In [110]:
```python
import pandas as pd
import numpy as np
```

Task 1 Bag of Words and simple Features

Linear model with non-review based features and L2 regularisation

In [111]:
```python
df = pd.read_csv('data/train.csv')
```

In [112]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 89970 entries, 0 to 89969
Data columns (total 14 columns):
Id                       89970 non-null int64
country                  89927 non-null object
description              89970 non-null object
designation              64086 non-null object
points                   89970 non-null int64
price                    83828 non-null float64
province                 89927 non-null object
region_1                 75108 non-null object
region_2                 34869 non-null object
taster_name              71843 non-null object
taster_twitter_handle    68373 non-null object
title                    89970 non-null object
variety                  89970 non-null object
winery                   89970 non-null object
dtypes: float64(1), int64(2), object(11)
memory usage: 9.6+ MB
```

In [5]:
```python
df.head()
```

Out[5]:

	Id	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle	title
0	69712	France	A buoyant wine, this has delicious acidity and...	La Riviera	89	17.0	Provence	Côtes de Provence	NaN	Roger Voss	@vossroger	Domai de Sanglie 2015 l Rivie Rosé (
1	1171	Germany	Despite weighing in at 12% alcohol, this seems...	Niersteiner Bergkirche Kabinett	86	20.0	Rheinhessen	NaN	NaN	Joe Czerwinski	@JoeCz	Lou Guntru 200 Nierstein Bergkirch Kabi
2	129181	US	Sweet, simple and fruity, this light-bodied wi...	Estate	84	23.0	California	Sierra Foothills	Sierra Foothills	Jim Gordon	@gordone_cellars	Naggi 201 Esta Musca Cane (Sier Foo
3	116814	US	Tastes like a sweet and sour sauce mixed into ...	NaN	84	19.0	California	California	California Other	NaN	NaN	One Ho 200 Sauvign Blan (Californ
4	116519	Spain	As a one-liter package you get one-third more ...	NaN	86	9.0	Central Spain	Vino de la Tierra de Castilla	NaN	Michael Schachner	@wineschach	Bodega Egur 200 Tempranil (Vino de Ti

In [113]:
```python
def dummy(df,col):
    drop = np.unique(df[col])[-1]
    dummy = pd.get_dummies(df[col])
    new_col = []
    for i in dummy.columns:
        new_col.append(col+'_'+str(i))
    dummy.columns = new_col
    df = pd.concat([df, dummy], axis=1)
    df.drop([col, new_col[-1]], inplace=True, axis=1)
    return df
```

In [114]:
```python
df.columns
```

Out[114]:
```
Index(['Id', 'country', 'description', 'designation', 'points', 'price',
       'province', 'region_1', 'region_2', 'taster_name',
       'taster_twitter_handle', 'title', 'variety', 'winery'],
      dtype='object')
```

In [115]:
```python
df['country'].fillna(df['country'].mode()[0], inplace=True)
df['designation'].fillna(df['designation'].mode()[0], inplace=True)
df['province'].fillna(df['province'].mode()[0], inplace=True)
df['region_1'].fillna(df['region_1'].mode()[0], inplace=True)
df['region_2'].fillna(df['region_2'].mode()[0], inplace=True)
df['taster_name'].fillna(df['taster_name'].mode()[0], inplace=True)
df['taster_twitter_handle'].fillna(df['taster_twitter_handle'].mode()[0], inplace=True)
```

In [116]:
```python
df['price'].fillna(df['price'].median(), inplace=True)
```

In [117]:
```python
cols = [ 'country', 'taster_name', 'variety']
```

In [118]:
```python
empty = []
for col in cols:
    try:
        df = dummy(df,col)
    except:
        empty.append(col)
```

In [119]:
```python
empty
```

Out[119]:
```
[]
```

In [120]:
```python
y = df['points'].values
```

In [121]:
```python
df.drop(['Id',
'description',
'designation',
'points',
'province',
'region_1',
'region_2',
'taster_twitter_handle',
'title',
'winery'],axis=1,inplace=True)
```

In [184]:
```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df_scaled = scaler.fit_transform(df)

df_scaled1 = pd.DataFrame(df_scaled, columns = df.columns)
```

In [185]:
```python
df_sub = df.iloc[:,(df_scaled1.var() > 0.02).values]
```

In [ ]:
```python
(df_scaled1.var() > 0.02).values
```

In [186]:
```python
df_sub['price'] = df['price']
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#index
ing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

In [187]:
```python
df_sub.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 89970 entries, 0 to 89969
Data columns (total 31 columns):
country_Argentina                   89970 non-null uint8
country_Austria                     89970 non-null uint8
country_Chile                       89970 non-null uint8
country_France                      89970 non-null uint8
country_Italy                       89970 non-null uint8
country_Portugal                    89970 non-null uint8
country_Spain                       89970 non-null uint8
country_US                          89970 non-null uint8
taster_name_Anna Lee C. Iijima      89970 non-null uint8
taster_name_Anne Krebiehl MW        89970 non-null uint8
taster_name_Jim Gordon              89970 non-null uint8
taster_name_Joe Czerwinski          89970 non-null uint8
taster_name_Kerin O'Keefe           89970 non-null uint8
taster_name_Matt Kettmann           89970 non-null uint8
taster_name_Michael Schachner       89970 non-null uint8
taster_name_Paul Gregutt            89970 non-null uint8
taster_name_Roger Voss              89970 non-null uint8
taster_name_Sean P. Sullivan        89970 non-null uint8
variety_Bordeaux-style Red Blend    89970 non-null uint8
variety_Cabernet Sauvignon          89970 non-null uint8
variety_Chardonnay                  89970 non-null uint8
variety_Merlot                      89970 non-null uint8
variety_Nebbiolo                    89970 non-null uint8
variety_Pinot Noir                  89970 non-null uint8
variety_Red Blend                   89970 non-null uint8
variety_Riesling                    89970 non-null uint8
variety_Rosé                        89970 non-null uint8
variety_Sangiovese                  89970 non-null uint8
variety_Sauvignon Blanc             89970 non-null uint8
variety_Syrah                       89970 non-null uint8
price                               89970 non-null float64
dtypes: float64(1), uint8(30)
memory usage: 3.3 MB
```

In [126]:
```python
X = df_sub.values
```

In [127]:
```python
from sklearn.linear_model import Ridge

clf = Ridge(alpha = 0.0, normalize = False)
clf.fit(X, y)
```

Out[127]:
```
Ridge(alpha=0.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

In [128]:
```python
clf.score(X, y)
```

Out[128]:
```
0.23772346398423183
```

In [129]:
```python
alphas = 10**np.linspace(10,-2,100)*0.5
alphas
```

Out[129]:
```
array([5.00000000e+09, 3.78231664e+09, 2.86118383e+09, 2.16438064e+09,
       1.63727458e+09, 1.23853818e+09, 9.36908711e+08, 7.08737081e+08,
       5.36133611e+08, 4.05565415e+08, 3.06795364e+08, 2.32079442e+08,
       1.75559587e+08, 1.32804389e+08, 1.00461650e+08, 7.59955541e+07,
       5.74878498e+07, 4.34874501e+07, 3.28966612e+07, 2.48851178e+07,
       1.88246790e+07, 1.42401793e+07, 1.07721735e+07, 8.14875417e+06,
       6.16423370e+06, 4.66301673e+06, 3.52740116e+06, 2.66834962e+06,
       2.01850863e+06, 1.52692775e+06, 1.15506485e+06, 8.73764200e+05,
       6.60970574e+05, 5.00000000e+05, 3.78231664e+05, 2.86118383e+05,
       2.16438064e+05, 1.63727458e+05, 1.23853818e+05, 9.36908711e+04,
       7.08737081e+04, 5.36133611e+04, 4.05565415e+04, 3.06795364e+04,
       2.32079442e+04, 1.75559587e+04, 1.32804389e+04, 1.00461650e+04,
       7.59955541e+03, 5.74878498e+03, 4.34874501e+03, 3.28966612e+03,
       2.48851178e+03, 1.88246790e+03, 1.42401793e+03, 1.07721735e+03,
       8.14875417e+02, 6.16423370e+02, 4.66301673e+02, 3.52740116e+02,
       2.66834962e+02, 2.01850863e+02, 1.52692775e+02, 1.15506485e+02,
       8.73764200e+01, 6.60970574e+01, 5.00000000e+01, 3.78231664e+01,
       2.86118383e+01, 2.16438064e+01, 1.63727458e+01, 1.23853818e+01,
       9.36908711e+00, 7.08737081e+00, 5.36133611e+00, 4.05565415e+00,
       3.06795364e+00, 2.32079442e+00, 1.75559587e+00, 1.32804389e+00,
       1.00461650e+00, 7.59955541e-01, 5.74878498e-01, 4.34874501e-01,
       3.28966612e-01, 2.48851178e-01, 1.88246790e-01, 1.42401793e-01,
       1.07721735e-01, 8.14875417e-02, 6.16423370e-02, 4.66301673e-02,
       3.52740116e-02, 2.66834962e-02, 2.01850863e-02, 1.52692775e-02,
       1.15506485e-02, 8.73764200e-03, 6.60970574e-03, 5.00000000e-03])
```

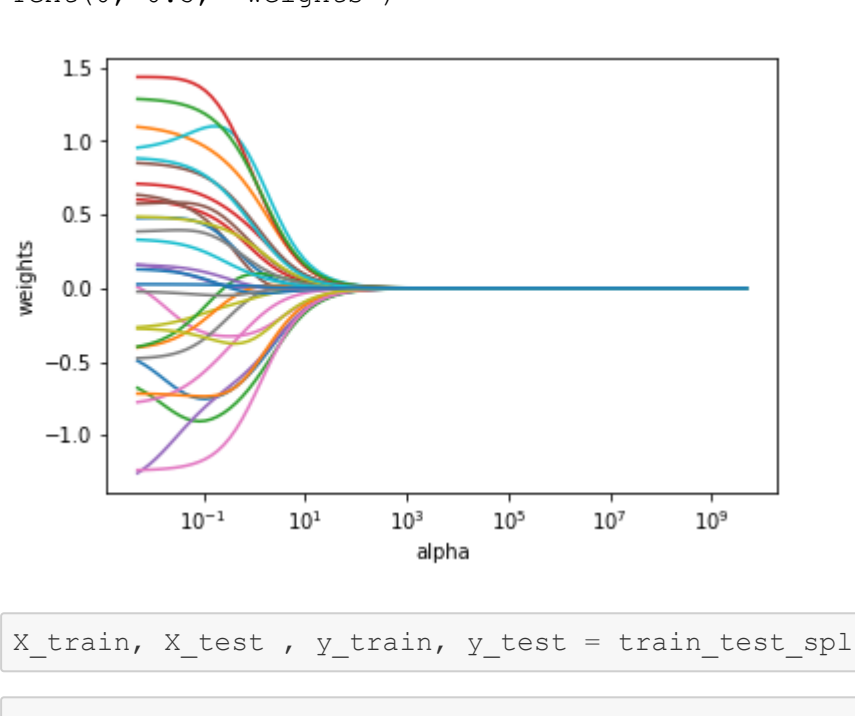
In [130]:
```python
ridge = Ridge(normalize = True)
coefs = []

for a in alphas:
    ridge.set_params(alpha = a)
    ridge.fit(X, y)
    coefs.append(ridge.coef_)

np.shape(coefs)
```

Out[130]:
```
(100, 31)
```

In [131]:
```python
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
from sklearn.metrics import mean_squared_error
```

In [132]:
```python
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('weights')
```

Out[132]:
```
Text(0, 0.5, 'weights')
```



In [176]:
```python
X_train, X_test , y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

In [177]:
```python
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
```

Out[177]:
```
0.020185086292982747
```

In [178]:
```python
ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)
mean_squared_error(y_test, ridge.predict(X_test))
```

Out[178]:
```
6.917063528557935
```

In [179]:
```python
ridge.score(X_test, y_test)
```

Out[179]:
```
0.24470745847603645
```

In [198]:
```python
public = pd.read_csv('data/public.csv')
```

In [213]:
```python
private = pd.read_csv('data/private.csv')
```

Predicting the non-review based model on private dataset

In [169]:
```python
private['country'].fillna(private['country'].mode()[0], inplace=True)
private['designation'].fillna(private['designation'].mode()[0], inplace=True)
private['province'].fillna(private['province'].mode()[0], inplace=True)
private['region_1'].fillna(private['region_1'].mode()[0], inplace=True)
private['region_2'].fillna(private['region_2'].mode()[0], inplace=True)
private['taster_name'].fillna(private['taster_name'].mode()[0], inplace=True)
private['taster_twitter_handle'].fillna(private['taster_twitter_handle'].mode()[0], inplace=True)

private['price'].fillna(private['price'].median(), inplace=True)

cols = [ 'country', 'taster_name', 'variety']

empty = []
for col in cols:
    try:
        private = dummy(private,col)
    except:
        empty.append(col)

empty
```

Out[169]:
```
[]
```

In [171]:
```python
y_private = private['points'].values
```

In [170]:
```python
df_sub.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 89970 entries, 0 to 89969
Data columns (total 31 columns):
country_Argentina                   89970 non-null uint8
country_Austria                     89970 non-null uint8
country_Chile                       89970 non-null uint8
country_France                      89970 non-null uint8
country_Italy                       89970 non-null uint8
country_Portugal                    89970 non-null uint8
country_Spain                       89970 non-null uint8
country_US                          89970 non-null uint8
taster_name_Anna Lee C. Iijima      89970 non-null uint8
taster_name_Anne Krebiehl MW        89970 non-null uint8
taster_name_Jim Gordon              89970 non-null uint8
taster_name_Joe Czerwinski          89970 non-null uint8
taster_name_Kerin O'Keefe           89970 non-null uint8
taster_name_Matt Kettmann           89970 non-null uint8
taster_name_Michael Schachner       89970 non-null uint8
taster_name_Paul Gregutt            89970 non-null uint8
taster_name_Roger Voss              89970 non-null uint8
taster_name_Sean P. Sullivan        89970 non-null uint8
variety_Bordeaux-style Red Blend    89970 non-null uint8
variety_Cabernet Sauvignon          89970 non-null uint8
variety_Chardonnay                  89970 non-null uint8
variety_Merlot                      89970 non-null uint8
variety_Nebbiolo                    89970 non-null uint8
variety_Pinot Noir                  89970 non-null uint8
variety_Red Blend                   89970 non-null uint8
variety_Riesling                    89970 non-null uint8
variety_Rosé                        89970 non-null uint8
variety_Sangiovese                  89970 non-null uint8
variety_Sauvignon Blanc             89970 non-null uint8
variety_Syrah                       89970 non-null uint8
price                               89970 non-null float64
dtypes: float64(1), uint8(30)
memory usage: 3.3 MB
```

In [191]:
```python
df_sub_private = private[df_sub.columns]

df_sub_private.info()

X_private = df_sub_private.values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 31 columns):
country_Argentina                   20000 non-null uint8
country_Austria                     20000 non-null uint8
country_Chile                       20000 non-null uint8
country_France                      20000 non-null uint8
country_Italy                       20000 non-null uint8
country_Portugal                    20000 non-null uint8
country_Spain                       20000 non-null uint8
country_US                          20000 non-null uint8
taster_name_Anna Lee C. Iijima      20000 non-null uint8
taster_name_Anne Krebiehl MW        20000 non-null uint8
taster_name_Jim Gordon              20000 non-null uint8
taster_name_Joe Czerwinski          20000 non-null uint8
taster_name_Kerin O'Keefe           20000 non-null uint8
taster_name_Matt Kettmann           20000 non-null uint8
taster_name_Michael Schachner       20000 non-null uint8
taster_name_Paul Gregutt            20000 non-null uint8
taster_name_Roger Voss              20000 non-null uint8
taster_name_Sean P. Sullivan        20000 non-null uint8
variety_Bordeaux-style Red Blend    20000 non-null uint8
variety_Cabernet Sauvignon          20000 non-null uint8
variety_Chardonnay                  20000 non-null uint8
variety_Merlot                      20000 non-null uint8
variety_Nebbiolo                    20000 non-null uint8
variety_Pinot Noir                  20000 non-null uint8
variety_Red Blend                   20000 non-null uint8
variety_Riesling                    20000 non-null uint8
variety_Rosé                        20000 non-null uint8
variety_Sangiovese                  20000 non-null uint8
variety_Sauvignon Blanc             20000 non-null uint8
variety_Syrah                       20000 non-null uint8
price                               20000 non-null float64
dtypes: float64(1), uint8(30)
memory usage: 742.3 KB
```

In [192]:
```python
mean_squared_error(y_private, ridge.predict(X_private))

ridge.score(X_private, y_private)
```

Out[192]:
```
0.24936179420755233
```

Predicting the non-review based model on public dataset

In [200]:
```python
public['country'].fillna(public['country'].mode()[0], inplace=True)
public['designation'].fillna(public['designation'].mode()[0], inplace=True)
public['province'].fillna(public['province'].mode()[0], inplace=True)
public['region_1'].fillna(public['region_1'].mode()[0], inplace=True)
public['region_2'].fillna(public['region_2'].mode()[0], inplace=True)
public['variety'].fillna(public['variety'].mode()[0], inplace=True)
public['taster_name'].fillna(public['taster_name'].mode()[0], inplace=True)
public['taster_twitter_handle'].fillna(public['taster_twitter_handle'].mode()[0], inplace=True)

public['price'].fillna(public['price'].median(), inplace=True)

cols = ['country', 'taster_name', 'variety']

empty = []
for col in cols:
    try:
        public = dummy(public,col)
    except:
        empty.append(col)

empty
```

Out[200]:
```
[]
```

In [201]:
```python
y_public = public['points'].values

df_sub_public = public[df_sub.columns]

df_sub_public.info()

X_public = df_sub_public.values

mean_squared_error(y_public, ridge.predict(X_public))

ridge.score(X_public, y_public)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20001 entries, 0 to 20000
Data columns (total 31 columns):
country_Argentina                   20001 non-null uint8
country_Austria                     20001 non-null uint8
country_Chile                       20001 non-null uint8
country_France                      20001 non-null uint8
country_Italy                       20001 non-null uint8
country_Portugal                    20001 non-null uint8
country_Spain                       20001 non-null uint8
country_US                          20001 non-null uint8
taster_name_Anna Lee C. Iijima      20001 non-null uint8
taster_name_Anne Krebiehl MW        20001 non-null uint8
taster_name_Jim Gordon              20001 non-null uint8
taster_name_Joe Czerwinski          20001 non-null uint8
taster_name_Kerin O'Keefe           20001 non-null uint8
taster_name_Matt Kettmann           20001 non-null uint8
taster_name_Michael Schachner       20001 non-null uint8
taster_name_Paul Gregutt            20001 non-null uint8
taster_name_Roger Voss              20001 non-null uint8
taster_name_Sean P. Sullivan        20001 non-null uint8
variety_Bordeaux-style Red Blend    20001 non-null uint8
variety_Cabernet Sauvignon          20001 non-null uint8
variety_Chardonnay                  20001 non-null uint8
variety_Merlot                      20001 non-null uint8
variety_Nebbiolo                    20001 non-null uint8
variety_Pinot Noir                  20001 non-null uint8
variety_Red Blend                   20001 non-null uint8
variety_Riesling                    20001 non-null uint8
variety_Rosé                        20001 non-null uint8
variety_Sangiovese                  20001 non-null uint8
variety_Sauvignon Blanc             20001 non-null uint8
variety_Syrah                       20001 non-null uint8
price                               20001 non-null float64
dtypes: float64(1), uint8(30)
memory usage: 742.3 KB
```

Out[201]:
```
0.2500127079784289
```

linear model using review based features and Count vectoriser and L2 regularisation

In [33]:
```python
import re # for regular expressions
import pandas as pd
pd.set_option("display.max_colwidth", 200)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
import nltk # for text manipulation
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

%matplotlib inline
```

In [34]:
```python
combi = pd.read_csv('data/train.csv')
```

In [35]:
```python
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for i in r:
        input_txt = re.sub(i, '', input_txt)

    return input_txt
```

In [ ]:
```python
combi['description'] = np.vectorize(remove_pattern)(combi['description'], "@[\w]*")
```

In [ ]:
```python
combi['description'] = combi['description'].str.replace("[^a-zA-Z#]", " ")
```

In [ ]:
```python
combi['description'] = combi['description'].str.replace('#','')
```

In [38]:
```python
combi['description'] = combi['description'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>2
]))
```

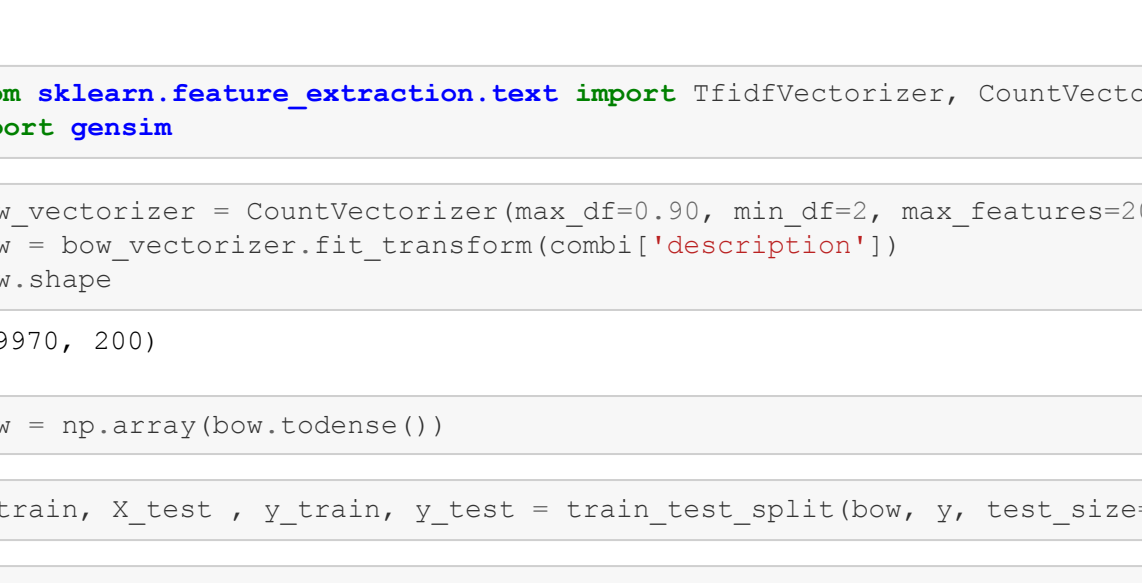
In [40]:
```python
tokenized_tweet = combi['description'].apply(lambda x: x.split()) # tokenizing
```

In [42]:
```python
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
tokenized_tweet.apply(lambda x: [lemmatizer.lemmatize(i) for i in x]) # stemming
```

Out[42]:
```
0        [buoyant, wine,, this, ha, delicious, acidity, and, soft,, immediately, accessible, textu
re., Red, berry, give, lively,, perfumed, fruitiness, this, open, wine, that, smell, and, taste, suns
hine.]
1        [Despite, weighing, 12%, alcohol,, this, seems, light, body, and, sweet, style,, with, hon
eyed, pineapple, aroma, and, flavor, apple, skin, and, lime., Finishes, stridently,, with, dusty, tex
ture.]
2                                                                              [Sweet,, simple,
and, fruity,, this, light-bodied, wine, smell, floral,, taste, appley, and, feel, soft, and, sugary,
sip.]
3                                                                       [Tastes, like, sweet, and,
sour, sauce, mixed, into, white, wine,, with, pineapple,, sugar, and, lime, flavor, and, alcoholic, b
uzz.]
4        [one-liter, package, you, get, one-third, more, wine, here, for, the, money,, which, only, c
ement, this, Tempranillo's, Best, Buy, credentials., It's, little, roasted, the, nose,, but, doesn't,
fa...
5        [This, really, drinkable, Pinot, Noir., It's, dry, and, silky, and, crisp,, and, the, flavo
r, are, enormously, rich,, suggesting, ripe, cherries,, raspberry, and, blackberry, baked, into, pi
e., Dr...
6        [Structured, and, elegant,, this, open, with, aroma, yellow, flower,, acacia,, Mediterranea
n, herb, and, just, slight, whiff, petrol., The, palate, delivers, mature, green, apple,, citrus,, mi
nera...
7        [Staunch, but, murky, aroma, coffee, grounds,, leafy, berry,, olive, and, lemon, peel, annou
nce, tight, raw, palate., Flavors, olive,, plum, skin, and, herb, finish, with, reprise, spent, coffe
e, ...
8                                                                                          [goo
d, everyday, Merlot, that's, dry, and, softly, tannic,, with, pleasant, cherry,, cola, and, spice, fl
avors.]
9        [You'll, love, this, you, like, your, Zins, sweet, and, intensely, fruity., show, flavor, ch
ocolate-covered, raisins,, blackberry, jam, and, savory, spices,, with, high, alcohol, lending, hea
t., G...
10       [Compelling, scent, acacia, flower,, beeswax, and, yellow, apple, skin, waft, out, the, glas
s., The, vibrant, palate, offer, white, peach,, pear, and, hint, citrus, zest, alongside, crisp, acid
ity...
11       [The, aroma, are, generous,, with, note, freshly, cut, wood,, custard, and, herb., The, pala
te, full, bodied,, rich, and, creamy, feel., The, wood, show, prominently, and, brings, warmth, the,
fin...
12       [This, open, with, scent, orchard, fruit,, citrus, and, touch, stone., The, aroma, float, ov
er, the, light-bodied,, straightforward, palate, along, with, hint, citrus., Bracing, acidity, give,
rac...
13       [Whiffs, saffron, and, spice, lend, earthen, complexity, cutting, lime, and, green, plum, th
is, zesty, Riesling., Off, dry, the, palate,, it's, delicately, sheer, and, spry., Its, invigorating,
st...
14       [Fermented, neutral, oak, barrel, and, aged, sur, lie, for, months,, this, wine,, which, inc
ludes, Viognier,, brings, herb,, honeycomb,, pear, and, orange, peel, aromas., drink, dry,, with, sle
ek,...
15       [Intense, citrus, note, perfume, this, bright,, sunny, off-dry, Riesling., Yellow, peach, an
d, pear, flavor, are, rich, and, ripe, the, palate,, but, balanced, firm, strike, lemon, acidity., Fi
nis...
16       [Unoaked, Chardonnays, can, fairly, neutral,, but, this, one, anything, but., Creamy, and, t
extural,, it's, laced, with, mixed, flavor, grapefruit,, gooseberry,, lime, and, lemon, zest., Delici
ous...
17       [For, winemaking, family, from, the, Loire,, must, delight, work, Marlborough's, almost, unf
ailing, sunshine., This, medium-bodied,, silky, Pinot, with, scent, compost., Add, that, gentle, eart
hin...
18       [This, pure, Merlot, racy,, ripe, and, ready, drink,, with, fruit, that, show, maturity., Ta
nnins, are, well, integrated, into, the, structure,, leaving, plenty, room, for, the, juicy, fruit, a
nd,...
19       [This, exotic, white, blend, from, Iowa, offer, aroma, orange, and, citrus, with, minerally,
zip,, and, the, palate,, bit, the, sweet, and, curvy, side., Not, terribly, inspired, but, still, pre
tt...
20       [The, tannin, this, vineyard, designate,, where, two, clone, Cab, are, grown, (337, and, 19
1), are, still, dry, and, thick, and, the, wine, well, made,, with, herbal,, baked, bread, and, rich,
red...
21       [More, than, year, making, dry, Alsatian, wine, the, Edna, Valley, are, display, this, class
ic, bottling., The, varietal's, familiar, gasoline, nose, pick, lightness, from, fresh, apple,, cris
p, n...
22       [This, dark,, intensely, astringent, wine, with, much, grape, and, oak, tannin, that, it's,
challenging, enjoy, now., Since, known, grape, that, provides, color, and, tannin, Bordeaux, wines,,
it'...
23       [Although, this, wine, listed, white, blend,, the, producer, say, the, variety, Vernaccia,,
not, found, often, California., Light, alcohol, and, approach,, open, bouquet, waxy, apple, and, Meye
r, ...
24       [Aromas, lemon, zest,, lemongrass, and, lime, juice, prepare, the, palate, for, flavor, fres
hly, squeezed, grapefruit, juice,, tart, apple, and, white, peach., The, finish, brisk, and, pleasant
ly,...
25                                                                                 [This, straigh
tforward, wine, ha, note, fresh, apple, and, pear., The, flavor, are, light, style,, seeming, bit, di
lute.]
26       [spicy, oak, aroma,, fresh, red, cherry, and, cranberry, flavor, and, brisk,, slightly, tann
ic, texture, make, this, wine, quite, appetizing., While, medium, bodied,, ha, plenty, flavor, impac
t, w...
27       [Weil's, Estate, Dry, Riesling, lighter, weight, than, one, might, expect,, with, hint, pine
apple, and, passion, fruit, the, nose, and, tightly, focused, apple, and, citrus, note, the, palate.,
Cr...
28                          [Smells, grassy,, jumpy, and, foxy,, and, while, sport, decent, feel, the, pala
te,, it's, herbal, and, tomatoey, term, flavor,, with, vegetal, undertone., Green, and, oaky, the, fi
nish.]
29       [All, four, Dão's, indigenous, red, grapes—Touriga, Nacional,, Tinta, Roriz,, Jaen, and, Alf
rocheiro—are, this, blend., rich, and, fruity, wine,, showing, lightly, perfumed, character, against,
ri...

          ...

89940    [This, tasty, wine, ha, vivid, cherry,, strawberry,, candylike, aroma, and, flavors., Mediu
m, bodied,, offer, mouthfeel, that's, almost, syrupy,, balance, that's, slightly, sweet, and, easy-dr
inki...
89941    [California's, Cameron, Hughes, dip, toe, into, Oregon, water, with, this, quite, drinkable,
Willamette, Valley, Pinot., float, light, cherry, fruit, around, some, woody,, herbal,, stiff, tannin
s....
89942                                      [Pretty, complex, for, unoaked, Chardonnay,, with, hint, anise,, c
innamon, and, yogurt, accenting, apple, and, citrus, flavors., It's, plump, the, mouth,, and, easy, d
rink.]
89943    [The, catch-all, nose, offer, grass,, grapefruit,, flint,, vanilla, and, baking, spices., Wh
at, follows, tasty, blend, citrus,, green, herbs,, Granny, Smith, apple, and, welcome, chalkiness., R
oun...
89944    [Hints, earthy, spice, and, bark, add, interest, the, somewhat, muted, blackberry, and, blac
k, plum, aromas., The, wine, open, more, the, palate,, revealing, lush, and, creamy, texture,, with,
den...
89945    [Spicy, plum,, berry, and, herbal, aroma, cover, the, bases., This, Cabernet, Sauvignon-Syra
h, blend, drawing, and, pulling, the, palate., Raw,, spicy, flavor, toasty, plum, and, herb, finish,
wit...
89946    [Red, rose,, mature, wild, berry,, dark, cooking, spice, and, balsamic, note, are, some, th
e, aroma, you'll, find, this, gorgeous, wine., The, structured,, elegant, palate, delivers, juicy, bl
ack,...
89947    [Loaded, with, smoke,, hickory, and, vanilla,, this, definitely, show, plenty, oak,, but, th
ere's, also, plenty, fruit, balance, the, ledger., Cola, and, blackberry, flavor, pick, slightly, tar
t, ...
89948    [This, wine, show, superb, balance, between, ripe, fruit, tones,, attractive, oak, influenc
e, and, natural, freshness., lead, with, scent, lightly, toasted, nuts,, orange, cream,, freshly, dri
ed, ...
89949    [rich,, ripe, and, complex, Cabernet,, with, classic, Napa, tannic, structure., Not, quite,
there, with, the, winery's, higher-priced, Dollarhide, Elu, bottlings,, but, showing, similar, elegan
ce,...
89950    [While, fresh, and, fruity, aroma,, this, feel, full, bodied,, taste, rich, but, not, swee
t,, and, ha, smooth, texture., Aromas, like, apple, and, banana, tone, down, milder, spicy,, baked-ap
ple, ...
89951    [Sweet, fruit, dominates, this, attractive,, spicy, wine., ha, ample, acidity,, giving, fin
e,, fresh, black-currant, character, and, tannin, that, are, already, melding, into, the, wine., For,
med...
89952    [This, straightforward, and, genuine, Barolo, with, deep, layer, bright, fruit,, spice, and,
mineral, packaged, seamlessly, together, whole., the, mouth,, the, wine, clean, and, elegant, with, d
ry...
89953    [Not, quite, there, with, the, '05,, but, certainly, better, than, the, '04,, this, wine, sh
ow, softly, ripe, flavor, cherries,, raspberry, and, red, currants,, with, complexity, mocha,, vanill
a,...
89954    [Pronounced, note, butter,, cheese, and, canned, cream, corn, dominate, this, savory, Chardo
nnay., It's, rather, fat,, cream-textured, wine, but, sunny, tangerine, acidity, and, fresher, note,
app...
89955    [Among, bone-dry, Cavas,, this, one, the, best, Spain., It's, focused, pastry, and, pear, ar
oma, lead, directly, onto, the, tangy,, citric, palate., The, flavor, orange,, grapefruit, and, miner
al,...
89956    [first, this, come, across, being, bit, heavy,, yeasty, and, tight., With, time,, however,,
this, should, full, bramble-fruit, flavors., This, well-crafted, wine,, with, structure, and, eleganc
e, ...
89957                         [Intense,, concentrated, wine, with, herbaceous, flavor, allied, gooseberry,, p
ink, grapefruit, and, steely, edge, acidity., With, it, powerful, fruits,, worth, aging, for, few, mo
nths.]
89958    [This, smooth,, creamy, wine,, with, apple, and, pear, aroma, and, flavors,, well, crisp, te
xture., It's, full-bodied,, very, fruity, and, ready, drink,, with, bursting, freshness, and, lingeri
ng,...
89959             [you, like, that, slightly, sour,, green, hit, gooseberry, your, Sauvignon, Blanc,, th
is, one's, for, you., Beyond, that,, it's, dry,, and, offer, citrus,, green, apple, and, vanilla, fla
vors.]
89960    [Aromas, cooking, spice,, game, and, mature, black, fruit, waft, out, the, glass., The, spic
y,, easy-drinking, palate, offer, dried, blackberry,, peppercorn, and, clove, alongside, velvety, tan
nins.]
89961    [For, wine, with, only, alcohol,, this, surprisingly, full, bodied,, with, plenty, weight, a
nd, richness., bit, kerosene, mark, the, nose,, alongside, scent, spice, and, bergamot,, while, the,
fla...
89962    [Brisk, acidity, and, minerally,, with, tantalizing, scour, steel,, this, Riesling, clean, a
nd, stimulating., With, green, apple,, citrus, and, floral, flavors,, straddle, the, boundary, betwee
n, ...
89963    [Pretty, hibiscus, nectar, and, lychee, note, add, exotic, twist, this, dainty, late-harves
t, Riesling., Light, it, feet,, it's, deliciously, juicy, the, palate, with, shower, pink, grapefrui
t, an...
89964    [Rusty, rose, color,, this, Southern, California, stab, Sherry, quite, rewarding,, showing,
caramelized, pecan, and, almonds,, light, tobacco, leaf, and, touch, iodine, the, extremely, nutty, n
ose...
89965    [Plump, red, and, black, fruit, immediately, evident, this, wine., Plum,, black, cherry, an
d, forest, berry, come, with, touch, licorice, and, cinnamon, spice., The, palate, mellow, and, suppl
e, ...
89966    [Green, and, vegetal, aroma, and, flavors., The, nose, deal, pea,, green, bean, and, asparag
us, tempered, lime, and, melon., The, palate, ha, weight, and, mix, green,, grassy, flavor, well, app
le,...
89967    [Named, after, the, Pombal,, dovecote, Vesúvio,, this, soft,, ready-to-drink, wine., ha, att
ractive, acidity, and, black, currant, flavors., Layers, tannin, offer, support, while, not, suggesti
ng,...
89968    [Produced, from, vine, with, average, age, years,, this, wine, wa, aged, lee, until, bottlin
g., That, give, extra, richness, the, ripe, green, and, white, fruits., Laced, with, citrus, flavor
s,, t...
89969    [Plump, red, cherry, note, meet, clove,, allspice, and, dried, ginger, the, nose, this, bott
ling., Though, light, body,, there's, decent, grip, the, palate,, where, candied, cherry, and, cranbe
rry...
Name: description, Length: 89970, dtype: object
```

In [168]:
```python
for i in range(len(tokenized_tweet)):
    tokenized_tweet[i] = ' '.join(tokenized_tweet[i])

combi['description'] = tokenized_tweet
```

In [43]:
```python
all_words = ' '.join([text for text in combi['description']])
from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(all_words)

plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```



In [44]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import gensim
```

In [217]:
```python
bow_vectorizer = CountVectorizer(max_df=0.90, min_df=2, max_features=200, stop_words='english')
bow = bow_vectorizer.fit_transform(combi['description'])
bow.shape
```

Out[217]:
```
(89970, 200)
```

In [203]:
```python
bow = np.array(bow.todense())
```

In [204]:
```python
X_train, X_test , y_train, y_test = train_test_split(bow, y, test_size=0.2, random_state=1)
```

In [205]:
```python
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
```

Out[205]:
```
0.008737642000038414
```

In [206]:
```python
ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)
mean_squared_error(y_test, ridge.predict(X_test))
```

Out[206]:
```
5.256016628719822
```

In [207]:
```python
ridge.score(X_test, y_test)
```

Out[207]:
```
0.42608158195897544
```

Checking accuracy of linear model using review based features, Count vectoriser and L2 regularisation on public dataset

In [218]:

```
import re # for regular expressions
import pandas as pd
pd.set_option("display.max_colwidth", 200)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
import nltk # for text manipulation
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

%matplotlib inline


def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for i in r:
        input_txt = re.sub(i, '', input_txt)

    return input_txt

public['description'] = np.vectorize(remove_pattern)(public['description'], "@[\w]*")

public['description'] = public['description'].str.replace("[^a-zA-Z#]", " ")

public['description'] = public['description'].str.replace('#','')

public['description'] = public['description'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>2]))

tokenized_tweet = public['description'].apply(lambda x: x.split()) # tokenizing

tokenized_tweet.head()

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
tokenized_tweet.apply(lambda x: [lemmatizer.lemmatize(i) for i in x]) # stemming
#lemmatized_output = ' '.join([lemmatizer.lemmatize(token) for token in tokens])

for i in range(len(tokenized_tweet)):
    tokenized_tweet[i] = ' '.join(tokenized_tweet[i])

public['description'] = tokenized_tweet

all_words = ' '.join([text for text in public['description']])

bow_public = bow_vectorizer.transform(public['description'])
bow.shape

bow_public = np.array(bow_public.todense())

y = public['points']

mean_squared_error(y, ridge.predict(bow_public))

ridge.score(bow_public, y)
```

Out[218]: 0.41728530345424897

Checking accuracy of linear model using review based features, Count vectoriser and L2 regularisation on private dataset

In [219]:

```
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for i in r:
        input_txt = re.sub(i, '', input_txt)

    return input_txt

private['description'] = np.vectorize(remove_pattern)(private['description'], "@[\w]*")

private['description'] = private['description'].str.replace("[^a-zA-Z#]", " ")

private['description'] = private['description'].str.replace('#','')

private['description'] = private['description'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>2]))

tokenized_tweet = private['description'].apply(lambda x: x.split()) # tokenizing

tokenized_tweet.head()

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
tokenized_tweet.apply(lambda x: [lemmatizer.lemmatize(i) for i in x]) # stemming
#lemmatized_output = ' '.join([lemmatizer.lemmatize(token) for token in tokens])

for i in range(len(tokenized_tweet)):
    tokenized_tweet[i] = ' '.join(tokenized_tweet[i])

private['description'] = tokenized_tweet

all_words = ' '.join([text for text in private['description']])


bow_private = bow_vectorizer.transform(private['description'])


bow_private = np.array(bow_private.todense())

y = private['points']


mean_squared_error(y, ridge.predict(bow_private))

ridge.score(bow_private, y)
```

Out[219]: 0.42575316972328936

Combine non-review and review based count vectoriser features

In [54]:

```
X.shape, bow.shape, type(bow), type(X)
```

Out[54]: ((89970, 31), (89970, 200), numpy.ndarray, numpy.ndarray)

In [55]:

```
X_bow = np.hstack((X,bow))
```

In [56]:

```
X_train, X_test , y_train, y_test = train_test_split(X_bow, y, test_size=0.2, random_state=1)
```

In [57]:

```
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
```

Out[57]: 0.006609705742330144

In [58]:

```
ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)
mean_squared_error(y_test, ridge.predict(X_test))
```

Out[58]: 4.333038880809418

In [60]:

```
ridge.score(X_test, y_test)
```

Out[60]: 0.5268639740985578

Trying review based tf-idf vectorizer

In [220]:

```
tfidf_vectorizer = TfidfVectorizer(max_df=0.90, min_df=2, max_features=200, stop_words='english')
tfidf = tfidf_vectorizer.fit_transform(combi['description'])
tfidf.shape
```

Out[220]: (89970, 200)

In [223]:

```
X_train, X_test, y_train, y_test = train_test_split(tfidf, combi['points'], test_size=0.2, random_state=1)
```

In [224]:

```
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
```

Out[224]: 0.005

In [225]:

```
ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)
mean_squared_error(y_test, ridge.predict(X_test))
```

Out[225]: 5.138117016528831

In [226]:

```
ridge.score(X_test, y_test)
```

Out[226]: 0.43895535380866313

Checking accuracy of tfidf vectoriser based model on public dataset

In [227]:

```
tfidf_public = tfidf_vectorizer.transform(public['description'])

mean_squared_error(public['points'], ridge.predict(tfidf_public))

ridge.score(tfidf_public, public['points'])
```

Out[227]: 0.42952393763233454

Checking accuracy of tfidf vectoriser based model on private dataset

In [228]:

```
tfidf_private = tfidf_vectorizer.transform(private['description'])

mean_squared_error(private['points'], ridge.predict(tfidf_private))

ridge.score(tfidf_private, private['points'])
```

Out[228]: 0.43694473915367305

Combining non review and review based tfidf features

In [48]:

```
tfidf = np.array(tfidf.todense())
X.shape, tfidf.shape, type(tfidf), type(X)
```

Out[48]: ((89970, 31), (89970, 200), numpy.ndarray, numpy.ndarray)

In [49]:

```
X_tfidf = np.hstack((X,tfidf))

X_train, X_test , y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=1)

ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_

ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)
mean_squared_error(y_test, ridge.predict(X_test))

ridge.score(X_test, y_test)
```

Out[49]: 0.539594247383281

Task 3: Custom Word Vectors

Custom word to vec embeddings using skip-gram model

In [70]:

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import gensim
```

In [229]:

```
tokenized_tweet = combi['description'].apply(lambda x: x.split()) # tokenizing

model_w2v = gensim.models.Word2Vec(
            tokenized_tweet,
            size=50, # desired no. of features/independent variables
            window=5, # context window size
            min_count=2,
            sg = 1, # 1 for skip-gram model
            hs = 0,
            negative = 10, # for negative sampling
            workers= 2, # no.of cores
            seed = 34)
```

In [230]:

```
model_w2v.train(tokenized_tweet, total_examples= len(combi['description']), epochs=20)
```

Out[230]: (49396884, 61990780)

model_w2v.shape

In [231]:

```
def word_vector(tokens, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in tokens:
        try:
            vec += model_w2v[word].reshape((1, size))
            count += 1.
        except KeyError: # handling the case where the token is not in vocabulary

            continue
    if count != 0:
        vec /= count
    return vec
```

In [232]:

```
wordvec_arrays = np.zeros((len(tokenized_tweet), 50))

for i in range(len(tokenized_tweet)):
    wordvec_arrays[i,:] = word_vector(tokenized_tweet[i], 50)

wordvec_df = pd.DataFrame(wordvec_arrays)
wordvec_df.shape
```

Out[232]: (89970, 50)

In [236]:

```
X_train, X_test , y_train, y_test = train_test_split(wordvec_df, combi['points'], test_size=0.2, random_state=1)
```

In [237]:

```
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
```

Out[237]: 0.005

In [238]:

```
ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)
mean_squared_error(y_test, ridge.predict(X_test))
```

Out[238]: 4.208923794471053

In [239]:

```
ridge.score(X_test, y_test)
```

Out[239]: 0.5404164300815004

Checking accuracy of custom word to vec embeddings using skip-gram model on private dataset

In [241]:

```
tokenized_tweet_private = private['description'].apply(lambda x: x.split()) # tokenizing

wordvec_arrays_private = np.zeros((len(tokenized_tweet_private), 50))

for i in range(len(tokenized_tweet_private)):
    wordvec_arrays_private[i,:] = word_vector(tokenized_tweet_private[i], 50)

wordvec_df_private = pd.DataFrame(wordvec_arrays_private)
wordvec_df_private.shape

mean_squared_error(private['points'], ridge.predict(wordvec_df_private))

ridge.score(wordvec_df_private, private['points'])
```

Out[241]: 0.5018946673156571

Checking accuracy of custom word to vec embeddings using skip-gram model on public dataset

In [242]:

```
tokenized_tweet_public = public['description'].apply(lambda x: x.split()) # tokenizing

wordvec_arrays_public = np.zeros((len(tokenized_tweet_public), 50))

for i in range(len(tokenized_tweet_public)):
    wordvec_arrays_public[i,:] = word_vector(tokenized_tweet_public[i], 50)

wordvec_df_public = pd.DataFrame(wordvec_arrays_public)
wordvec_df_public.shape

mean_squared_error(public['points'], ridge.predict(wordvec_df_public))

ridge.score(wordvec_df_public, public['points'])
```

Out[242]: 0.4933264314798058

Combine custom skip gram word2vec and bow features

In [82]:

```
X_bow_sg_w2v = np.hstack((wordvec_df,bow))
```

In [83]:

```
X_bow_sg_w2v
```

Out[83]:

```
array([[-0.00262551, -0.23499813,  0.2317798 , ...,  0.        ,
         0.        ,  0.        ],
       [-0.27572221, -0.10347272,  0.18937457, ...,  0.        ,
         0.        ,  0.        ],
       [-0.03521588, -0.15286143,  0.28447456, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [-0.17225228, -0.19234901,  0.18830725, ...,  0.        ,
         0.        ,  0.        ],
       [-0.16508608, -0.23316762,  0.32500752, ...,  0.        ,
         0.        ,  0.        ],
       [-0.32364449, -0.09797542,  0.10868892, ...,  0.        ,
         0.        ,  0.        ]])
```

In [84]:

```
X_train, X_test , y_train, y_test = train_test_split(X_bow_sg_w2v, y, test_size=0.2, random_state=1)
```

In [85]:

```
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
```

Out[85]: 0.005

In [86]:

```
ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)
mean_squared_error(y_test, ridge.predict(X_test))
```

Out[86]: 3.545981161827917

In [87]:

```
ridge.score(X_test, y_test)
```

Out[87]: 0.6128048972144841

Custom word to vec embeddings using cbow model

In [243]:

```
tokenized_tweet = combi['description'].apply(lambda x: x.split()) # tokenizing

model_w2v = gensim.models.Word2Vec(
            tokenized_tweet,
            size=50, # desired no. of features/independent variables
            window=5, # context window size
            min_count=2,
            sg = 1, # 1 for skip-gram model
            hs = 0,
            negative = 10, # for negative sampling
            workers= 2, # no.of cores
            seed = 34)
```

In [244]:

```
model_w2v.train(tokenized_tweet, total_examples= len(combi['description']), epochs=20)
```

Out[244]: (49396884, 61990780)

In [245]:

```
wordvec_arrays = np.zeros((len(tokenized_tweet), 50))

for i in range(len(tokenized_tweet)):
    wordvec_arrays[i,:] = word_vector(tokenized_tweet[i], 50)

wordvec_df = pd.DataFrame(wordvec_arrays)
wordvec_df.shape
```

Out[245]: (89970, 50)

In [247]:

```
X_train, X_test , y_train, y_test = train_test_split(wordvec_df, combi['points'], test_size=0.2, random_state=1)
```

In [248]:

```
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
```

Out[248]: 0.005

In [249]:

```
ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)
mean_squared_error(y_test, ridge.predict(X_test))
```

Out[249]: 4.650935144232271

In [250]:

```
ridge.score(X_test, y_test)
```

Out[250]: 0.49215203561954066

Custom word to vec embeddings using cbow model on private dataset

In [251]:

```
tokenized_tweet_private = private['description'].apply(lambda x: x.split()) # tokenizing

wordvec_arrays_private = np.zeros((len(tokenized_tweet_private), 50))

for i in range(len(tokenized_tweet_private)):
    wordvec_arrays_private[i,:] = word_vector(tokenized_tweet_private[i], 50)

wordvec_df_private = pd.DataFrame(wordvec_arrays_private)
wordvec_df_private.shape

mean_squared_error(private['points'], ridge.predict(wordvec_df_private))

ridge.score(wordvec_df_private, private['points'])
```

Out[251]: 0.4720751102727523

Custom word to vec embeddings using cbow model on public dataset

In [252]:

```
tokenized_tweet_public = public['description'].apply(lambda x: x.split()) # tokenizing

wordvec_arrays_public = np.zeros((len(tokenized_tweet_public), 50))

for i in range(len(tokenized_tweet_public)):
    wordvec_arrays_public[i,:] = word_vector(tokenized_tweet_public[i], 50)

wordvec_df_public = pd.DataFrame(wordvec_arrays_public)
wordvec_df_public.shape

mean_squared_error(public['points'], ridge.predict(wordvec_df_public))

ridge.score(wordvec_df_public, public['points'])
```

Out[252]: 0.4628463725214597

Task 2 Pre-trained Word Vectors

Using pre-trained word2vec embeddings created by Google

In [253]:

```
model = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
```

In [254]:

```
def word_vector_pretrained(tokens, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in tokens:
        try:
            vec += model[word].reshape((1, size))
            count += 1.
        except KeyError: # handling the case where the token is not in vocabulary

            continue
    if count != 0:
        vec /= count
    return vec
```

In [255]:

```
wordvec_arrays = np.zeros((len(tokenized_tweet), 300))

for i in range(len(tokenized_tweet)):
    wordvec_arrays[i,:] = word_vector_pretrained(tokenized_tweet[i], 300)

wordvec_df = pd.DataFrame(wordvec_arrays)
wordvec_df.shape
```

Out[255]: (89970, 300)

In [257]:

```
X_train, X_test , y_train, y_test = train_test_split(wordvec_df, combi['points'], test_size=0.2, random_state=1)
```

In [258]:

```
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
```

Out[258]: 0.008737642000038414

In [259]:

```
ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)
mean_squared_error(y_test, ridge.predict(X_test))
```

Out[259]: 5.239015673425837

In [261]:

```
ridge.score(X_test, y_test)
```

Out[261]: 0.4279379614297319

Using pre-trained word2vec embeddings on private dataset

In [262]:

```
tokenized_tweet_private = private['description'].apply(lambda x: x.split()) # tokenizing

wordvec_arrays_private = np.zeros((len(tokenized_tweet_private), 300))

for i in range(len(tokenized_tweet_private)):
    wordvec_arrays_private[i,:] = word_vector_pretrained(tokenized_tweet_private[i], 300)

wordvec_df_private = pd.DataFrame(wordvec_arrays_private)
wordvec_df_private.shape

mean_squared_error(private['points'], ridge.predict(wordvec_df_private))

ridge.score(wordvec_df_private, private['points'])
```

Out[262]: 0.4846125449904594

Using pre-trained word2vec embeddings on public dataset

In [264]:

```
tokenized_tweet_public = public['description'].apply(lambda x: x.split()) # tokenizing

wordvec_arrays_public = np.zeros((len(tokenized_tweet_public), 300))

for i in range(len(tokenized_tweet_public)):
    wordvec_arrays_public[i,:] = word_vector_pretrained(tokenized_tweet_public[i], 300)

wordvec_df_public = pd.DataFrame(wordvec_arrays_public)
wordvec_df_public.shape

mean_squared_error(public['points'], ridge.predict(wordvec_df_public))

ridge.score(wordvec_df_public, public['points'])
```

Out[264]: 0.4765056668795022

Combine pretrained word embedding with bag of words features

In [103]:

```
wordvec_bow = np.hstack((wordvec_df,bow))
```

In [107]:

```
X_train, X_test , y_train, y_test = train_test_split(wordvec_bow, y, test_size=0.2, random_state=1)
```

In [108]:

```
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
```

Out[108]: 0.01155064850041579

In [109]:

```
ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)
mean_squared_error(y_test, ridge.predict(X_test))
ridge.score(X_test, y_test)
```

Out[109]: 0.5598097292977211

In [ ]: