homework_1

October 10, 2021

1 Tasks — Regression (week 1):

1.0.1 Task 2:

[134]:
```
import pandas as pd
import numpy as np

reviews = pd.read_json('https://cseweb.ucsd.edu/classes/fa21/cse258-b/data/
 ↪fantasy_10000.json.gz', lines=True)
```

[135]:
```
ratings = [review for review in reviews["rating"]]
lengths = [len(review) for review in reviews["review_text"]]

y = np.matrix(ratings).T
X = np.matrix([[1,length] for length in lengths])

theta = np.linalg.inv(X.T*X)*X.T*y
print("Theta([Theta_0; Theta_1]): \n", theta)
```

```
Theta([Theta_0; Theta_1]):
 [[3.68568136e+00]
 [6.87371675e-05]]
```

[136]:
```
sse = sum([x**2 for x in (y - X*theta)]) #Summing up all square errors
mse = sse / len(y) #Dividing by length to get mean square error
print("MSE = ", mse)
```

```
MSE =  [[1.55220866]]
```

1.0.2 Task 3:

[137]:
```
import dateutil.parser
dates = [dateutil.parser.parse(review_time) for review_time in␣
 ↪reviews["date_added"]]
```

3. To use the piecewise function implementation of one-hot encoding the weekdays, we need a 1×6 vector:

$$
\begin{aligned}
&[0\ 0\ 0\ 0\ 0\ 0] \text{ if monday} \\
&[1\ 0\ 0\ 0\ 0\ 0] \text{ if tuesday} \\
&[0\ 1\ 0\ 0\ 0\ 0] \text{ if wednesday} \\
&\quad\vdots \qquad\qquad\qquad \vdots \\
&[0\ 0\ 0\ 0\ 0\ 1] \text{ if sunday}
\end{aligned}
$$

To one-hot encode the years in the same general manner is a bit worse, since there in theory might be an infinite amount of years. However, since I know the years in this dataset ranges from 2006 to 2017, I took the liberty of limiting the encoding to these years. We then need a 1×11 vector:

$$
\begin{aligned}
&[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0] \text{ if } 2006 \\
&[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0] \text{ if } 2007 \\
&[0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0] \text{ if } 2008 \\
&\quad\vdots \\
&[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1] \text{ if } 2017
\end{aligned}
$$

The feature vectors for the first two samples then becomes as follows:

$$x_1 = [1\ \overbrace{2086}^{\text{length}}\ \overbrace{0\ 0\ 0\ 0\ 0\ 1}^{\text{day(sunday)}}\ \overbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1}^{\text{year(2017)}}]$$

$$x_2 = [1\ \overbrace{1521}^{\text{length}}\ \overbrace{0\ 1\ 0\ 0\ 0\ 0}^{\text{day(wednesday)}}\ \overbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0}^{\text{year(2014)}}]$$

1.0.3 Task 4:

[138]:
```
#Using values directly as features
def feature(index):
    feature = [1]
    feature.append(lengths[index])
```

```
    feature.append(dates[index].weekday())
    feature.append(dates[index].year-2006)
    return feature

y = np.matrix(ratings).T
X = np.matrix([feature(index) for index in range(0, len(lengths))])

theta = np.linalg.inv(X.T*X)*X.T*y
print("Theta_1: \n", theta[0],
"\nTheta_2: \n", theta[1],
"\nTheta_3: \n", theta[2],
"\nTheta_4: \n", theta[3])

sse = sum([x**2 for x in (y - X*theta)]) #Summing up all square errors
mse = sse / len(y) #Dividing by length to get mean square error
print("MSE = ", mse)
```

```
Theta_1:
 [[3.29014782]]
Theta_2:
 [[5.50923292e-05]]
Theta_3:
 [[0.00875072]]
Theta_4:
 [[0.05235923]]
MSE =  [[1.53677405]]
```

[139]:

```
#Using one-hot encoding
days = [date.weekday() for date in dates]
years = [date.year for date in dates]

def oneHotFeature(index):
    feature = [1]

    length = lengths[index]
    feature.append(length)

    day = dates[index].weekday()
    dayEncoded = [0]*6
    if day != 0:
        dayEncoded[day-1] = 1
    for elem in dayEncoded:
        feature.append(elem)

    year = dates[index].year
    yearEncoded = [0]*11
    if year != 2006:
```

```python
        yearEncoded[year-2007] = 1
    for elem in yearEncoded:
        feature.append(elem)

    return feature


y = np.matrix(ratings).T
X = np.matrix([oneHotFeature(index) for index in range(0, len(lengths))])

theta = np.linalg.inv(X.T*X)*X.T*y
print("Theta_1: \n", theta[0],
"\nTheta_2: \n", theta[1],
"\nTheta_3: \n", theta[2:8],
"\nTheta_4: \n", theta[8:])

sse = sum([x**2 for x in (y - X*theta)]) #Summing up all square errors
mse = sse / len(y) #Dividing by length to get mean square error
print("MSE = ", mse)
```

```
Theta_1:
 [[4.87171479]]
Theta_2:
 [[5.15709386e-05]]
Theta_3:
 [[0.04890034]
 [0.1457098 ]
 [0.1066464 ]
 [0.12616832]
 [0.03834177]
 [0.1028469 ]]
Theta_4:
 [[-1.58244783]
 [-1.70447417]
 [-1.68316056]
 [-1.67023905]
 [-1.62877001]
 [-1.19956705]
 [-1.10444816]
 [-1.09162361]
 [-1.20861354]
 [-1.23647487]
 [-1.23331225]]
MSE =  [[1.51235787]]
```

1.0.4 Task 5:

[147]:
```
# distributing "random" indices to each set
training_indices = [i for i in range(0, len(lengths), 2)]
test_indices = [i for i in range(1, len(lengths), 2)]

training_ratings = [ratings[i] for i in training_indices]
test_ratings = [ratings[i] for i in test_indices]
```

[148]:
```
#Using values directly as features
training_y = np.matrix(training_ratings).T
test_y = np.matrix(test_ratings).T
training_X = np.matrix([feature(index) for index in training_indices])
trained_theta = np.linalg.inv(training_X.T*training_X)*training_X.T*training_y

sse = sum([x**2 for x in (training_y - training_X*trained_theta)]) #Summing up␣
↪all square errors
mse = sse / len(test_y) #Dividing by length to get mean square error
print("MSE(training data) = ", mse)
sse = sum([x**2 for x in (test_y - training_X*trained_theta)]) #Summing up all␣
↪square errors
mse = sse / len(test_y) #Dividing by length to get mean square error
print("MSE(test data) = ", mse)
```

```
MSE(training data) =  [[1.54991753]]
MSE(test data) =  [[1.52654798]]
```

[149]:
```
#Using one-hot encoding
training_y = np.matrix(training_ratings).T
test_y = np.matrix(test_ratings).T
training_X = np.matrix([oneHotFeature(index) for index in training_indices])
trained_theta = np.linalg.inv(training_X.T*training_X)*training_X.T*training_y

sse = sum([x**2 for x in (training_y - training_X*trained_theta)]) #Summing up␣
↪all square errors
mse = sse / len(test_y) #Dividing by length to get mean square error
print("MSE(training data) = ", mse)
sse = sum([x**2 for x in (test_y - training_X*trained_theta)]) #Summing up all␣
↪square errors
mse = sse / len(test_y) #Dividing by length to get mean square error
print("MSE(test data) = ", mse)
```

```
MSE(training data) =  [[1.52069864]]
MSE(test data) =  [[1.51164629]]
```

1.0.5 Task 6:

(6) $$MAE = \sum_{i=1}^{10000} |y_i - \theta_0|$$

$$\frac{\partial}{\partial \theta_0} MAE = -\underbrace{\sum_{i=1}^{10000} \delta(y_i > \theta_0)}_{y_i > \theta_0} + \underbrace{\sum_{i=1}^{10000} \delta(y_i < \theta_0)}_{y_i < \theta_0} = 0$$

If $y_i > \theta_0$ and $y_i < \theta_0$ happens exactly the same amount of times, the two terms above will cancel each other out, and $\frac{\partial}{\partial \theta_0} MAE$ will be 0.

This has to be a global minima, since it does not makes sense that increasing or decreasing θ_0 to the extreme will lead to a smaller MAE, which it would have done if it was a global maxima or a saddle point.

The only way to make the two terms cancel each other out, i.e. that $y_i > \theta_0$ and $y_i < \theta_0$ occurs the same amount of times, is if θ_0 has the value of the median of y.

2 Tasks — Classification (week 2):

2.0.1 Task 7:

[150]:
```
import pandas as pd
import numpy as np
import sklearn
from sklearn import linear_model
from urllib.request import urlopen

def readFromURL(URL):
    for l in urlopen(URL):
        yield eval(l)

data = list(readFromURL("https://cseweb.ucsd.edu/classes/fa21/cse258-b/data/
 ↪beer_50000.json"))
reviews = pd.DataFrame(data)
```

[152]:
```
lengths = [len(review) for review in reviews["review/text"]]
X = np.matrix([[1,length] for length in lengths])
y = [rating >= 4 for rating in reviews["review/overall"]]

model = sklearn.linear_model.LogisticRegression(class_weight='balanced')
model.fit(X,y)
predictions = model.predict(X)

truePos = sum([(predictions[i] and y[i]) for i in range(0, len(y))])
fakePos = sum([(predictions[i] and not y[i]) for i in range(0, len(y))])
trueNeg = sum([(not predictions[i] and not y[i]) for i in range(0, len(y))])
fakeNeg = sum([(not predictions[i] and y[i]) for i in range(0, len(y))])

FPR = fakePos/(fakePos+trueNeg)
FNR = fakeNeg/(fakeNeg+truePos)

BER = 0.5*(FPR + FNR)

print("True Positive = ", truePos)
print("Fake Positive = ", fakePos)
print("True Negative = ", trueNeg)
print("Fake Negative = ", fakeNeg)
print("Balanced Error Rates = ", BER)
```

```
True Positive =  14201
Fake Positive =  5885
True Negative =  10503
Fake Negative =  19411
Balanced Error Rates =  0.46830315259572763
```

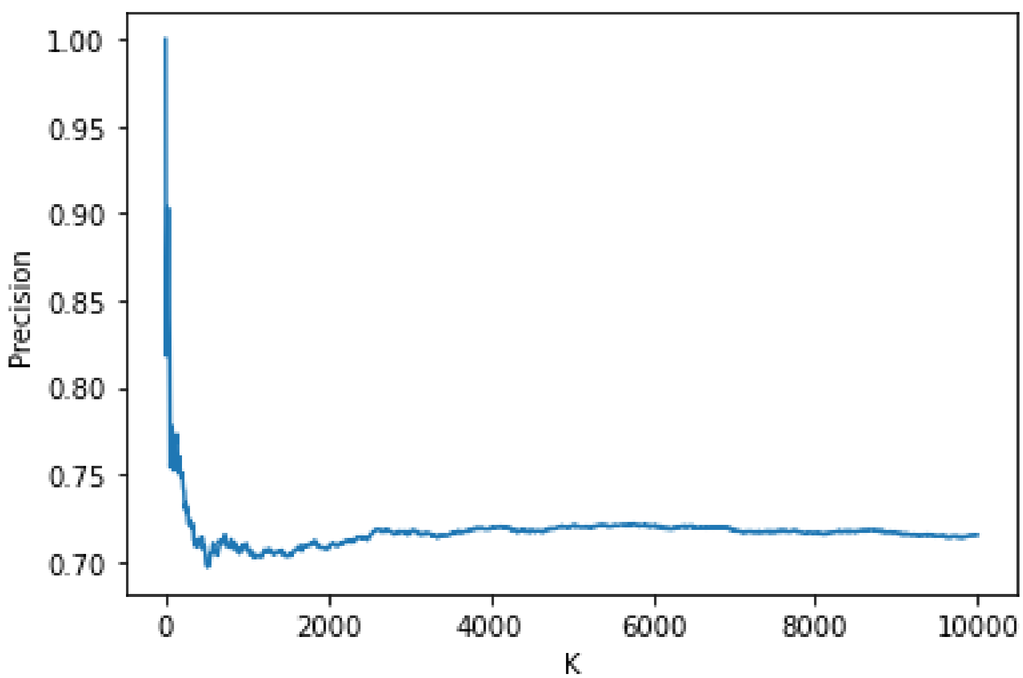
2.0.2 Task 8:

[154]:
```
import matplotlib.pyplot as plt

confidences = model.decision_function(X)
sortedByConfidence = list(zip(confidences, y))
sortedByConfidence.sort(reverse=True)

K = [K for K in range(1, 10000+1, 10)]
precisionsK = []
for k in K:
    retrievedLabels = [x[1] for x in sortedByConfidence[:k]]
    precisionK = sum(retrievedLabels)/len(retrievedLabels)
    precisionsK.append(precisionK)

plt.plot(K, precisionsK)
plt.xlabel("K")
plt.ylabel("Precision")
plt.show()
print(f"precision@K for k=1: {precisionsK[0]}, k=100: {precisionsK[100//10-1]},␣
↪k=10000: {precisionsK[10000//10-1]}")
```




```
precision@K for k=1: 1.0, k=100: 0.7692307692307693, k=10000: 0.7152437193474127
```

2.0.3 Task 9:

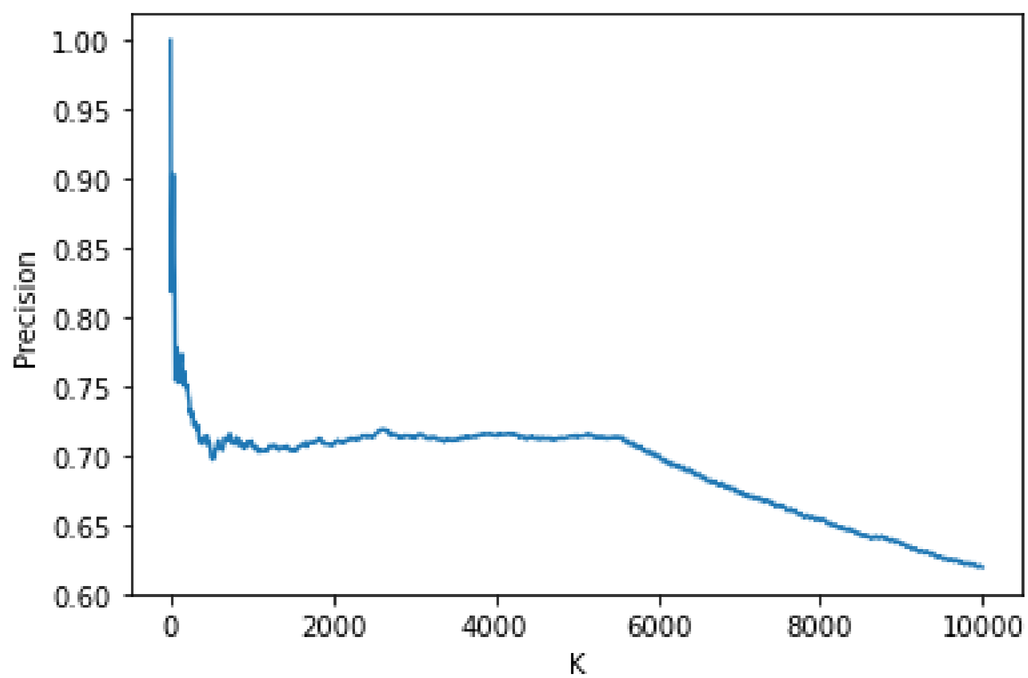
[155]:
```
import matplotlib.pyplot as plt

confidences = model.decision_function(X)
sortedByConfidence = list(zip(abs(confidences), y, predictions))
sortedByConfidence.sort(reverse=True)

K = [K for K in range(1, 10000+1, 10)]
precisionsK = []
for k in K:
    retrievedLabels = [[x[1], x[2]] for x in sortedByConfidence[:k]]
    precisionK = 0
    for elem in retrievedLabels:
        precisionK += (int)(elem[0]==elem[1])
    precisionK = precisionK/k
    precisionsK.append(precisionK)

plt.plot(K, precisionsK)
plt.xlabel("K")
plt.ylabel("Precision")
plt.show()

print(f"precision@K for k=1: {precisionsK[0]}, k=100: {precisionsK[100//10-1]},␣
↪k=10000: {precisionsK[10000//10-1]}")
```



```
precision@K for k=1: 1.0, k=100: 0.7692307692307693, k=10000: 0.6188569712741467
```