,label='val')
plt.legend()
plt.title('min_leaf vs f1-macro')
print(min_leaf[np.argmax(val_score2)])
print(np.max(val_score2))
pass
plt.figure()
plt.plot(min_samples_spl,train_score3,label='train')
plt.plot(min_samples_spl,val_score3,label='val')
plt.legend()
plt.title('min_samples_spl vs f1-macro')
print(min_samples_spl[np.argmax(val_score3)])
print(np.max(val_score3))
pass
```

```
24
0.23279484750733576
1
0.23430500637813695
10
0.23430500637813695
```


depth vs f1-macro
0.7
0.6
0.5
0.4
0.3
0.2
0
25
50
75
100
125
150
175
train
val


min_leaf vs f1-macro
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
train
val



In [83]:
```
1 joblib.dump(m_opt,'../input/model_w2v/decision_tree_w2v.pkl')
```

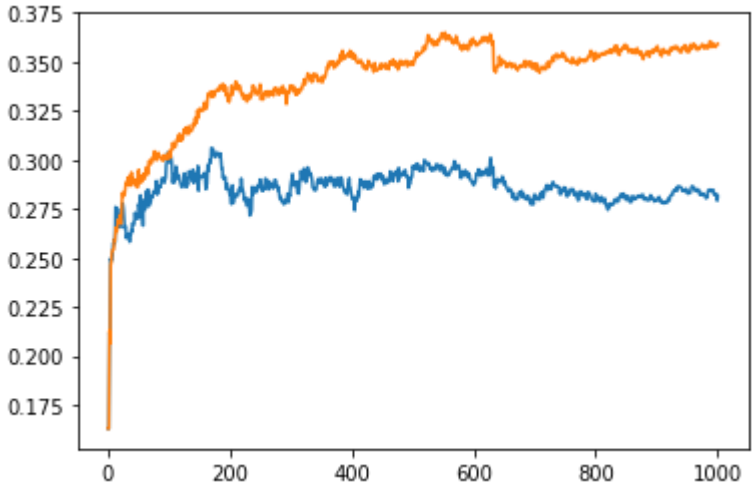
Out[83]: ['../input/model_w2v/decision_tree_w2v.pkl']

In [71]:

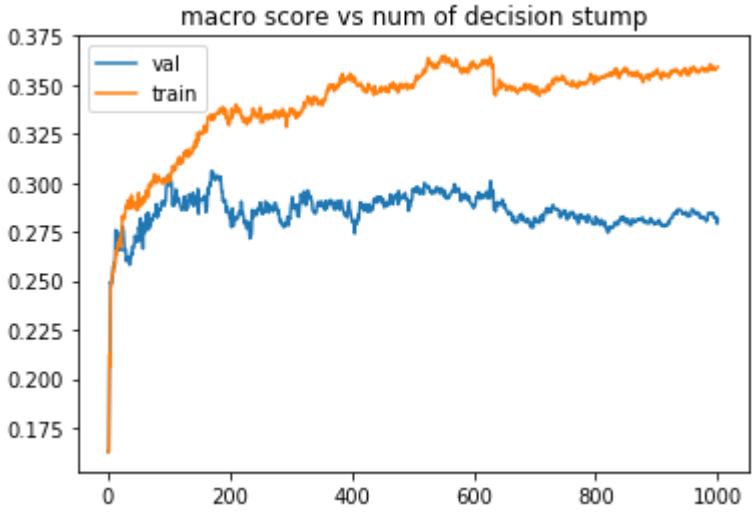
```
# Adaboost with tree method
#import warnings
#warnings.filterwarnings("ignore")
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
print(Counter(y_train))
print(Counter(y_val))
stra = {5.0:15214,
        4.0:10000,
        3.0:2000,
        2.0:1000,
        1.0:1000}
sm = SMOTE(random_state=42, sampling_strategy=stra)
X_res, y_res = sm.fit_resample(ave_vec, y_train)
print(Counter(y_res))
clf = AdaBoostClassifier(n_estimators=1000, random_state=1)
clf.fit(X_res, y_res)
real_val_macro = [0]
train_macro = []
for real_test_predict in clf.staged_predict(ave_vec_val):
    if f1_score(y_val, real_test_predict, average='macro')> np.max(real_val_macro) :
        pred_opt = f1_score(y_val, real_test_predict, average='macro')
    real_val_macro.append(
        f1_score(y_val, real_test_predict, average='macro'))
for real_train_predict in clf.staged_predict(ave_vec):
    train_macro.append(f1_score(y_train, real_train_predict, average='macro'))


n_trees_real = len(clf)
plt.figure()
plt.plot(range(1, n_trees_real + 1),
         real_val_macro[1:],
         label='val')
plt.plot(range(1, n_trees_real + 1),
         train_macro,
         label='train')
```

```
Counter({5.0: 15214, 4.0: 5466, 3.0: 959, 1.0: 263, 2.0: 257})
Counter({5.0: 3804, 4.0: 1367, 3.0: 240, 1.0: 65, 2.0: 64})
Counter({5.0: 15214, 4.0: 10000, 3.0: 2000, 2.0: 1000, 1.0: 1000})
```

Out[71]: [<matplotlib.lines.Line2D at 0x7fcea9b37ac8>]



In [72]:
```
plt.figure()
plt.plot(range(1, n_trees_real + 1),
         real_val_macro[1:],
         label='val')
plt.plot(range(1, n_trees_real + 1),
         train_macro,
         label='train')
plt.legend()
plt.title('macro score vs num of decision stump')
adaboost_opt = AdaBoostClassifier(n_estimators=np.argmax(real_val_macro[1:])+1,
                                  random_state=1)
```



In [73]:
```
pred_opt
```

Out[73]: 0.30599352334303764

In [76]:
```
adaboost_opt.fit(X_res, y_res)
```

Out[76]:
```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=170, random_state=1)
```

In [84]:
```
f1_score(y_val, adaboost_opt.predict(ave_vec_val), average='macro')
```

Out[84]: 0.30599352334303764

In [89]:
```
joblib.dump(adaboost_opt,'../input/model_w2v/w2v_adaboost.pkl')
```

Out[89]: ['../input/model_w2v/w2v_adaboost.pkl']

In [86]:
```
np.argmax(real_val_macro[1:])+1
```

Out[86]: 170

In [88]:
```
w2v_adaboost = adaboost_opt
print('Word2Vec feature: Adaboost, best F1-macro score on validation set:',f1_score(y_val, w2v_adaboost.predict(ave_vec_
mat = confusion_matrix(y_val, w2v_adaboost.predict(ave_vec_val))
print('confusion matrix on validation set is: \n',mat)
print('F1 score of each class:\n',f1_score(y_val, w2v_adaboost.predict(ave_vec_val), average=None))
print('accuracy on validation set:', w2v_adaboost.score(ave_vec_val,y_val),'\n\n')
```

```
Word2Vec feature: Adaboost, best F1-macro score on validation set: 0.30599352334303764
confusion matrix on validation set is:
 [[  18    7    4   15   21]
 [  15    6    2   19   22]
 [  18   14   21  111   76]
 [  28   32   28  429  850]
 [  46   30   26  580 3122]]
F1 score of each class:
 [0.18947368 0.07843137 0.13084112 0.34034113 0.7908803 ]
accuracy on validation set: 0.6490974729241877
```

In [2]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score

df_train = pd.read_csv('../input/train.csv')
train = df_train['review'].to_numpy()
df_y_train = pd.read_csv('../input/y_train.csv')
y_train = df_y_train['rate'].to_numpy()
df_val = pd.read_csv('../input/val.csv')
val = df_val['review'].to_numpy()
df_y_val = pd.read_csv('../input/y_val.csv')
y_val = df_y_val['rate'].to_numpy()

import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
from gensim.test.utils import datapath
from gensim import utils
from pathlib import Path
import gensim
model = gensim.models.Word2Vec.load('../embedding/word2vec.model')
ave_vec = np.zeros((train.shape[0],50),dtype='float')
word_vectors = model.wv
len(word_vectors.vocab)
for k, seq in enumerate(train):
    tokens = gensim.utils.simple_preprocess(seq)
    for i in tokens:
        if i in word_vectors.vocab:
            ave_vec[k] += model.wv[i]
    ave_vec[k] /= len(tokens)
print(ave_vec.shape)
```

...

In [3]:
```
word_vectors = model.wv
len(word_vectors.vocab)
ave_vec_val = np.zeros((val.shape[0],50),dtype='float')
for k, seq in enumerate(val):
    tokens = gensim.utils.simple_preprocess(seq)
    for i in tokens:
        if i in word_vectors.vocab:
            ave_vec_val[k] += model.wv[i]
    ave_vec_val[k] /= np.array(len(tokens))
print(ave_vec_val.shape)
```

```
(5540, 50)
```

In [4]:
```
from imblearn.over_sampling import SMOTE # doctest: +NORMALIZE_WHITESPACE
from collections import Counter
n_alphas = 36
#alphas = np.exp(np.linspace(-5,10,n_alphas))
C = np.exp(np.linspace(-5,10,n_alphas))
print(Counter(y_train))
print(Counter(y_val))
stra = {5.0:15214,
        4.0:10000,
        3.0:2000,
        2.0:1000,
        1.0:1000}
sm = SMOTE(random_state=42, sampling_strategy=stra)
X_res, y_res = sm.fit_resample(ave_vec, y_train)
print(Counter(y_res))

train_score = []
val_score = []
for j,i in enumerate(C):
    m = LogisticRegression(penalty='l2',C=i,class_weight='balanced',n_jobs=-1).fit(X_res, y_res)
    train_pre = m.predict(ave_vec)
    val_pre = m.predict(ave_vec_val)
    train_score.append(f1_score(y_train, train_pre, average='macro'))
    val_score.append(f1_score(y_val, val_pre, average='macro'))

```

...

In [7]:
```
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.externals import joblib
ax = plt.gca()
ax.plot(C, train_score, label='train')
ax.plot(C, val_score, label='val')
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim())  # reverse axis
plt.xlabel('C')
plt.ylabel('weights')
plt.title('logistic regression with L2 penalty(Word2Vec)')
plt.axis('tight')
plt.legend()
plt.show()
C_opt = C[np.argmax(val_score)]
m_opt = LogisticRegression(penalty='l2',C=C_opt,class_weight='balanced',n_jobs=-1).fit(X_res, y_res)
print(f1_score(y_val, m_opt.predict(ave_vec_val), average='macro'))
mat = confusion_matrix(y_val, m_opt.predict(ave_vec_val))
print('confusion matrix is: \n',mat)
print('f1 of each class:\n',f1_score(y_val, m_opt.predict(ave_vec_val), average=None))
joblib.dump(m_opt,'../input/model_w2v/logistic_L2.pkl')
```

...

In [9]:

```
from imblearn.over_sampling import SMOTE # doctest: +NORMALIZE_WHITESPACE
from collections import Counter
n_alphas = 36
#alphas = np.exp(np.linspace(-5,10,n_alphas))
C = np.exp(np.linspace(-5,10,n_alphas))
print(Counter(y_train))
print(Counter(y_val))
stra = {5.0:15214,
        4.0:10000,
        3.0:2000,
        2.0:1000,
        1.0:1000}
sm = SMOTE(random_state=42, sampling_strategy=stra)
X_res, y_res = sm.fit_resample(ave_vec, y_train)
print(Counter(y_res))

train_score = []
val_score = []
for j,i in enumerate(C):
    m = LogisticRegression(penalty='l1',C=i,class_weight='balanced',n_jobs=-1, solver='saga', tol=0.001).fit(X_res, y_
    train_pre = m.predict(ave_vec)
    val_pre = m.predict(ave_vec_val)
    train_score.append(f1_score(y_train, train_pre, average='macro'))
    val_score.append(f1_score(y_val, val_pre, average='macro'))
```

...

In [11]:

```
# SMOTE with L1
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.externals import joblib
ax = plt.gca()
ax.plot(C, train_score, label='train')
ax.plot(C, val_score,label='val')
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim())  # reverse axis
plt.xlabel('C')
plt.ylabel('F1-macro')
plt.title('macro score of training set and validation set')
plt.axis('tight')
plt.legend()
plt.show()
C_opt = C[np.argmax(val_score)]
m_opt = LogisticRegression(penalty='l1',C=C_opt,class_weight='balanced',n_jobs=-1, solver='saga', tol=0.001).fit(X_res,
print(f1_score(y_val, m_opt.predict(ave_vec_val), average='macro'))
mat = confusion_matrix(y_val, m_opt.predict(ave_vec_val))
print('confusion matrix is: \n',mat)
print('F1 score of each class:\n',f1_score(y_val, m_opt.predict(ave_vec_val), average=None))
joblib.dump(m_opt,'../input/model_w2v/logistic_L1.pkl')
```

...

In [19]:
```
# elastic net
from imblearn.over_sampling import SMOTE # doctest: +NORMALIZE_WHITESPACE
from collections import Counter
n_alphas = 36
#alphas = np.exp(np.linspace(-5,10,n_alphas))
C = np.exp(np.linspace(-5,10,n_alphas))
print(Counter(y_train))
print(Counter(y_val))
stra = {5.0:15214,
        4.0:10000,
        3.0:2000,
        2.0:1000,
        1.0:1000}
sm = SMOTE(random_state=42, sampling_strategy=stra)
X_res, y_res = sm.fit_resample(ave_vec, y_train)
print(Counter(y_res))
a = np.linspace(0.2,0.7,6)
train_score = [[],[],[],[],[],[]]
val_score = [[],[],[],[],[],[]]
for n,u in enumerate(a):
    for j,i in enumerate(C):
        m = LogisticRegression(penalty='elasticnet',C=i,class_weight='balanced',n_jobs=-1, l1_ratio=u,solver='saga').f
        train_pre = m.predict(ave_vec)
        val_pre = m.predict(ave_vec_val)
        train_score[n].append(f1_score(y_train, train_pre, average='macro'))
        val_score[n].append(f1_score(y_val, val_pre, average='macro'))

```

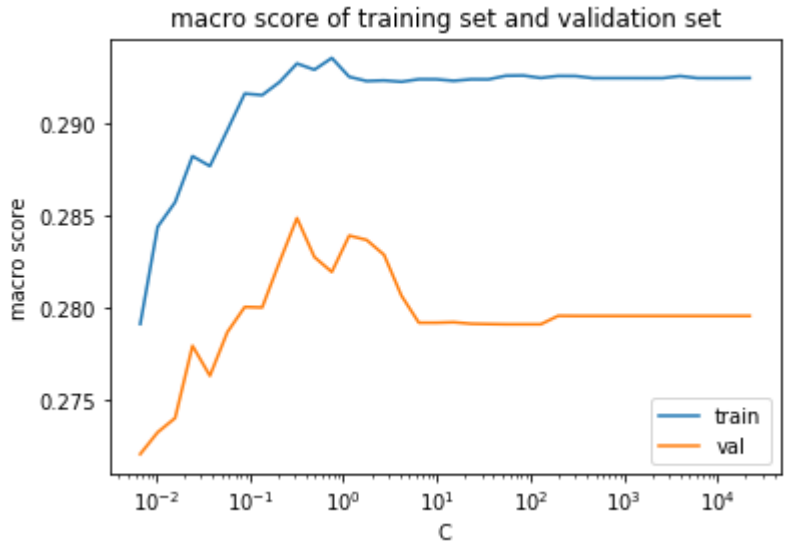
...

In [29]:
```
elastic_opt = LogisticRegression(penalty='elasticnet',C=C[(np.argmax(val_score))%36],class_weight='balanced',n_jobs=-1,
```

In [30]:
```
elastic_opt.fit(X_res,y_res)
print('the best macro score on validation set:',f1_score(y_val, elastic_opt.predict(ave_vec_val), average='macro'))
mat = confusion_matrix(y_val, elastic_opt.predict(ave_vec_val))
print('confusion matrix is: \n',mat)
print('accuracy of each class:\n',f1_score(y_val, elastic_opt.predict(ave_vec_val), average=None))
```

```
/home/zixi/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:469: FutureWarning: Default multi_class
will be changed to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)

the best macro score on validation set: 0.2848942737772833
confusion matrix is:
 [[  28   20    9    2    6]
 [  19   34    6    3    2]
 [  32   65   76   43   24]
 [  88  161  263  374  481]
 [ 242  230  286  667 2379]]
accuracy of each class:
 [0.11814346 0.1184669  0.17272727 0.30456026 0.71057348]
```

In [33]:
```
train_score = np.array(train_score)
val_score = np.array(val_score)
ax1 = plt.gca()
# randomly choose 50 coefficients in the 9676 variables
# to plot figure of coefs vs regularizer.
ax1.plot(C, train_score[(np.argmax(val_score))//36], label='train')
ax1.plot(C, val_score[(np.argmax(val_score))//36],label='val')
ax1.set_xscale('log')
#ax1.set_xlim(ax1.get_xlim()[::-1])  # reverse axis
plt.xlabel('C')
plt.ylabel('macro score')
plt.title('macro score of training set and validation set')
plt.axis('tight')
plt.legend()
plt.show()
```



In [27]:
```
val_score[(np.argmax(val_score))//36][(np.argmax(val_score))%36]
```

Out[27]: 0.2848942737772833

In [34]:
```
1 joblib.dump(elastic_opt,'../input/model_w2v/logistic_elastic.pkl')
```

Out[34]: ['../input/model_w2v/logistic_elastic.pkl']

In [ ]:
```
1
```

In [ ]:
```
1
```