CSE258-HW1

October 13, 2019

0.1 Import required libraries

[27]:
```
import numpy as np
import gzip
import csv
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
import time
```

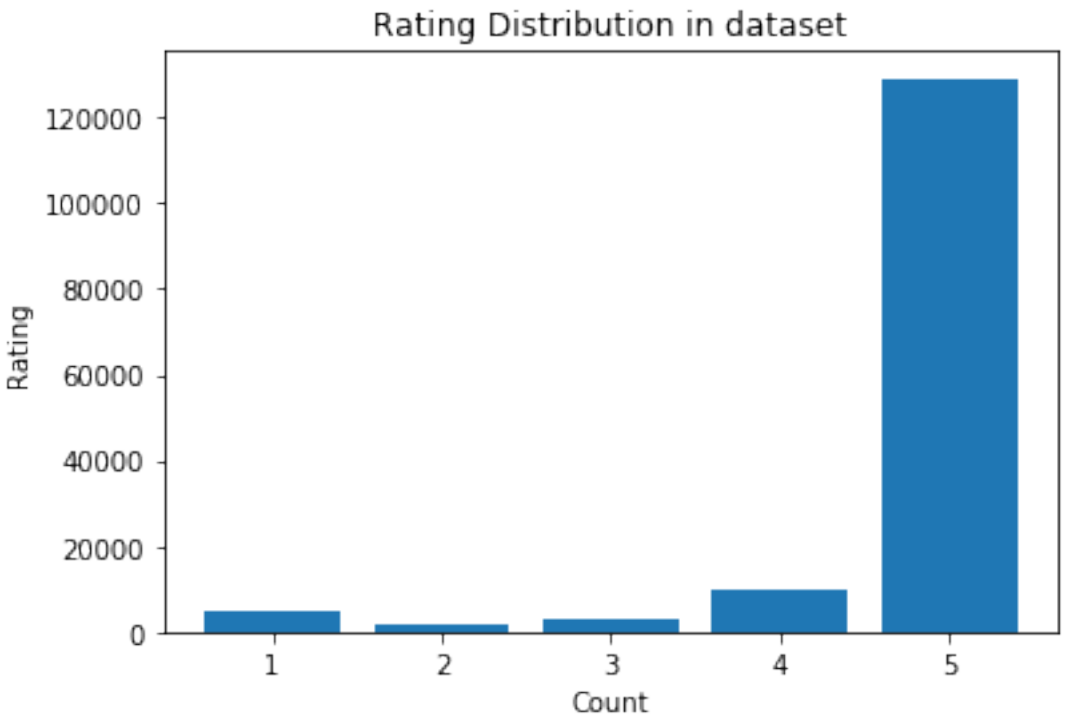
0.2 Plotting the ratings distribution graph

[28]:
```
c = csv.reader(gzip.open("amazon_reviews_us_Gift_Card_v1_00.tsv.gz", 'rt'),␣
 ↪delimiter = '\t')
dataset = []
first = True
for line in c:
    # The first line is the header
    if first:
        header = line
        first = False
    else:
        d = dict(zip(header, line))
        # Convert strings to integers for some fields:
        d['star_rating'] = int(d['star_rating'])
        d['helpful_votes'] = int(d['helpful_votes'])
        d['total_votes'] = int(d['total_votes'])
        dataset.append(d)

c1 = 0
c2 = 0
c3 =0
c4 = 0
c5= 0
ratings = [1,2,3,4,5]
for d in dataset:
```

```
    if d['star_rating']  == 1:
        c1 += 1
    if d['star_rating']  == 2:
        c2 += 1
    if d['star_rating']  == 3:
        c3 += 1
    if d['star_rating']  == 4:
        c4 += 1
    if d['star_rating']  == 5:
        c5 += 1

counts = [c1,c2,c3,c4,c5]
counts

ratingsdist = dict(zip(ratings,counts))

X = list(ratingsdist.keys())
Y = [ratingsdist[x] for x in X]

print("Rating Distribution in dataset for " + str(ratings) +" is as follows:" +␣
 ↪str(counts))
plt.xlabel("Count")
plt.ylabel("Rating")
plt.title("Rating Distribution in dataset")
plt.bar(X, Y)
```

```
Rating Distribution in dataset for [1, 2, 3, 4, 5] is as follows:[4766, 1560,
3147, 9808, 129029]
```

[28]: <BarContainer object of 5 artists>



0.3 Linear regression with verified purchase and review length

[29]:
```
len(dataset)
dataset2 = [d for d in dataset if d['review_body']]
len(dataset2)


def feature1(datum):
  feat = [1]
  if datum['verified_purchase'] == "Y":
      feat.append(1)
  else:
      feat.append(0)
  feat.append(len(datum['review_body']))
  return feat

X1 = [feature1(d) for d in dataset2]
Y1 = [d['star_rating'] for d in dataset2]
theta1 = np.linalg.lstsq(X1, Y1)

theta1[0]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:17: FutureWarning:
`rcond` parameter will change to the default of machine precision times ``max(M,
N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
```

[29]:
```
array([ 4.84502099e+00,  4.98709142e-02, -1.24544521e-03])
```

Solution 3- The theta coefficients represent the sensitivity of each of the parameters with an offset value. The ratings are positively correlated with the reviews that are verified meaning that verified purchased tend to give slightly higher ratings but the increase is very small. The negative value of theta value corresponding to the review_body length says that as the size of the review increases,the review usually becomes more negative and rating decreases.

0.4 Linear regression with verified purchase

[30]:
```
def feature2(datum):
  feat = [1]
  if datum['verified_purchase'] == "Y":
      feat.append(1)
  else:
      feat.append(0)
  return feat

X2 = [feature2(d) for d in dataset]
Y2 = [d['star_rating'] for d in dataset]
theta2 = np.linalg.lstsq(X2, Y2)

theta2[0]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: FutureWarning:
`rcond` parameter will change to the default of machine precision times ``max(M,
N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  # This is added back by InteractiveShellApp.init_path()
```

[30]:
```
array([4.578143   , 0.16793392])
```

Solution 4- The coefficients for both the models are different. This is because the parameter review body length is removed from the matrix and hence the sensitivity of review being verified increases with respect to ratings given. In other words, the sum of coefficients of both the models remain almost the same, but the distribution of the values changes depending on the parameter.

0.5 MSE on splitting the data in 90% train and 10% test set

[31]:
```
train_dataset = dataset[:int(0.9*len(dataset))]
test_dataset = dataset[int(0.9*len(dataset)):]

X3 = [feature2(d) for d in train_dataset]
Y3 = [d['star_rating'] for d in train_dataset]
theta3 = np.linalg.lstsq(X3, Y3)
YP3 = np.matmul(X3,theta3[0])

MSE1 = np.square(np.subtract(Y3,YP3)).mean()

X4 = [feature2(d) for d in test_dataset]
Y4 = [d['star_rating'] for d in test_dataset]
YP4 = np.matmul(X4,theta3[0])
MSE2 = np.square(np.subtract(Y4,YP4)).mean()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: FutureWarning:
`rcond` parameter will change to the default of machine precision times ``max(M,
N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
```

[32]:
```
MSE1
```

[32]:
```
0.6554842196694356
```

[33]:
```
MSE2
```

[33]:
```
0.9723851990304192
```

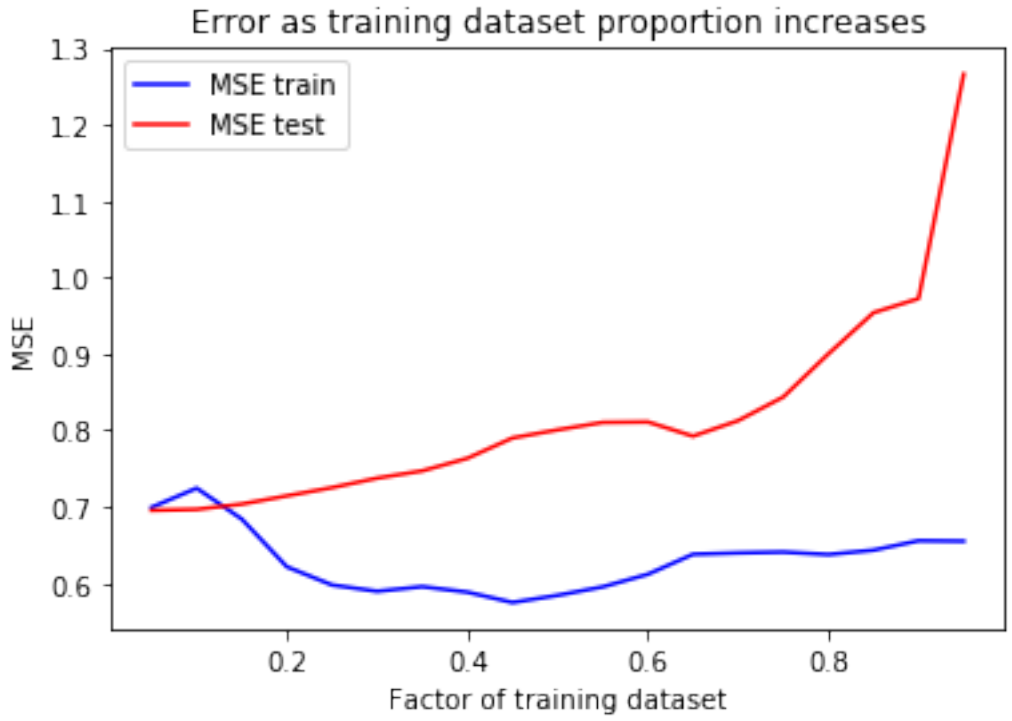
0.6 Plot varying training data size

[34]:
```
factor =[]
for i in range(5, 100, 5):
    factor.append(i/100.0)


MSEtrain =[]
MSEtest =[]
for i,j in enumerate(factor):
    train = dataset[:int(j * len(dataset))]
    test = dataset[int(j * len(dataset)):]
    Xtrain = [feature2(d) for d in train]
    Ytrain = [d['star_rating'] for d in train]
```

```
    ttrain = np.linalg.lstsq(Xtrain, Ytrain)
    YPtrain = np.matmul(Xtrain, ttrain[0])
    MSEtrain.append( np.square(np.subtract(Ytrain, YPtrain)).mean() )
    Xtest = [feature2(d) for d in test]
    Ytest = [d['star_rating'] for d in test]
    YPtest = np.matmul(Xtest, ttrain[0])
    MSEtest.append(np.square(np.subtract(Ytest, YPtest)).mean())

plt.close()
plt.xlabel("Factor of training dataset")
plt.ylabel("MSE")
plt.title("Error as training dataset proportion increases")
plt.plot(factor,MSEtrain,color = "blue", label = "MSE train")
plt.plot(factor,MSEtest,color = "red",label = "MSE test")
plt.legend()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarning:
`rcond` parameter will change to the default of machine precision times ``max(M,
N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  del sys.path[0]
```

[34]: <matplotlib.legend.Legend at 0x7fdb021124e0>



Solution 7 - Yes the size of the training dataset has a significant effect on the test error. In this instance since only a single parameter is considered to predict the rating, the results cannot be considered to be the most accurate. The test error increases drastically as the training data proportion approaches 95%, this is because training on large data results in a generalised theta coefficient values and when applied on the small test dataset, it cannot predict the rating accurately increasing the error value.

0.7 Classification

[35]:
```
def feature3(datum):
  feat = [1]
  feat.append(datum['star_rating'])
  feat.append(len(datum['review_body']))
  return feat

XC1 = [feature3(d) for d in dataset2]
YC1 = []
for datum in dataset2:
  if datum['verified_purchase'] == "Y":
      YC1.append(1)
  else:
      YC1.append(0)


XCtrain = XC1[:int(0.9*len(dataset2))]
YCtrain = YC1[:int(0.9*len(dataset2))]

XCtest = XC1[int(0.9*len(dataset2)):]
YCtest = YC1[int(0.9*len(dataset2)):]

len(XC1), len(XCtrain), len(XCtest)

model = linear_model.LogisticRegression()
model.fit(XCtrain, YCtrain)

predtrain = model.predict(XCtrain)
predtest = model.predict(XCtest)

correctpredtrain = predtrain == YCtrain
correctpredtest = predtest == YCtest
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

Train accuracy and Test accuracy (Classification accuracy)

[36]:
```
sum(correctpredtrain)*1.0 / len(correctpredtrain)
```

[36]: 0.9511754746100481

[37]:
```
sum(correctpredtest)*1.0 / len(correctpredtest)
```

[37]: 0.5597734475085968

Proportion of labels that are positive and Proportion of predictions that are positive

[38]:
```
sum(d['verified_purchase'] == "Y" for d in dataset2)*1.0/len(dataset2)
```

[38]: 0.9122103176476141

[39]:
```
(sum(correctpredtest)+sum(correctpredtrain))*1.0/len(dataset2)
```

[39]: 0.9120350079900748

0.8 Designed feature vector for more accuracy

Clearly as the above solution suggests there is a class imbalance in the dataset. We need to decide a more effective parameter to model the feature vector. The dataset is ordered by timestamp values and it makes sense to shuffle the data to reduce the bias in the timestamp used to train the model. Unverified purchases are a recent phenomena and corresponding review timestamps are taken into account. Keywords like 'love','easy' are usually associated with verified purchases and hence used as predictors to train the model.

[40]:
```
def classaccuracy():
    np.random.shuffle(dataset2)
    def feature4(datum):
        feat = [1]
        try:
            a = ['discount','love','easy']
            if any(x in datum['review_body'] for x in a):
                feat.append(1)
            else:
                feat.append(0)
            if 1404284400 < (time.mktime(time.strptime(datum['review_date'],␣
↪"%Y-%m-%d"))) < 1441004400.00:
                feat.append(1)
            else:
                feat.append(0)
        except KeyError:
           feat.append(0)
        return feat
```

```
    XC2 = [feature4(d) for d in dataset2]

    YC2 = []
    for datum in dataset2:
        if datum['verified_purchase'] == "Y":
            YC2.append(1)
        else:
            YC2.append(0)


    XC2train = XC2[:int(0.9*len(dataset2))]
    YC2train = YC2[:int(0.9*len(dataset2))]

    XC2test = XC2[int(0.9*len(dataset2)):]
    YC2test = YC2[int(0.9*len(dataset2)):]

    len(XC2), len(XC2train), len(XC2test)

    model2 = linear_model.LogisticRegression()
    model2.fit(XC2train, YC2train)

    pred2train = model2.predict(XC2train)
    pred2test = model2.predict(XC2test)

    correctpredtrain2 = pred2train == YC2train
    correctpredtest2 = pred2test == YC2test

    return sum(correctpredtrain2)*1.0 /␣
↪len(correctpredtrain2),sum(correctpredtest2)*1.0 / len(correctpredtest2)

print("Train and test accuracy of the model = \n"+ str(classaccuracy()))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)

Train and test accuracy of the model =
(0.9118506420533721, 0.9154473737441845)
```

As shown above this model results in a significant increase in accuracy.

[ ]: