Homework 1: CSE 258

By Amogha Sekhar, A53301791

Basic Preprocessing

In [1]:
```
path= "amazon_reviews_us_Gift_Card_v1_00.tsv.gz"
```

In [2]:
```
import gzip
f= gzip.open(path, 'rt', encoding= "utf8")
```

In [3]:
```
header= f.readline()
header
```

Out[3]:
```
'marketplace\tcustomer_id\treview_id\tproduct_id\tproduct_parent\tproduct
_title\tproduct_category\tstar_rating\thelpful_votes\ttotal_votes\tvine\t
verified_purchase\treview_headline\treview_body\treview_date\n'
```

In [4]:
```
header= header.strip().split('\t')
```

In [5]:
```
header
```

Out[5]:
```
['marketplace',
 'customer_id',
 'review_id',
 'product_id',
 'product_parent',
 'product_title',
 'product_category',
 'star_rating',
 'helpful_votes',
 'total_votes',
 'vine',
 'verified_purchase',
 'review_headline',
 'review_body',
 'review_date']
```

In [6]:
```
lines= []
for line in f:
    fields= line.split('\t')
    lines.append(fields)
```

In [7]:
```
lines[0]
```

Out[7]:
```
['US',
 '24371595',
 'R27ZP1F1CD0C3Y',
 'B004LLIL5A',
 '346014806',
 'Amazon eGift Card - Celebrate',
 'Gift Card',
 '5',
 '0',
 '0',
 'N',
 'Y',
 'Five Stars',
 'Great birthday gift for a young adult.',
 '2015-08-31\n']
```

In [8]:
```
z= zip(header, lines[0])
list(z)
```

Out[8]:
```
[('marketplace', 'US'),
 ('customer_id', '24371595'),
 ('review_id', 'R27ZP1F1CD0C3Y'),
 ('product_id', 'B004LLIL5A'),
 ('product_parent', '346014806'),
 ('product_title', 'Amazon eGift Card - Celebrate'),
 ('product_category', 'Gift Card'),
 ('star_rating', '5'),
 ('helpful_votes', '0'),
 ('total_votes', '0'),
 ('vine', 'N'),
 ('verified_purchase', 'Y'),
 ('review_headline', 'Five Stars'),
 ('review_body', 'Great birthday gift for a young adult.'),
 ('review_date', '2015-08-31\n')]
```

In [9]:
```
d= dict(zip(header, lines[0]))
d
```

Out[9]:
```
{'customer_id': '24371595',
 'helpful_votes': '0',
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'B004LLIL5A',
 'product_parent': '346014806',
 'product_title': 'Amazon eGift Card - Celebrate',
 'review_body': 'Great birthday gift for a young adult.',
 'review_date': '2015-08-31\n',
 'review_headline': 'Five Stars',
 'review_id': 'R27ZP1F1CD0C3Y',
 'star_rating': '5',
 'total_votes': '0',
 'verified_purchase': 'Y',
 'vine': 'N'}
```

In [10]:
```
d['star_rating']= int(d['star_rating'])
d['helpful_votes']= int(d['helpful_votes'])
d['total_votes']= int(d['total_votes'])
d['customer_id']= int(d['customer_id'])
d['product_parent']= int(d['product_parent'])
d['review_date']= d['review_date'].strip()
```

In [11]:
```
dataset= []
for line in lines:
    d = dict(zip(header, line))
    d['star_rating'] = int(d['star_rating'])
    d['helpful_votes'] = int(d['helpful_votes'])
    d['total_votes'] = int(d['total_votes'])
    dataset.append(d)
```

In [12]:
```
nRatings= len(dataset)
nRatings
```

Out[12]: 149086

In [13]:
```
dataset
```

Out[13]:
```
[{'customer_id': '24371595',
  'helpful_votes': 0,
  'marketplace': 'US',
  'product_category': 'Gift Card',
  'product_id': 'B004LLIL5A',
  'product_parent': '346014806',
  'product_title': 'Amazon eGift Card - Celebrate',
  'review_body': 'Great birthday gift for a young adult.',
  'review_date': '2015-08-31\n',
  'review_headline': 'Five Stars',
  'review_id': 'R27ZP1F1CD0C3Y',
  'star_rating': 5,
  'total_votes': 0,
  'verified_purchase': 'Y',
  'vine': 'N'},
 {'customer_id': '42489718',
  'helpful_votes': 0,
  'marketplace': 'US',
  'product_category': 'Gift Card',
```

In [15]:
```
import pandas as pd
df= pd.DataFrame(dataset)
```

In [16]:
```
df.head()
```

Out[16]:

	customer_id	helpful_votes	marketplace	product_category	product_id	product_parer
0	24371595	0	US	Gift Card	B004LLIL5A	346014806
1	42489718	0	US	Gift Card	B004LLIKVU	473048287
2	861463	0	US	Gift Card	B00IX1I3G6	926539283
3	25283295	0	US	Gift Card	B00IX1I3G6	926539283
4	397970	0	US	Gift Card	B005ESMGV4	379368939

In [17]:
```
print(type(df))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [18]:
```
df.describe()
```

Out[18]:

	helpful_votes	star_rating	total_votes
count	149086.000000	149086.000000	149086.000000
mean	0.396603	4.731363	0.489449
std	20.649231	0.829306	22.766277
min	0.000000	1.000000	0.000000
25%	0.000000	5.000000	0.000000
50%	0.000000	5.000000	0.000000
75%	0.000000	5.000000	0.000000
max	5987.000000	5.000000	6323.000000

In [19]:
```
df.shape
```

Out[19]: (149086, 15)

In [20]:
```
df= df.dropna()
```

In [21]:
```
df.shape
```

Out[21]: (149086, 15)

Question 1: What is the distribution of ratings in the dataset? That is, how many 1-star, 2-star, 3-star (etc.) reviews are there? You may write out the values or include a simple plot (1 mark).

In [22]:
```
df.groupby(['star_rating']).count()
```

Out[22]:

	customer_id	helpful_votes	marketplace	product_category	product_id	produc
star_rating						
1	4793	4793	4793	4793	4793	4793
2	1569	1569	1569	1569	1569	1569
3	3156	3156	3156	3156	3156	3156
4	9859	9859	9859	9859	9859	9859
5	129709	129709	129709	129709	129709	129709

Answer for Question 1

We have 4793 users who gave a rating of 1.

We have 1569 users who gave a rating of 2.

We have 3156 users who gave a rating of 3.

We have 9859 users who gave a rating of 4.

We have 129709 users who gave a rating of 5.

In [ ]:
```
%matplotlib inline

df1= df[['star_rating']]
df1.plot.bar()
```

Question 3: Train a simple predictor to predict the star rating using two features: star rating ' θ0 + θ1 × [review is verified] + θ2 × [review length] Report the values of θ0, θ1, and θ2. Briefly describe your interpretation of these values, i.e., what do θ0, θ1, and θ2 represent? Explain these in terms of the features and labels, e.g. if the coefficient of 'review length' is negative, what would that say about positive versus negative reviews (1 mark)?

In [23]:
```
import numpy
import scipy.optimize
from sklearn.linear_model import LinearRegression
from sklearn import metrics

df['review_length']= df['review_body'].str.len()
df['verified_purchase']= df['verified_purchase'].replace(to_replace= "Y", va
df['verified_purchase']= df['verified_purchase'].replace(to_replace= "N", va
df['ones']= 1
df.head()

X= df[['ones', 'verified_purchase', 'review_length']].values
y= df['star_rating'].values

theta,residuals,rank,s = numpy.linalg.lstsq(X, y)
theta
```

Out[23]:
```
array([  4.84461817e+00,   5.04148265e-02,  -1.24659895e-03])
```

Answer for Question 3

star_rating= 4.8446 + 0.0504* (verified purchase) - 0.00124(review length)

Theta_0 is the global bias, and can be interpreted as the approximate average ratings of all the users(4.7313). Theta_1 is a positive value, implying that if it is a verified purchase, the rating goes up because it holds more value when a customer is confirmed to have actually purchased the product on Amazon. If more such users continue to buy the product and give it a good rating, the product is

good. Theta_2 is negative for review length, which implies that if a review is very long, it is usually negative and hence the rating goes down. A positive review is usually considered to have a shorter review length, and as such the star_rating value will not get penalized much for a positive review.

Question 4: Train another predictor that only uses one feature: star rating ' θ0 + θ1 × [review is verified] Report the values of θ0 and θ1. Note that coefficient you found here might be quite different (i.e., much larger or smaller) than the one from Question 3, even though these coefficients refer to the same feature. Provide an explanation as to why these coefficients might vary so significantly (1 mark).

In [24]:
```
X= df[['ones', 'verified_purchase']].values
y= df['star_rating'].values

theta,residuals,rank,s = numpy.linalg.lstsq(X, y)
theta

df['verified_purchase'].corr(df['review_length'])
```

Out[24]:
```
-0.17649153877022028
```

Answer for Question 4

star_rating= 4.5775+ 0.1685* (verified purchase)

The values of theta0 and theta1 are 4.5775 and 0.1685 respectively.

The coefficients for verified purchase might vary significantly because in the first predictor, we also had review length, as such review length and review verified contributed to the star rating(both having opposite effects, that is negative and positive coefficients respectively). However, in this predictor, we only have one dependent variable, that is verified purchase, so more importance is on this variable. So if a product has a verified purchase, we know that it has been bought by a customer from Amazon and that customer's rating has more importance. Moreover, in the previous question, verified_purchase and review_length are negatively correlated (-0.1764) as shown above. As a result, in Question 4, having just verified_purchase, theta_1 is larger as we can confidently give importance to verified_purchase, as there is no other variable that is doing the opposite effect. This could probably account for the coefficients varying significantly.

Question 5: Split the data into two fractions – the first 90% for training, and the remaining 10% testing (based on the order they appear in the file). Train the same model as in Question 4 on the training set only. What is the model's MSE on the training and on the test set (1 mark)?

In [25]:
```
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

X= df[['verified_purchase']].values
y= df['star_rating'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, ran

reg = LinearRegression()
reg.fit(X_train, y_train)
```

Out[25]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fal
se)
```

In [26]:
```
y_pred_train = reg.predict(X_train)
y_pred_test= reg.predict(X_test)
```

In [27]:
```
print('Mean Squared Error on Training Set:', metrics.mean_squared_error(y_tr
print('Mean Squared Error on Test Set: ', metrics.mean_squared_error(y_test,
```

```
Mean Squared Error on Training Set: 0.683407668054
Mean Squared Error on Test Set:  0.704104215554
```

Answer for Question 5

Mean Squared Error on the Training Set: 0.683407668054

Mean Squared Error on the Test Set: 0.704104215554

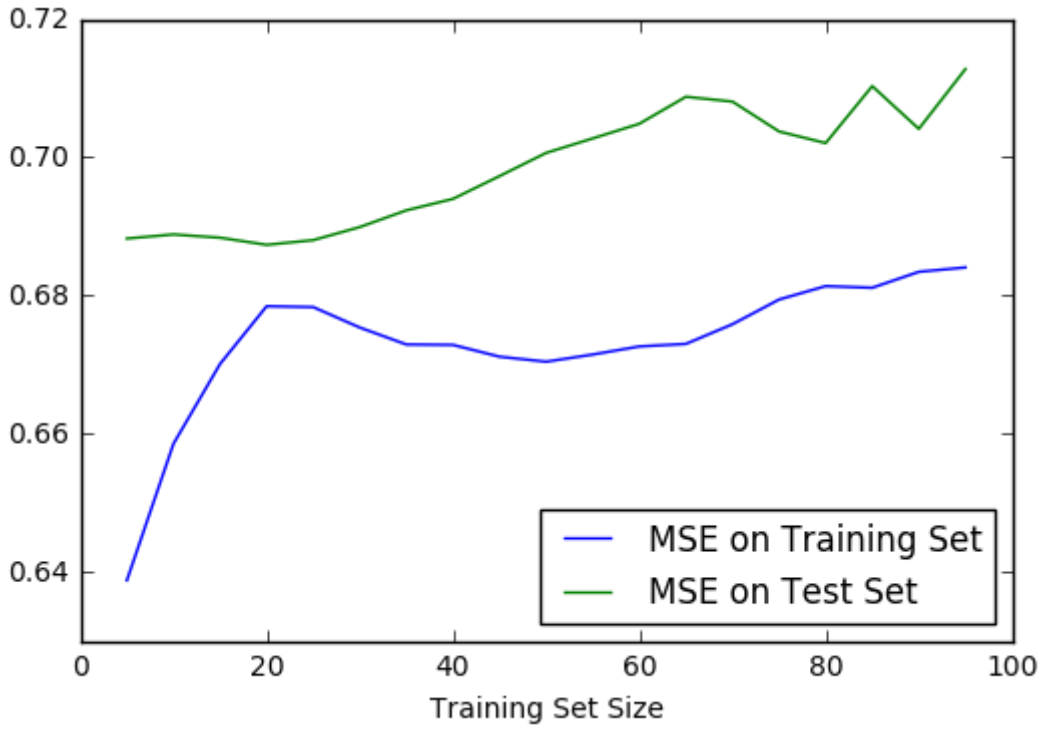
Question 7: (CSE258 only) Repeat the above experiment, varying the size of the training and test fractions between 5% and 95% for training (using the complement for testing). Show how the training and test error vary as a function of the training set size (again using a simple plot or table). Does the size of the training set make a significant difference in testing performance? Comment on why it might or might not make a significant difference in this instance (2 marks).

In [28]:
```
MSE_train= []
MSE_test= []
training_percent= []
test_percent= []

i= 0.05
while i<1:

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= i,
    reg = LinearRegression()
    reg.fit(X_train, y_train)
    y_pred_train = reg.predict(X_train)
    y_pred_test= reg.predict(X_test)
    #print('Mean Squared Error on Training Set:', metrics.mean_squared_erro
    #print('Mean Squared Error on Test Set: ', metrics.mean_squared_error(y_
    MSE_train_val = metrics.mean_squared_error(y_train, y_pred_train)
    MSE_test_val = metrics.mean_squared_error(y_test, y_pred_test)
    MSE_train.append(MSE_train_val)
    MSE_test.append(MSE_test_val)
    training_percent_val= (1-i)*100
    test_percent_val= i*100
    training_percent.append(training_percent_val)
    test_percent.append(test_percent_val)
    i= i+0.05

print(MSE_train)
print(MSE_test)
print(test_percent)
print(training_percent)
```

```
[0.6840410595806784, 0.68340766805436437, 0.68109946807610944, 0.68133654
799911758, 0.67939700850701168, 0.67582232862473413, 0.67297249525687164,
0.67260569080113519, 0.67143489411973645, 0.67039642364721319, 0.67111951
387733604, 0.67282880375081544, 0.6728876411638105, 0.67534161012450677,
0.67830127780409455, 0.67842652960627481, 0.6700699057009053, 0.658515014
55169115, 0.63873574270063826]
[0.71276330995736226, 0.70410421555433833, 0.71029378082090078, 0.7020467
8135352239, 0.70372894445894874, 0.70805169313998273, 0.7087565352231938
2, 0.70484286945888674, 0.70272501010050314, 0.70062689593553684, 0.69727
020889810165, 0.69396684140256293, 0.69230995111674143, 0.689887577548106
92, 0.68799330232600597, 0.68731529301587058, 0.68834757602297258, 0.6888
1962634657212, 0.68822614920705671]
[5.0, 10.0, 15.000000000000002, 20.0, 25.0, 30.0, 35.0, 40.0, 44.99999999
999999, 49.99999999999999, 54.99999999999999, 60.0, 65.0, 70.0, 75.000000
00000001, 80.00000000000001, 85.00000000000001, 90.00000000000003, 95.000
00000000003]
[95.0, 90.0, 85.0, 80.0, 75.0, 70.0, 65.0, 60.00000000000001, 55.00000000
000001, 50.0, 45.00000000000001, 40.0, 35.0, 29.999999999999993, 24.99999
999999999, 19.999999999999986, 14.99999999999998, 9.999999999999975, 4.99
9999999999972]
```

In [29]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

plt.plot(training_percent, MSE_train)
plt.plot(training_percent, MSE_test)
plt.legend(['MSE on Training Set', 'MSE on Test Set'], loc=0)
plt.xlabel('Training Set Size')
```

Out[29]: <matplotlib.text.Text at 0x12cec34e0>



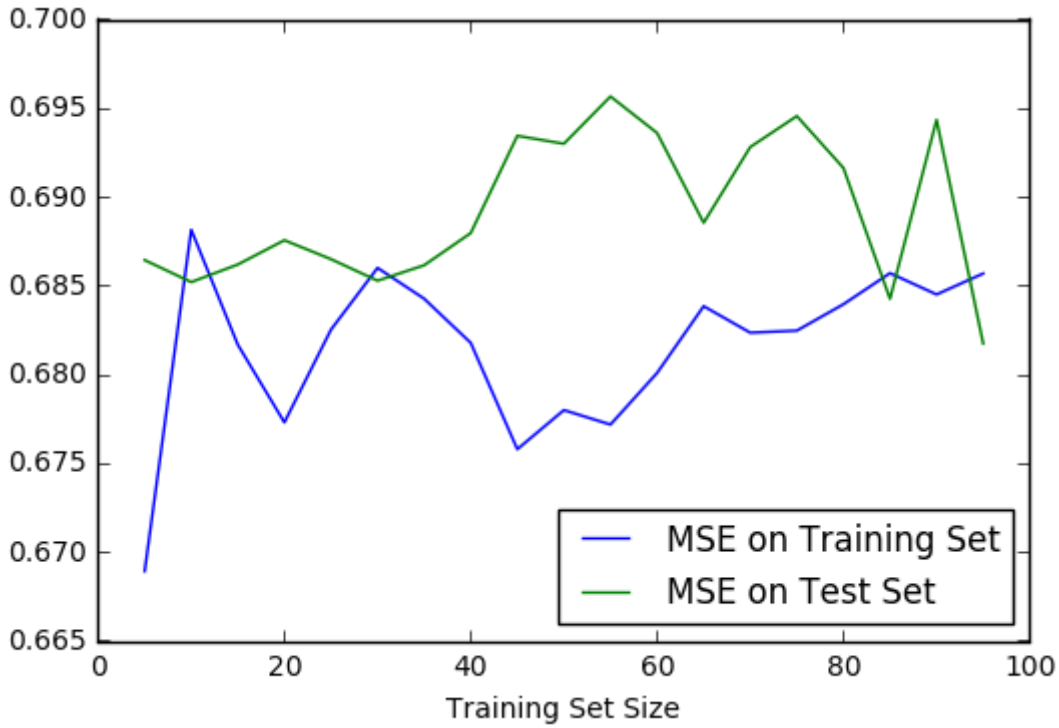
In [30]:
```
#With shuffling

Xy = list(zip(X,y))
np.random.shuffle(Xy)
X = [d[0] for d in Xy]
y = [d[1] for d in Xy]

MSE_train= []
MSE_test= []
training_percent= []
test_percent= []

i= 0.05
while i<1:

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= i,
    reg = LinearRegression()
    reg.fit(X_train, y_train)
    y_pred_train = reg.predict(X_train)
    y_pred_test= reg.predict(X_test)
    #print('Mean Squared Error on Training Set:', metrics.mean_squared_erro
    #print('Mean Squared Error on Test Set: ', metrics.mean_squared_error(y_
    MSE_train_val = metrics.mean_squared_error(y_train, y_pred_train)
    MSE_test_val = metrics.mean_squared_error(y_test, y_pred_test)
    MSE_train.append(MSE_train_val)
    MSE_test.append(MSE_test_val)
    training_percent_val= (1-i)*100
    test_percent_val= i*100
    training_percent.append(training_percent_val)
    test_percent.append(test_percent_val)
    i= i+0.05

print(MSE_train)
print(MSE_test)
print(test_percent)
print(training_percent)
```

```
[0.68567422348935791, 0.68449410196227101, 0.68569533939635341, 0.6839427
5072768973, 0.6824588770210448, 0.68234310479162197, 0.68383392373993312,
0.68007531741535554, 0.67717093389435001, 0.67798536383994357, 0.67578238
069479779, 0.68176964627017167, 0.68425586289582718, 0.68599102707075488,
0.68251403755198237, 0.67729221960342589, 0.6816646304128966, 0.688138204
28395944, 0.66891467093513102]
[0.68173487316942605, 0.69432978363258202, 0.684248452095784, 0.691619154
43733568, 0.69453939842358847, 0.69279417335190396, 0.68853130447420474,
0.69358673783219749, 0.69564464301668782, 0.69298074267739518, 0.69342820
427629903, 0.68795062683387032, 0.68613880557367124, 0.68526915830327173,
0.68648209133989424, 0.68754803749328264, 0.68618156182254431, 0.68519127
632322374, 0.68642957861617859]
[5.0, 10.0, 15.000000000000002, 20.0, 25.0, 30.0, 35.0, 40.0, 44.99999999
999999, 49.99999999999999, 54.99999999999999, 60.0, 65.0, 70.0, 75.000000
00000001, 80.00000000000001, 85.00000000000001, 90.00000000000003, 95.000
00000000003]
[95.0, 90.0, 85.0, 80.0, 75.0, 70.0, 65.0, 60.00000000000001, 55.00000000
000001, 50.0, 45.00000000000001, 40.0, 35.0, 29.999999999999993, 24.99999
```

```
999999999, 19.999999999999986, 14.99999999999998, 9.999999999999975, 4.99
9999999999972]
```

In [31]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

plt.plot(training_percent, MSE_train)
plt.plot(training_percent, MSE_test)
plt.legend(['MSE on Training Set', 'MSE on Test Set'], loc=0)
plt.xlabel('Training Set Size')
```

Out[31]: <matplotlib.text.Text at 0x13260bd68>



Answer for Question 7

As can be seen in the first graph, as the size of the training set increases, the mean squared error on the test and training set increases. This can be because the dataset is not evenly distributed as shown by the second graph when the data is shuffled.

When the data is not shuffled, as the size of the training set decreases, the MSE on the test set also decreases. At 95%-5% (i.e 95% of the data is used for training, 5% for testing), MSE on the training set is 0.6840410595806784 and the MSE on the test set is 0.71276330995736226. At 5%-95%, MSE on the training set is 0.6387357427006382 and the MSE on the test set is 0.68822614920705671. So MSE on the training set varies from 0.68 to 0.63 when it moves from 95% to 5%, and MSE on the test set varies from 0.71 to 0.68 as it moves from 5% to 95%. Hence, the size of the training set makes a significant difference in testing performance. This is probably again because the data is not evenly distributed.

Question 8: First, let's train a predictor that estimates whether a review is

Question 8: First, let's train a predictor that estimates whether a review is verified using the rating and the length: p(review is verified) = σ(θ0 + θ1 × [star rating] + θ2 × [review length]) Train a logistic regressor to make the above prediction (you may use a logistic regression library with default parameters, e.g. linear model.LogisticRegression() from sklearn). Report the classification accuracy of this predictor. Report also the proportion of labels that are positive (i.e., the proportion of reviews that are verified) and the proportion of predictions that are positive (1 mark).

In [32]:
```
X= df[['star_rating','review_length']].values
y= df['verified_purchase'].values


X_train = X[:134178]
y_train = y[:134178]

X_test = X[134178:]
y_test = y[134178:]

from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression()
logisticRegr.fit(X_train, y_train)
```

Out[32]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False)
```

In [33]:
```
y_train_predictions = logisticRegr.predict(X_train)
y_test_predictions = logisticRegr.predict(X_test)
```

In [34]:
```
#accuracy
1-sum(abs(y_test_predictions - y_test)) / len(y_test)
```

Out[34]:
```
0.55896163133887844
```

In [35]:
```
sum( y== 1)/len(y) #Proportion of labels that are positive
```

Out[35]:
```
0.91250687522637941
```

In [36]:
```
(sum((y_test_predictions == 1))+ sum( (y_train_predictions == 1)))/len(y) #
```

Out[36]:
```
0.99959754772413234
```

Answer for Question 8

The classification accuracy for this predictor is 55.8961%. The proportion of labels that was marked positive (the proportions of reviews that are verified) is 0.9125. The proportion of predictions that is positive is 0.9995.

Question 9: Considering same prediction problem as above, can you come up with a more accurate predictor (e.g. using features from the text, timestamp, etc.)? Write down the feature vector you design, and report its train/test accuracy (1 mark)

In [37]:
```
X= df[['star_rating', 'review_length', 'helpful_votes', 'total_votes']].valu
y= df['verified_purchase'].values

X_train = X[:134178]
y_train = y[:134178]

X_test = X[134178:]
y_test = y[134178:]

from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression()
logisticRegr.fit(X_train, y_train)

y_train_predictions = logisticRegr.predict(X_train)
y_test_predictions = logisticRegr.predict(X_test)

print('Training Accuracy: ',(1-sum(abs(y_train_predictions- y_train))/len(y_
print('Test Accuracy: ',(1-sum(abs(y_test_predictions - y_test)) / len(y_tes
```

```
Training Accuracy:  95.1653773346
Test Accuracy:  55.9229943654
```

In [38]:
```
#By shuffling the dataset

Xy = list(zip(X,y))
np.random.shuffle(Xy)
X = [d[0] for d in Xy]
y = [d[1] for d in Xy]

X_train = X[:134178]
y_train = y[:134178]

X_test = X[134178:]
y_test = y[134178:]

from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression()
logisticRegr.fit(X_train, y_train)

y_train_predictions = logisticRegr.predict(X_train)
y_test_predictions = logisticRegr.predict(X_test)

print('Training Accuracy: ',(1-sum(abs(y_train_predictions- y_train))/len(y_
print('Test Accuracy: ',(1-sum(abs(y_test_predictions - y_test)) / len(y_tes
```

```
Training Accuracy:  91.0775238862
Test Accuracy:  91.286557553
```

In [39]:
```
#Balanced model

X_train = X[:134178]
y_train = y[:134178]

X_test = X[134178:]
y_test = y[134178:]

from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression(C=1.0, class_weight='balanced')
logisticRegr.fit(X_train, y_train)

y_train_predictions = logisticRegr.predict(X_train)
y_test_predictions = logisticRegr.predict(X_test)

print('Training Accuracy: ',(1-sum(abs(y_train_predictions- y_train))/len(y_
print('Test Accuracy: ',(1-sum(abs(y_test_predictions - y_test)) / len(y_tes
```

```
Training Accuracy:  74.212613096
Test Accuracy:  74.6176549504
```

In [40]:
```
#By shuffling the dataset and balancing the model

Xy = list(zip(X,y))
np.random.shuffle(Xy)
X = [d[0] for d in Xy]
y = [d[1] for d in Xy]

X_train = X[:134178]
y_train = y[:134178]

X_test = X[134178:]
y_test = y[134178:]

from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression(C=1.0, class_weight='balanced')
logisticRegr.fit(X_train, y_train)

y_train_predictions = logisticRegr.predict(X_train)
y_test_predictions = logisticRegr.predict(X_test)

print('Training Accuracy: ',(1-sum(abs(y_train_predictions- y_train))/len(y_
print('Test Accuracy: ',(1-sum(abs(y_test_predictions - y_test)) / len(y_tes
```

```
Training Accuracy:  74.3139709938
Test Accuracy:  74.3761738664
```

Answer for Question 9

The feature vector chosen by me is as follows: X= ['star_rating', 'review_length', 'helpful_votes', 'total_votes']

a) When the model was trained with 95% of the data used as training set, and 5% as test set, accuracy was as follows:

Training Accuracy: 95.1653773346

Test Accuracy: 55.9229943654

This shows that as features are added, the accuracy does not change by much. In the previous question, we got a test accuracy of 55.8961% and in this question, with the addition of two features, 'helpful_votes' and 'total_votes', the test accuracy is 55.9229%, which isn't significantly different.

b) When the data was shuffled and model trained as 95%-5% training and test set size, accuracy was as follows:

Training Accuracy: 91.0775238862

Test Accuracy: 91.286557553

c) When the model was balanced and trained on 95%-5% training and test set size, the accuracy was as follows:

Training Accuracy: 74.212613096

Test Accuracy: 74.6176549504

d) When the model was balanced, shuffled and trained on 95% of the data, the accuracy was as follows:

Training Accuracy: 74.3139709938

Test Accuracy: 74.3761738664

As such when the data is shuffled, it shows the best results at 95%-5% distribution with a training accuracy of 91.0775238862 and a test accuracy of 91.286557553. This is the most accurate predictor.