Homework4

November 30, 2020

#Question 1

#Part 1

The problem explicitly states NOT to find the pdf of the posterior, but I was unable to come up with a cleverer way to do this on my own. Therefore, here is a derivation that I adapted from these sources: http://krasserm.github.io/2018/03/19/gaussian-processes/,https://see.stanford.edu/materials/aimlcs229/cs229-gp.pdf,https://www.csie.ntu.edu.tw/~cjlin/mlgrou

Let $A = -K(X^*, X)K(X, X)^{-1}$. Then, let $z = y^* + Ay$.

Now, we can show the mean as the following:

$E(y^*|X^*, X, y) = E(z - Ay|X^*, X, y) = E(z|X^*, X, y) - E(A|X^*, X, y) = -Ay = -K(X^*, X)K(X, X)^{-1}y$

And the variance as:

$Var(y^*|X^*, X, y)$

$= Var(z|X^*, X, y) + Var(Ay|X^*, X, y) - Cov(z, -Ay|X^*, X, y) - Cov(-Ay, z|X^*, X, y)$

$= Var(y^* + Ay|X^*, X)$

$= Var(y^*|X^*, X) + Var(Ay, X^*, X) - Cov(y^*, Ay|X^*, X) - Cov(Ay, y^*|X^*, X)$

$= K(X^*, X^*) - K(X^*, X)K(X, X)^{-1}K(X, X^*)$

#Part 2

[9]:
```
import numpy as np
from numpy.linalg import inv
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn import preprocessing
from sklearn import utils
import numpy
import math
```

For this question, I used the following resources: http://krasserm.github.io/2018/03/19/gaussian-processes/, http://www.gaussianprocess.org/gpml/chapters/RW5.pdf, and https://www.cs.toronto.edu/~hinton/csc2515/notes/gp_slides_fall08.pdf

[4]:
```python
def RBFKernel(X1,X2,sigma,h): #same RBF kernel function as in the last homework
    K = np.zeros((X1.shape[0],X2.shape[0]))
    for i in range(X1.shape[0]):
        for j in range(X2.shape[0]):
            K[i,j] = sigma*np.exp(((-np.linalg.norm(X1[i]-X2[j])**2)/2*(h**2)))
    return K

def GPRegression(XTrain, yTrain, XTest, gamma, sigma, h):
    num = np.zeros(XTrain.shape[0])
    y_mean = np.mean(yTrain)
    y = yTrain - y_mean
    K = RBFKernel(XTrain, XTrain, sigma,h) #can repeat this for test data as␣
 ↪well
    L_ = gamma*np.eye(len(num)) #identity matrix
    L_n = K + (np.tril(L_,k=0))
    L = np.linalg.cholesky(L_n) #computing the cholesky decomposition
    alpha = np.linalg.solve(L.T,np.linalg.solve(L,y)) #solves the matrix eqn␣
 ↪for the cholesky decomp and the target
    GPMean = y_mean + ((K.T)*alpha) #final regression mean

    var = np.linalg.solve(L,K) #solves matrix eqn for the cholesky decomp and␣
 ↪the kernel
    GPVariance = K - ((var.T)*var) #final regression variance


    return GPMean, GPVariance

def LogMarginalLikelihood(XTrain,yTrain,gamma,sigma,h):
    n = np.zeros(XTrain.shape[0],dtype=int)

    ym = np.mean(yTrain)
    #print(ym)
    y = yTrain - ym
    K = RBFKernel(XTrain,XTrain,sigma,h)

    #same calculation as before
    L_ = gamma*np.eye(len(n))
    L_n = K + (np.tril(L_,k=0))
    L = np.linalg.cholesky(L_n)


    alpha = np.linalg.solve(L.T,np.linalg.solve(L,y))
    #use equation from second resource
```

```
    logml = np.mean((-1/2*y.T*alpha)-(sum(np.log(np.diag(L))) - ((n/2)*np.
↪log(2*np.pi))))
    return logml,sigma,h

def HyperParameters(XTrain,yTrain,hs,sigmas):
    gamma = 0.01*(np.std(yTrain))

    margs_list = []
    #couldnt figure out how to do this with gridsearch, so I'm doing it␣
↪manually with a for loop
    for i in sigmas:
        for j in hs:
            lm,s,h = LogMarginalLikelihood(XTrain,yTrain,gamma,i,j)
            margs_list.append(lm)
            if np.min(margs_list) == lm:
                new_min = [lm,s,h]
            else:
                pass
    h = new_min[2]
    sigma = new_min[1]
    return gamma,h,sigma


data = pd.read_excel('/Users/lvbenson/Research_Projects/MachineLearning/
↪Assignment4/Concrete_Data.xls')
X = data.iloc[:,:-1] # Features
y = data.iloc[:,-1] # Target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=2)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


#logspace(-1,1,10)'?norm(std(XTrain))
hs_ = (np.logspace(-1,1,num=10).T)*np.linalg.norm(np.std(X_train))

sigmas_ = (np.logspace(-1,1,num=10).T)*(np.std(y_train))
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)

gamma = 0.01*(np.std(y_train))
g,h,s = HyperParameters(X_train,y_train,hs_,sigmas_)
M,V = GPRegression(X_train, y_train, X_test, g, s, h)
print(M,'mean')
print(V,'variance')
```

```
print(np.mean(y_train), 'naive mean')
```

```
[[   26.22852155    36.09550331    36.09550331 …    36.09550331
     36.09550331    36.09550331]
 [   36.09550331     2.85790058    36.09550331 …    36.09550331
     36.09550331    36.09550331]
 [   36.09550331    36.09550331    29.7968497  …    36.09550331
     36.09550331    36.09550331]
 …
 [   36.09550331    36.09550331    36.09550331 …    23.04316675
     36.09550331    36.09550331]
 [   36.09550331    36.09550331    36.09550331 …    36.09550331
     41.59678782    36.09550331]
 [   36.09550331    36.09550331    36.09550331 …    36.09550331
     36.09550331 -1391.64773799]] mean
[[1.68710162e-001 9.01707893e-312 0.00000000e+000 … 3.13691184e-109
  5.68545337e-220 6.42874404e-198]
 [9.01707893e-312 1.68710162e-001 0.00000000e+000 … 0.00000000e+000
  0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 1.68710162e-001 … 0.00000000e+000
  0.00000000e+000 0.00000000e+000]
 …
 [3.13691184e-109 0.00000000e+000 0.00000000e+000 … 1.25712962e+001
  1.25271100e-029 6.17455158e-045]
 [5.68545337e-220 0.00000000e+000 0.00000000e+000 … 1.25271100e-029
  1.76743542e-001 4.28887605e-019]
 [6.42874404e-198 0.00000000e+000 0.00000000e+000 … 6.17455158e-045
  4.28887605e-019 1.68794559e+002]] variance
36.09550330722694 naive mean
```

#Question 2

#Part 1

[95]:
```
data = pd.DataFrame({
        "age": [24,53,23,25,32,52,22,43,52,48],
        "salary":␣
 ↪[40000,52000,25000,77000,48000,110000,38000,44000,27000,65000],
        "degree": [1,0,0,1,1,1,1,0,0,1],
    })
#lab_enc = preprocessing.LabelEncoder()
X = data.iloc[:,:-1] # Features
y = data.iloc[:,-1] # Target
#X = lab_enc.fit_transform(X_)
#y = lab_enc.fit_transform(y_)
```

[96]:
```
#Building the decision tree
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
clf = DecisionTreeClassifier(criterion="entropy")
clf = clf.fit(X,y)
y_pred = clf.predict(X)
print("Accuracy:",metrics.accuracy_score(y, y_pred))
```

```
Accuracy: 1.0
```

[102]:
```
from sklearn.tree.export import export_text
show_tree = export_text(clf, feature_names=list(X))
print(show_tree)
```

```
|--- salary <= 32500.00
|   |--- class: 0
|--- salary >  32500.00
|   |--- age <= 37.50
|   |   |--- class: 1
|   |--- age >  37.50
|   |   |--- salary <= 58500.00
|   |   |   |--- class: 0
|   |   |--- salary >  58500.00
|   |   |   |--- class: 1
```

[104]:
```
info_gain = pd.DataFrame({'feature':X.columns,'info':clf.feature_importances_,})
info_gain = info_gain.sort_values('info',ascending=False)
print(info_gain)
```

```
  feature      info
1  salary  0.743527
0     age  0.256473
```

The depth of this tree is 3. The information gain at each split is: 0.743527, 0.256473

#Part b

[105]:
```
from sklearn.tree import DecisionTreeRegressor

mult_tree = DecisionTreeRegressor()

mult_tree.fit(X,y)
mult_tree_rules = export_text(mult_tree, feature_names=list(X))
print(mult_tree_rules)
```

```
|--- salary <= 32500.00
|   |--- value: [0.00]
|--- salary >  32500.00
|   |--- age <= 52.50
|   |   |--- age <= 37.50
|   |   |   |--- value: [1.00]
```

```
|   |   |--- age >  37.50
|   |   |   |--- age <= 45.50
|   |   |   |   |--- value: [0.00]
|   |   |   |--- age >  45.50
|   |   |   |   |--- value: [1.00]
|   |--- age >  52.50
|   |   |--- value: [0.00]
```

[106]:
```
mult_info_gain = pd.DataFrame({'feature':X.columns,'info':mult_tree.
 ↪feature_importances_,})
mult_info_gain = mult_info_gain.sort_values('info',ascending=False)
print(info_gain)
```

```
  feature      info
1  salary  0.743527
0     age  0.256473
```

#Part c

A multivariate decision tree would be a poor choice with a small dataset, like the one we have used for this problem, given the increased potential for overfitting.

#Problem 3

#Part 1

5 * ($4^2$3) * (3$\mathit{3}$3) = 20,480

#Part 2

No idea

#Part 3

No idea

#Part 4

Because the data set is so small, in this case, avoiding overfitting and high variance is the most important thing. Having fewer numbers of splits in this case would help these issues.