CSE-258 Homework 1

October 11, 2021

[1]:
```python
# Importing Libraries
import numpy as np
import scipy.optimize
import random
import ast
```

[2]:
```python
def parseDataFromFile(fname):
    for l in open(fname):
        yield eval(l)
```

[3]:
```python
data=list(parseDataFromFile("fantasy_10000.json"))
```

[4]:
```python
# Printing First Data
print(data[0])
```


```
{'user_id': '8842281e1d1347389f2ab93d60773d4d', 'book_id': '18245960',
'review_id': 'dfdbb7b0eb5a7e4c26d59a937e2e5feb', 'rating': 5, 'review_text':
'This is a special book. It started slow for about the first third, then in the
middle third it started to get interesting, then the last third blew my mind.
This is what I love about good science fiction - it pushes your thinking about
where things can go. \n It is a 2015 Hugo winner, and translated from its
original Chinese, which made it interesting in just a different way from most
things I\'ve read. For instance the intermixing of Chinese revolutionary history
- how they kept accusing people of being "reactionaries", etc. \n It is a book
about science, and aliens. The science described in the book is impressive - its
a book grounded in physics and pretty accurate as far as I could tell. Though
when it got to folding protons into 8 dimensions I think he was just making
stuff up - interesting to think about though. \n But what would happen if our
SETI stations received a message - if we found someone was out there - and the
person monitoring and answering the signal on our side was disillusioned? That
part of the book was a bit dark - I would like to think human reaction to
discovering alien civilization that is hostile would be more like Enders Game
where we would band together. \n I did like how the book unveiled the Trisolaran
culture through the game. It was a smart way to build empathy with them and also
understand what they\'ve gone through across so many centuries. And who know a 3
body problem was an unsolvable math problem? But I still don\'t get who made the
game - maybe that will come in the next book. \n I loved this quote: \n "In the
long history of scientific progress, how many protons have been smashed apart in
```

```
accelerators by physicists? How many neutrons and electrons? Probably no fewer
than a hundred million. Every collision was probably the end of the
civilizations and intelligences in a microcosmos. In fact, even in nature, the
destruction of universes must be happening at every second--for example, through
the decay of neutrons. Also, a high-energy cosmic ray entering the atmosphere
may destroy thousands of such miniature universes..."', 'date_added': 'Sun Jul
30 07:44:10 -0700 2017', 'date_updated': 'Wed Aug 30 00:00:26 -0700 2017',
'read_at': 'Sat Aug 26 12:05:52 -0700 2017', 'started_at': 'Tue Aug 15 13:23:18
-0700 2017', 'n_votes': 28, 'n_comments': 1}
```

[5]:
```
# 2. Train a simple predictor that estimates rating from review length, i.e.,␣
↪star rating  0 + 1 × [review length in characters]
```

[6]:
```
# 2. Review Length Predictor
# def feature
def feature(datum):
    f=[1]
    f.append(len(datum["review_text"]))
    return(f)
```

[7]:
```
X=[feature(d) for d in data]
```

[8]:
```
print(X[:10])
```

```
[[1, 2086], [1, 1521], [1, 1519], [1, 1791], [1, 1762], [1, 470], [1, 823], [1,
532], [1, 616], [1, 548]]
```

[9]:
```
def y_rating(datum):
    f=[]
    f.append(datum["rating"])
    return(f)
```

[10]:
```
y=[y_rating(d) for d in data]
print(y[0:10])
```

```
[[5], [5], [5], [4], [3], [5], [5], [5], [4], [5]]
```

[11]:
```
theta,residual,rank,s=np.linalg.lstsq(X,y)
```

```
/Users/kakshak/opt/anaconda3/envs/tf/lib/python3.7/site-
packages/ipykernel_launcher.py:1: 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`.
  """Entry point for launching an IPython kernel.
```

[12]:

```
#Printing Values of Theta 0 and 1
print(theta)
```

```
[[3.68568136e+00]
 [6.87371675e-05]]
```

[13]:

```
X=np.matrix(X)
y=np.matrix(y)
print(theta)
```

```
[[3.68568136e+00]
 [6.87371675e-05]]
```

[14]:

```
# Calculating MSE for Second Question
z=(X)*(theta)
o=y-z
# Printing MSE
sum(np.asarray(o).flatten()**2)/len(data)
```

[14]: 1.5522086622355353

[15]:

```
# 3. Extend your model to include (in addition to the length) features based on␣
 ↪the time of the review.
```

[16]:

```
# 3 Question Date and Time
import dateutil.parser
t1 = dateutil.parser.parse(data[0]['date_added'])
t1.weekday(), t1.year # etc.
```

[16]: (6, 2017)

[17]:

```
feature_matrix=[]
week=[]
year=[]
for i in range(len(data)):
    matrix=[]
    t1 = dateutil.parser.parse(data[i]['date_added'])
    week.append(t1.weekday())
    year.append(t1.year)
    matrix.append(1)
    matrix.append(len(data[i]["review_text"]))
    matrix.append(t1.weekday())
    matrix.append(t1.year)
    feature_matrix.append(matrix)
```

[18]:

```
# Feature Normal Feature Matrix of first 2 elements(Length, Week, Year)
print("Feature Matrix of First Element Data",feature_matrix[0])
```

```
print("Feature Matrix of Second Element Data",feature_matrix[1])
```

```
Feature Matrix of First Element Data [1, 2086, 6, 2017]
Feature Matrix of Second Element Data [1, 1521, 2, 2014]
```

[19]:
```
#One Hot Encoding for Week
week_encoding=[]
for i in range(len(week)):
    o=[0]*6
    if(week[i]!=0):
        o[week[i]-1]=1
    week_encoding.append(o)
```

[20]:
```
#One Hot Encoding for Year
ma=max(year)
mi=min(year)
year_encoding=[]
for i in range(len(year)):
    o=[0]*(ma-mi)
    if((year[i]-mi)!=0):
        o[year[i]-mi-1]=1
    year_encoding.append(o)
```

[21]:
```
# Printing Feature Matrix of One Hot Encoded form of Week and Year
X=[]
for i in range(len(data)):
    k=[]
    z=len(data[i]["review_text"])
    k.append(1)
    k.append(z)
    for j in range(len(week_encoding[0])):
        k.append(week_encoding[i][j])
    for j in range(len(year_encoding[0])):
        k.append(year_encoding[i][j])
    X.append(k)
print("Feature Matrix of First Element using one-hot encoded from Data: ",X[0])
print("Feature Matrix of Second Element one-hot encoded form Data: ",X[1])
```

```
Feature Matrix of First Element using one-hot encoded from Data:  [1, 2086, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
Feature Matrix of Second Element one-hot encoded form Data:  [1, 1521, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
```

[22]:
```
# 4. Using the weekday and year values directly and using one hot encoded form␣
↪as features, i.e., star rating  0 + 1 × [review length in characters] + 2 × [t.
↪weekday()] + 3 × [t.year]
```

[23]:
```
# 4 Question (a)
X=[]
y=[]
for i in range(len(data)):
    k=[]
    z=len(data[i]["review_text"])
    k.append(1)
    k.append(z)
    k.append(week[i])
    k.append(year[i])
    X.append(k)
    y.append(data[i]["rating"])
```

[24]:
```
theta,residual,rank,s=np.linalg.lstsq(X,y)
```

```
/Users/kakshak/opt/anaconda3/envs/tf/lib/python3.7/site-
packages/ipykernel_launcher.py:1: 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`.
  """Entry point for launching an IPython kernel.
```

[25]:
```
#Printing Values of Theta 0, 1 and 2
print(theta)
```

```
[-1.01742461e+02  5.50923292e-05  8.75072300e-03  5.23592268e-02]
```

[26]:
```
X=np.matrix(X)
y=np.matrix(y)
```

[27]:
```
# Calculating MSE
z=(theta)*(X.T)
o=y-z
# Printing MSE
sum(np.asarray(o).flatten()**2)/len(data)
```

[27]: 1.5367740498705302

[28]:
```
# 4 Question (b)
X=[]
y=[]
for i in range(len(data)):
    k=[]
    z=len(data[i]["review_text"])
    k.append(1)
    k.append(z)
```

```python
    for j in range(len(week_encoding[0])):
        k.append(week_encoding[i][j])
    for j in range(len(year_encoding[0])):
        k.append(year_encoding[i][j])
    X.append(k)
    y.append(data[i]["rating"])
```

[29]:
```python
theta,residual,rank,s=np.linalg.lstsq(X,y)
```

```
/Users/kakshak/opt/anaconda3/envs/tf/lib/python3.7/site-
packages/ipykernel_launcher.py:1: 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`.
  """Entry point for launching an IPython kernel.
```

[30]:
```python
print(theta)
```

```
[ 4.87171479e+00  5.15709386e-05  4.89003441e-02  1.45709798e-01
  1.06646403e-01  1.26168316e-01  3.83417660e-02  1.02846903e-01
 -1.58244783e+00 -1.70447417e+00 -1.68316056e+00 -1.67023905e+00
 -1.62877001e+00 -1.19956705e+00 -1.10444816e+00 -1.09162361e+00
 -1.20861354e+00 -1.23647487e+00 -1.23331225e+00]
```

[31]:
```python
X=np.matrix(X)
y=np.matrix(y)
```

[32]:
```python
# Calculating MSE
z=(theta)*(X.T)
o=y-z
# Printing MSE
sum(np.asarray(o).flatten()**2)/len(data)
```

[32]: 1.512357865642828

[33]:
```python
# 5 Repeat the above question, but this time split the data into a training and␣
↪test set. You should split the data randomly into 50%/50% train/test fractions.
↪ Report the MSE of each model separately on the training and test sets.
```

[34]:
```python
# Using Normal Feature Vectors of Review Length, Week, Year
```

[35]:
```python
# 5 Question
random.shuffle(data)
X=[]
y=[]
for i in range(len(data)):
```

```
    k=[]
    z=len(data[i]["review_text"])
    k.append(1)
    k.append(z)
    k.append(week[i])
    k.append(year[i])
    X.append(k)
    y.append(data[i]["rating"])
```

[36]:
```
# from sklearn.model_selection import train_test_split
# X_train, X_test, y_train, y_test = train_test_split(X, y,train_size =0.5)
N=len(X)
X_train=X[:N//2]
X_test=X[N//2:]
y_train=y[:N//2]
y_test=y[N//2:]
```

[37]:
```
theta,residual,rank,s=np.linalg.lstsq(X_train,y_train)
```

```
/Users/kakshak/opt/anaconda3/envs/tf/lib/python3.7/site-
packages/ipykernel_launcher.py:1: 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`.
  """Entry point for launching an IPython kernel.
```

[38]:
```
print(theta)
```

```
[-1.76178449e+01  6.34591175e-05 -1.67033321e-04  1.05797468e-02]
```

[39]:
```
# MSE of Train - Normal Feature Model
```

[40]:
```
X_train=np.matrix(X_train)
y_train=np.matrix(y_train)
```

[41]:
```
# Calculating MSE
z=(theta)*(X_train.T)
o=y_train-z
# Printing MSE
sum(np.asarray(o).flatten()**2)/(len(data)/2)
```

[41]: 1.54288491705003

[42]:
```
# MSE of Test - Normal Feature Model
```

[43]:
```
X_test=np.matrix(X_test)
y_test=np.matrix(y_test)
```

[44]:
```
# Calculating MSE
z=(theta)*(X_test.T)
o=y_test-z
# Printing MSE
sum(np.asarray(o).flatten()**2)/(len(data)/2)
```

[44]: 1.5610371069154052

[45]:
```
# Using One Hot Encoding Feature Vectors of Review Length, Week, Year
```

[46]:
```
X=[]
y=[]
for i in range(len(data)):
    k=[]
    z=len(data[i]["review_text"])
    k.append(1)
    k.append(z)
    for j in range(len(week_encoding[0])):
        k.append(week_encoding[i][j])
    for j in range(len(year_encoding[0])):
        k.append(year_encoding[i][j])
    X.append(k)
    y.append(data[i]["rating"])
```

[47]:
```
# from sklearn.model_selection import train_test_split
# X_train, X_test, y_train, y_test = train_test_split(X, y,train_size =0.5)
N=len(X)
X_train=X[:N//2]
X_test=X[N//2:]
y_train=y[:N//2]
y_test=y[N//2:]
```

[48]:
```
theta,residual,rank,s=np.linalg.lstsq(X_train,y_train)
```

```
/Users/kakshak/opt/anaconda3/envs/tf/lib/python3.7/site-
packages/ipykernel_launcher.py:1: 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`.
  """Entry point for launching an IPython kernel.
```

[49]:
```
print(theta)
```

```
[ 4.24826265e+00  6.45709250e-05  3.96342769e-02  7.72601716e-02
```

```
 3.94895301e-02  3.65881964e-02  3.15469351e-02  1.76938460e-02
-9.83219675e-01 -6.46124091e-01 -5.19851248e-01 -7.48875834e-01
-6.59941879e-01 -6.11180204e-01 -5.77181981e-01 -6.62477782e-01
-5.55192726e-01 -5.79856023e-01 -5.53563798e-01]
```

[50]:
```
# MSE of Train - One Hot Encoding Feature Model
```

[51]:
```
X_train=np.matrix(X_train)
y_train=np.matrix(y_train)
```

[52]:
```
# Calculating MSE
z=(theta)*(X_train.T)
o=y_train-z
# Printing MSE
sum(np.asarray(o).flatten()**2)/(len(data)/2)
```

[52]: 1.5400050742288256

[53]:
```
# MSE of Test - One Hot Encoding Feature Model
```

[54]:
```
X_test=np.matrix(X_test)
y_test=np.matrix(y_test)
```

[55]:
```
# Calculating MSE
z=(theta)*(X_test.T)
o=y_test-z
# Printing MSE
sum(np.asarray(o).flatten()**2)/(len(data)/2)
```

[55]: 1.5622872320337204

6) It is given that for a trivial predictor $y = \theta_0$

Mean Absolute Error for n training examples $= \dfrac{\sum_{i=1}^{n} |y_i - \theta_0|}{n}$

Taking derivative of Mean Absolute Error with respect to θ_0

$$\frac{d\,MAE}{d\theta_0} = \frac{d}{d\theta_0}\left(\frac{1}{n}\sum_{i=1}^{n} |y_i - \theta_0|\right)$$

$$\frac{d\,MAE}{d\theta_0} = \begin{cases} +1 & y_{true} > \theta_0 \\ -1 & y_{true} < \theta_0 \end{cases}$$

To get the most optimum value of θ_0,

$$\frac{d\,(MAE)}{d\theta_0} = 0$$

Therefore $\frac{d(MAE)}{d\theta_0} = 0$, so there

should be equal number of terms having 1 (when $y_{true} > \theta_0$) and -1 (when $y_{true} < \theta_0$, and rest all can be 0.

So θ_0 must be greater than half of the y_{true} terms and θ_0 must be less than half of y_{true} terms. Therefore θ_0 must be median of $\{y_1, y_2 \ldots\ldots y_n\}$.

CSE-258 Homework 1

October 11, 2021

[1]:
```
# Tasks - Classification
```

[2]:
```
# Importing Dataset and Libraries
```

[3]:
```
import numpy as np
import scipy.optimize
import random
import ast
from sklearn import linear_model
```

[4]:
```
def parseDataFromFile(fname):
    for l in open(fname):
        yield eval(l)
```

[5]:
```
data=list(parseDataFromFile("beer_50000.json"))
print(data[0])
```

```
{'review/appearance': 2.5, 'beer/style': 'Hefeweizen', 'review/palate': 1.5,
'review/taste': 1.5, 'beer/name': 'Sausa Weizen', 'review/timeUnix': 1234817823,
'beer/ABV': 5.0, 'beer/beerId': '47986', 'beer/brewerId': '10325',
'review/timeStruct': {'isdst': 0, 'mday': 16, 'hour': 20, 'min': 57, 'sec': 3,
'mon': 2, 'year': 2009, 'yday': 47, 'wday': 0}, 'review/overall': 1.5,
'review/text': 'A lot of foam. But a lot.\tIn the smell some banana, and then
lactic and tart. Not a good start.\tQuite dark orange in color, with a lively
carbonation (now visible, under the foam).\tAgain tending to lactic
sourness.\tSame for the taste. With some yeast and banana.', 'user/profileName':
'stcules', 'review/aroma': 2.0}
```

[6]:
```
# 7. Fit a logistic regressor that estimates the binarized score from review␣
↪length, i.e., p(rating is positive)  (0 + 1 × [length]) Using the class␣
↪weight='balanced' option, report the True Positive, True Negative, False␣
↪Positive, False Negative, and Balanced Error Rates of the predictor
```

[7]:
```
y = [d["review/overall"] >= 4 for d in data]
```

[8]:
```
def feature(datum):
    f=[1]
```

```
    f.append(len(datum["review/text"]))
    return(f)
```

[9]:
```
X=[feature(d) for d in data]
```

[10]:
```
print(X[:3])
```

```
[[1, 262], [1, 338], [1, 396]]
```

[11]:
```
print(y[:10])
```

```
[False, False, False, False, True, False, False, False, True, True]
```

[12]:
```
mod = linear_model.LogisticRegression(C=1.0, class_weight='balanced')
mod.fit(X,y)
```

[12]:
```
LogisticRegression(class_weight='balanced')
```

[13]:
```
predict=mod.predict(X)
```

[14]:
```
correct= predict==y
```

[15]:
```
TP_c = np.logical_and(predict, y)
FP_c = np.logical_and(predict, np.logical_not(y))
TN_c = np.logical_and(np.logical_not(predict), np.logical_not(y))
FN_c = np.logical_and(np.logical_not(predict), y)
```

[16]:
```
TP = sum(TP_c)
FP = sum(FP_c)
TN = sum(TN_c)
FN = sum(FN_c)
print("True Positive : ",TP)
print("False Positive : ",FP)
print("True Negative : ",TN)
print("False Negative : ",FN)
TPR=TP/(TP+FN)
FPR=1-TPR
TNR=TN/(TN+FP)
FNR=1-TNR
print("True Positive Rate : ",TPR)
print("False Positive Rate: ",FPR)
print("True Negative Rate: ",TNR)
print("False Negative Rate : ",FNR)
```

```
True Positive :  14201
False Positive :  5885
True Negative :  10503
False Negative :  19411
```

```
True Positive Rate :  0.4224979174104487
False Positive Rate:  0.5775020825895514
True Negative Rate:  0.6408957773980962
False Negative Rate :  0.3591042226019038
```

[17]:
```
# accuracy
print(sum(correct) / len(correct))
accuracy=(TP + TN) / (TP + FP + TN + FN)
print("Accuracy: ",accuracy)
```

```
0.49408
Accuracy:  0.49408
```

[18]:
```
# BER
BER=1 - 0.5 * (TP / (TP + FN) + TN / (TN + FP))
print("Balanced Error Rate: ",BER)
```

```
Balanced Error Rate:  0.4683031525957275
```

[19]:
```
# 8. Plot the precision@K of your classifier for K = {1 . . . 10000} (i.e., the␣
↪x-axis of your plot should be K, and the y-axis of your plot should be the␣
↪precision@K)
```

[20]:
```
confidence = mod.decision_function(X)
confidenceandLabels=list(zip(confidence,y))
```

[21]:
```
confidenceandLabels.sort()
confidenceandLabels.reverse()
print(confidenceandLabels[0:5])
```

```
[(1.4203973087838948, True), (1.408714811886357, True), (1.3478242219961605,
True), (1.3127767313035474, True), (1.2858715869334605, True)]
```

[22]:
```
labelsrank=[z[1] for z in confidenceandLabels]
```

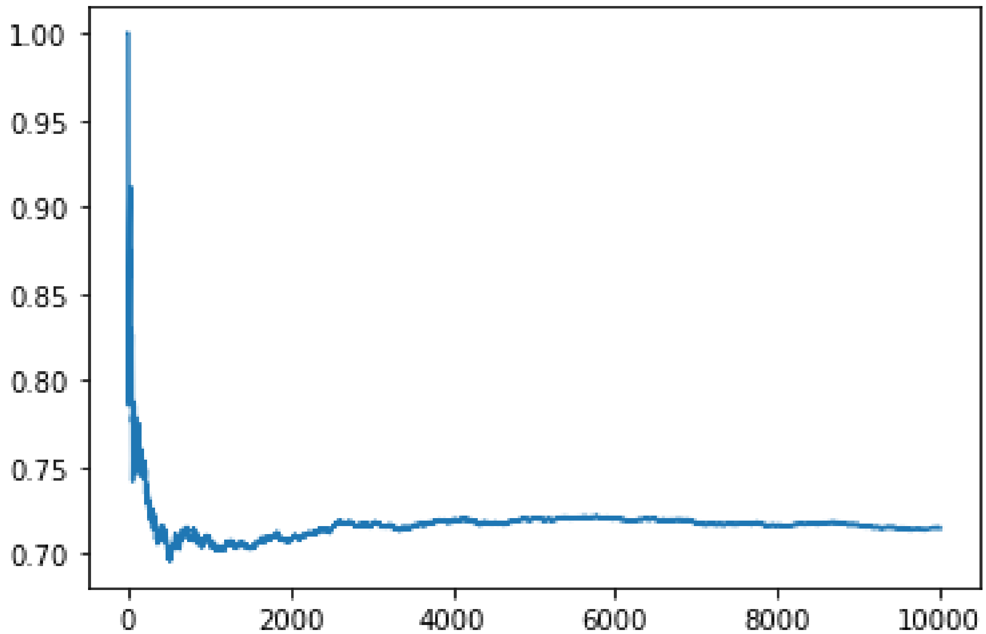
[23]:
```
o=[]
def precisionK(K,y_sorted):
    w1=[]
    w1.append(K)
    w1.append(sum(y_sorted[:K])/K)
    o.append(w1)
```

[24]:
```
for i in range(1,10001):
    precisionK(i,labelsrank)
```

[25]:
```
import matplotlib.pyplot as plt
xs = [x[0] for x in o]
ys = [x[1] for x in o]
```

```
plt.plot(xs, ys)
```

[25]: [<matplotlib.lines.Line2D at 0x7f9ab68b09d0>]



[26]:
```
# 9. Our precision@K plot from Question 8 only measures precision with regard to
↪the positive class. For this type of binary classification, we may be equally
↪interested in the classifier's accuracy for both the positive and negative
↪classes. Recompute confidence scores for your classifier so that the 'most
↪confi- dent' predictions include either the most confident positive or the
↪most confident negative predictions (i.e., probability closest to 1 or
↪probability closest to zero).3 The precision@K now measures whether the
↪classifier has the correct label (either 'positive' or 'negative') among the K
↪most confident entries. Report this precision@K for K  {1, 100, 10000} and
↪include a plot as in Question 8.
```

[27]:
```
confidence = mod.decision_function(X)
confidenceLabels=list(zip(confidence,y))
print(confidenceLabels[0:5])
```

```
[(-0.1581558326137028, False), (-0.1312506882436159, False),
(-0.1107178149085496, False), (-0.1089477396210439, False),
(0.15443946315980667, True)]
```

[28]:
```
q=[]
for i in range(len(confidenceLabels)):
    w=[]
```

```
    if(confidenceLabels[i][0]<0 and confidenceLabels[i][1]==False):
        w.append(abs(confidenceLabels[i][0]))
        w.append(confidenceLabels[i][1])
        w.append(1)
        q.append(w)
    elif(confidenceLabels[i][0]>0 and confidenceLabels[i][1]==True):
        w.append(confidenceLabels[i][0])
        w.append(confidenceLabels[i][1])
        w.append(1)
        q.append(w)
    elif(confidenceLabels[i][0]<0 and confidenceLabels[i][1]==True):
        w.append(abs(confidenceLabels[i][0]))
        w.append(confidenceLabels[i][1])
        w.append(0)
        q.append(w)
    elif(confidenceLabels[i][0]>0 and confidenceLabels[i][1]==False):
        w.append(confidenceLabels[i][0])
        w.append(confidenceLabels[i][1])
        w.append(0)
        q.append(w)
q.sort(key=lambda item :item[0],reverse=True)
print(q[0:5])
```

```
[[1.4203973087838948, True, 1], [1.408714811886357, True, 1],
[1.3478242219961605, True, 1], [1.3127767313035474, True, 1],
[1.2858715869334605, True, 1]]
```

[29]:
```
p23=[]
for i in range(len(q)):
    p23.append(q[i][2])
```

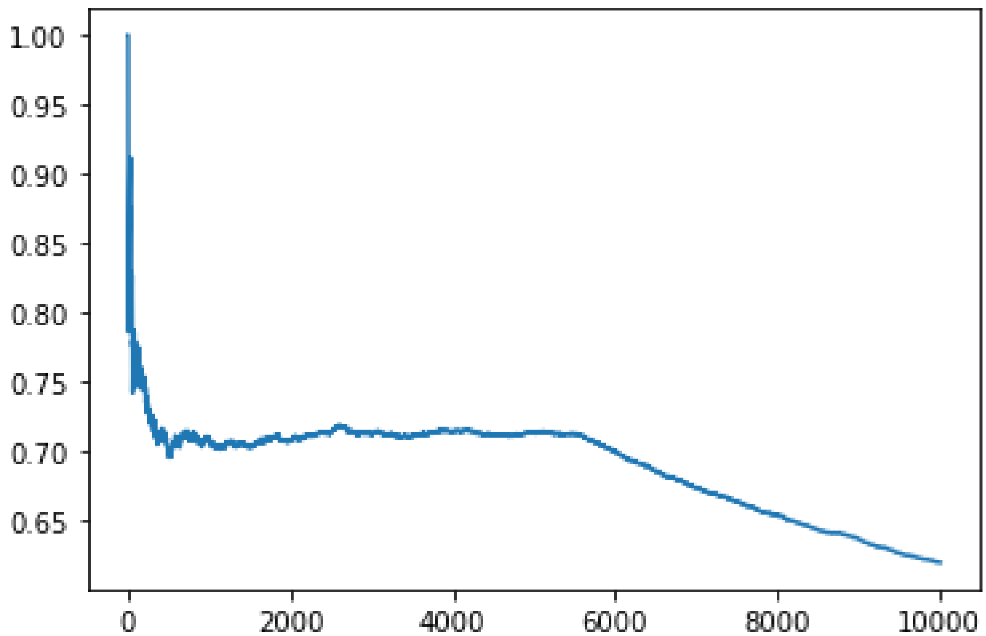
[30]:
```
o=[]
def precisionK(K,y_sorted):
    w1=[]
    w1.append(K)
    w1.append(sum(y_sorted[:K])/K)
    if(K==1):
        print("Precision@1 : ",sum(y_sorted[:K])/K)
    if(K==100):
        print("Precision@100 : ",sum(y_sorted[:K])/K)
    if(K==10000):
        print("Precision@10000 : ",sum(y_sorted[:K])/K)
    o.append(w1)
```

[31]:
```
for i in range(1,10001):
    precisionK(i,p23)
```

```
Precision@1 :  1.0
Precision@100 :  0.75
Precision@10000 :  0.6196
```

[32]:
```python
import matplotlib.pyplot as plt
xs = [x[0] for x in o]
ys = [x[1] for x in o]
plt.plot(xs, ys)
```

[32]:
```
[<matplotlib.lines.Line2D at 0x7f9aa1b6a1f0>]
```



[33]:
```python
confidence = mod.decision_function(X)
confidenceandLabels=list(zip(confidence,y))
confidence = mod.decision_function(X)
confidenceLabels=list(zip(confidence,y))
```

[34]:
```python
probab=mod.predict_proba(X)
```

[35]:
```python
prob=[]
for i in range(len(probab)):
    prob.append(probab[i][1])
```

[36]:
```python
for i in range(len(prob)):
    prob[i]=prob[i]-0.5
```

[37]:
```
z1=[]
for i in range(len(confidenceandLabels)):
    z2=[]
    z2.append(prob[i])
    z2.append(confidenceandLabels[i][1])
    z1.append(z2)
```

[38]:
```
q=[]
for i in range(len(z1)):
    w=[]
    if(z1[i][0]<0 and z1[i][1]==False):
        w.append(abs(z1[i][0]))
        w.append(z1[i][1])
        w.append(1)
        q.append(w)
    elif(z1[i][0]>0 and z1[i][1]==True):
        w.append(z1[i][0])
        w.append(z1[i][1])
        w.append(1)
        q.append(w)
    elif(z1[i][0]<0 and z1[i][1]==True):
        w.append(abs(z1[i][0]))
        w.append(z1[i][1])
        w.append(0)
        q.append(w)
    elif(z1[i][0]>0 and z1[i][1]==False):
        w.append(z1[i][0])
        w.append(z1[i][1])
        w.append(0)
        q.append(w)
q.sort(key=lambda item :item[0],reverse=True)
print(q[0:5])
```

```
[[0.30540069433524597, True, 1], [0.3035631565911836, True, 1],
[0.29377368747531496, True, 1], [0.28797743254902197, True, 1],
[0.2834475927473613, True, 1]]
```

[39]:
```
p23=[]
for i in range(len(q)):
    p23.append(q[i][2])
```

[40]:
```
o=[]
def precisionK(K,y_sorted):
    w1=[]
    w1.append(K)
    w1.append(sum(y_sorted[:K])/K)
    if(K==1):
```

```
        print("Precision@1 : ",sum(y_sorted[:K])/K)
    if(K==100):
        print("Precision@100 : ",sum(y_sorted[:K])/K)
    if(K==10000):
        print("Precision@10000 : ",sum(y_sorted[:K])/K)
    o.append(w1)
```

[41]:
```
for i in range(1,10001):
    precisionK(i,p23)
```

```
Precision@1 :  1.0
Precision@100 :  0.75
Precision@10000 :  0.6196
```

[42]:
```
import matplotlib.pyplot as plt
xs = [x[0] for x in o]
ys = [x[1] for x in o]
plt.plot(xs, ys)
```

[42]: [<matplotlib.lines.Line2D at 0x7f9a90e02fd0>]