Report for Sheet 3

Lab Course Machine Learning and Data Analysis

Hiromasa Okada

June 16, 2017

Implementation comments

In this exercise I implemented Cross-validation function

```
def cv(X, y, method, parameters, nfolds, nrepetitions, loss function)
```

and Kernel Ridge Regression class

```
class krr(kernel, kernelparameter, regularization)
```

The cross validation function will find the best parameter combination for Kernel Ridge Regression by computing loss of the prediction and find the parameters which make the loss to be near the mean loss. The reason why the loss shuld be mean instead of minimum is that if we choose the minimum loss, the overfitting will occure.

Kernel Ridge Regression class saves kernel, kernelparameter, regularization as object when this class is constructed. This class has following three functions

```
def getkernel(self, X, Y=None):
def fit(self, X, y):
def predict(self, X):
```

X is dataset and y is true values. The getkernel() will get a kernel. It can make a kernel from one dataset and from two dataset. The default is with one dataset. When fit() is called, dataset X is saved as an object Xfit and the coefficients α is saved as an object alpha. The function predict will estimate Y. This function first gets a kernel from a new kernel from Xfit and new dataset X then estimate Y by using the saved alpha.

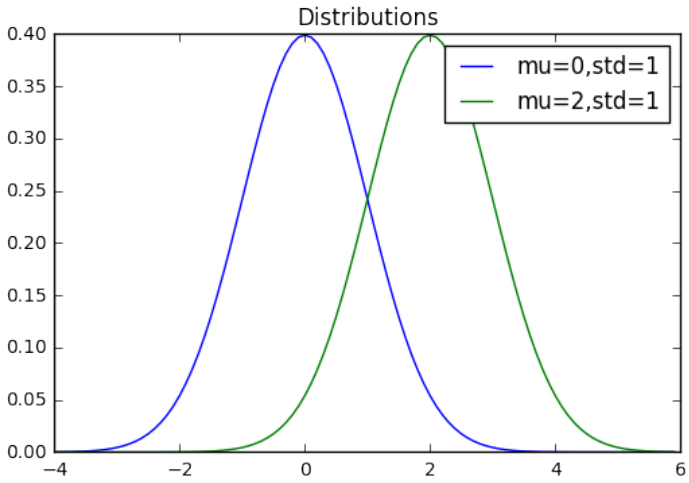
Assignment 3

We know the following distributions

$$p(x|y=-1) \sim N(\mu=0;\sigma=1)$$
$$p(x|y=+1) \sim N(\mu=2;\sigma=1)$$

Then we will plot the graph of this by following way.

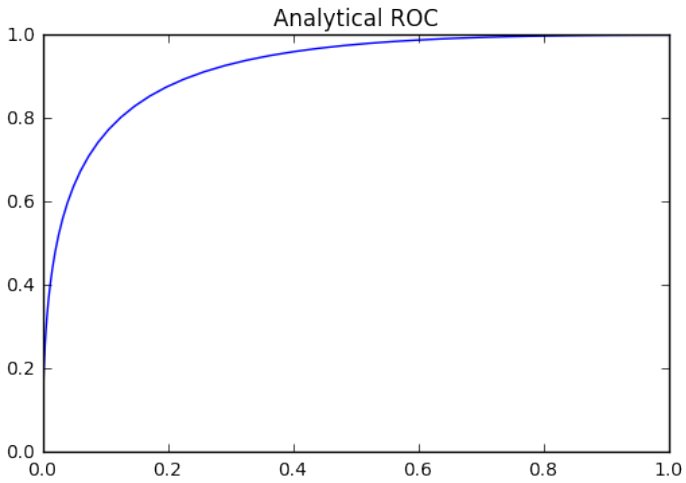
```
def nomdis(x,mu, sigma):
    return np.exp(-((x-mu)**2)/(2*sigma**2))/np.sqrt(2*np.pi*sigma**2)

pp1 = nomdis(np.arange(-4,6,0.1),0,1)
pp2 = nomdis(np.arange(-4,6,0.1),2,1)
plt.plot(np.arange(-4,6,0.1),nomdis(np.arange(-4,6,0.1),0,1),label='mu=0,std=1')
plt.plot(np.arange(-4,6,0.1),nomdis(np.arange(-4,6,0.1),2,1),label='mu=2,std=1')
plt.legend()
plt.title("Distributions")
```



And we can calculate the ROC curve analytically by using the probability density function of gaussian. I first had to decide the enough long interval of random variables. In case of the upper distribution I chose np.arange(-4,6,0.1). From -4 to 6 the threshold is slided with step interval 0.1. At each

step the right side of the area of 1st distribution and 2nd distribution are calicurated. The right side of the area of 1st distribution divided by whole area of 1st distribution is false positive rate and The right side of the area of 2nd distribution divided by whole area of 2nd distribution is true positive rate.

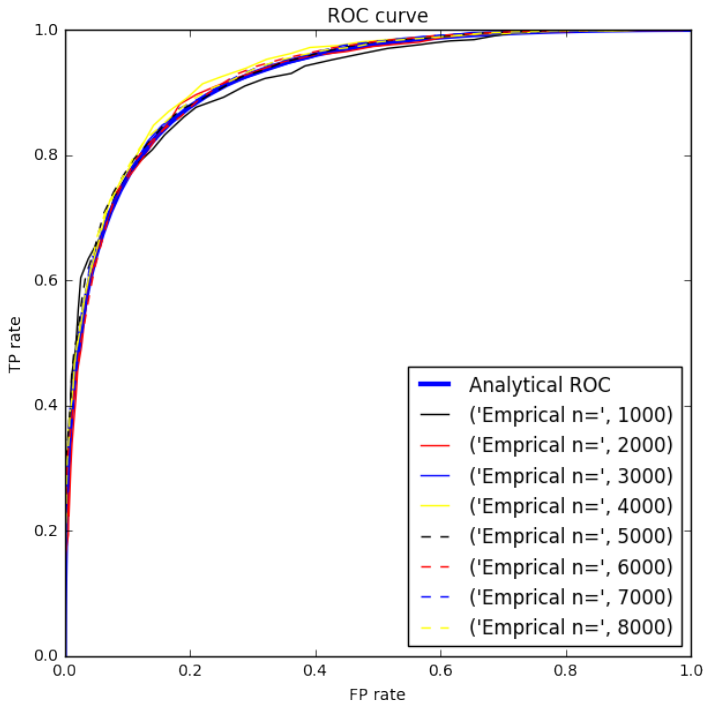
```
s1 = pp1.sum()
s2 = pp2.sum()
c1 = np.cumsum(pp1)/np.cumsum(pp1)[-1]
c2 = np.cumsum(pp2)/np.cumsum(pp2)[-1]
fp = 1-c1
tp =1-c2
plt.plot(fp,tp)
plt.title("Analytical ROC")
```



After that the empirical ROC curve is calicurated with n datasets by the following algorithum. And ploted with analytical ROC curve.

```
def ROCcul(P):
    f= np.linspace(-4,6,21)#np.arange(11)-4
    ptrue = np.array(P[:,1])
    ttp = ptrue.sum()
    tfp = len(ptrue)-ttp
    roc = np.zeros((2,len(f)))
    for i in range(len(f)):
        pred =np.zeros(len(P))
```

```
        pred[np.where(P[:,0]>f[i])]=1
        roc[1][i] = ptrue[np.where(pred==1)].sum()/ttp
        roc[0][i] = pred[np.where(ptrue==0)].sum()/tfp
    ROC=roc[:,np.argsort(roc[0])]
    return ROC

N = np.arange(1000,9000,1000)
plt.figure(figsize=(7,7))
plt.plot(fp,tp,label="Analytical ROC",linewidth=3.0)
cl = ['black','red','blue','yellow','black','red','blue','yellow']
line = ['solid','solid','solid','solid','dashed','dashed','dashed','dashed']
ite = 0
for n in N:
    p1 = np.random.randn(n/2)
    p2 = np.random.randn(n/2) +2
    P1 = np.append(p1.reshape(1,len(p1)),np.zeros((1,len(p1))),0)
    P2 = np.append(p2.reshape(1,len(p2)),np.ones((1,len(p2))),0)
    P = np.append(P1,P2,1)
    P =P.T[np.argsort(P[0])]
    ptrue = np.array(P[:,1])
    sroc = ROCcul(P)
    plt.plot(sroc[0],sroc[1], label=("Emprical n=",n),color=cl[ite],linestyle=line[
    ite = ite+1
plt.legend(loc="lower right")
plt.xlabel("FP rate")
plt.ylabel("TP rate")
plt.title("ROC curve")
```



The analytical ROC curve is much smoother than all empirical ROC curves. And it lies on middle of empirical ROC curves. In case of empirical ROC curve The more sample data points are, the clother the ROC curve seems to lie on the upper left side. But this difference is not much remarkable.

Assignment 4

Results

```
{'banana': {'cvloss': 0.13734693877551021,
  'kernel': 'gaussian',
  'kernelparameter': 4.6415888336127775,
  'regularization': 0.0001,
  'ypred': array([[ 0.77109269],
         [-0.2448248 ],
         [-0.53843675],
         ...,
         [ 1.04244162],
```

```
        [-0.60053262],
        [-0.16481672]])},
'diabetis': {'cvloss': 0.24666666666666667,
 'kernel': 'linear',
 'kernelparameter': 0,
 'regularization': 0.0001,
 'ypred': array([[ 0.2926619 ],
        [ 0.06084429],
        [ 0.8399555 ],
        ...,
        [-1.019224  ],
        [-0.42906231],
        [ 0.02838715]])},
'flare-solar': {'cvloss': 0.3075,
 'kernel': 'linear',
 'kernelparameter': 0,
 'regularization': 0.0001,
 'ypred': array([[-0.24184525],
        [-0.24184525],
        [-0.26401811],
        ...,
        [-0.28619095],
        [-0.24184525],
        [-0.24184525]])},
'image': {'cvloss': 0.1792079207920792,
 'kernel': 'linear',
 'kernelparameter': 0,
 'regularization': 0.0001,
 'ypred': array([[ 0.68771487],
        [-0.6650025 ],
        [ 0.25757335],
        ...,
        [-0.53029856],
        [-0.05421134],
        [ 0.06081382]])},
'ringnorm': {'cvloss': 0.24357142857142858,
```

```
 'kernel': 'linear',
 'kernelparameter': 0,
 'regularization': 0.0001,
 'ypred': array([[-0.26620809],
        [-0.55446703],
        [-0.29458036],
        ...,
        [-0.2445433 ],
        [-0.14951235],
        [-0.03140741]])}}
```

ROC and AUC

To calicurate ROC, AUC and loss I defined a following function. This function returns only when plot = True. Therefore it works only as a loss function during the cross varidation but if we caricurate the loss of the test, we can plot a ROC curve and print AUC value.

```
def roc_fun(y_true, y_pred, plot = False):
    assert(len(y_true) == len(y_pred))
    bins=100
    pred = np.array(y_pred[:,0])
    true = np.array(y_true[:,0])
    true[np.where(true==-1)]=0
    n=len(pred)
    thres = np.linspace(np.min(pred),np.max(pred),bins).reshape(bins,1)*np.ones((1
    predbr = pred.reshape(1,n)*np.ones((bins ,1))
    prebl=(predbr>thres).astype(np.int64)
    comp = (prebl==true).astype(np.int64)
    tpr = (prebl*comp).sum(1)/true.sum()
    fpr = (((comp-prebl))==(-np.ones(comp.shape))).sum(1)/(n-true.sum())
    idx=np.argsort(fpr)
    roc = np.append(fpr.reshape(1,len(fpr)),tpr.reshape(1,len(tpr)),0)
    D = np.linalg.norm(roc - np.array([[0],[1]]),axis=0)
    minidx=np.argmin(D)
    loss=(n-comp[minidx].sum())/n
```

```
    ROC = roc[:,idx]

    if(plot==True):
        fpdif = ROC[0,1:]-ROC[0,:-1]
        lower = fpdif*ROC[1,:-1]
        upper = fpdif*ROC[1,1:]
        AUC = (lower.sum()+upper.sum())/2
        print("AUC = ",AUC)
        plt.plot(ROC[0],ROC[1])
        plt.title("ROC curve")
        plt.xlabel("FP")
        plt.ylabel("TP")
    return loss
```

The plots of best results of ROC and AUC are following.

```
ROC curve of the classification of Banana
AUC = 0.927940824614
```

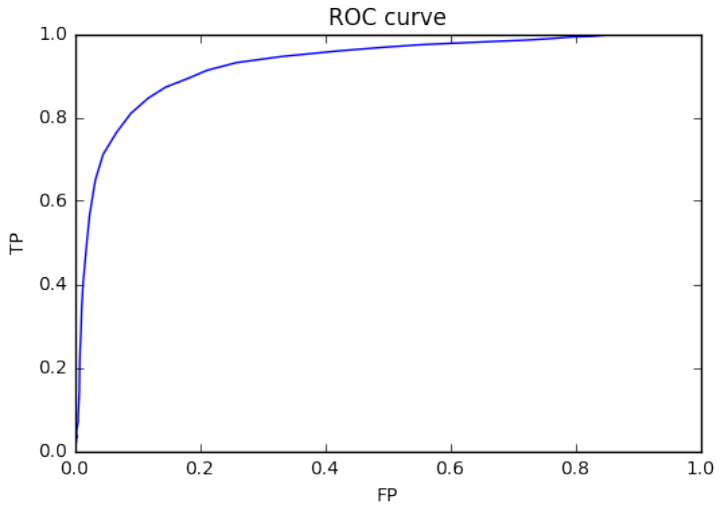


Figure 1: ROC curve of the Banana

```
ROC curve of the classification of diabetis
AUC =0.846986417657
```

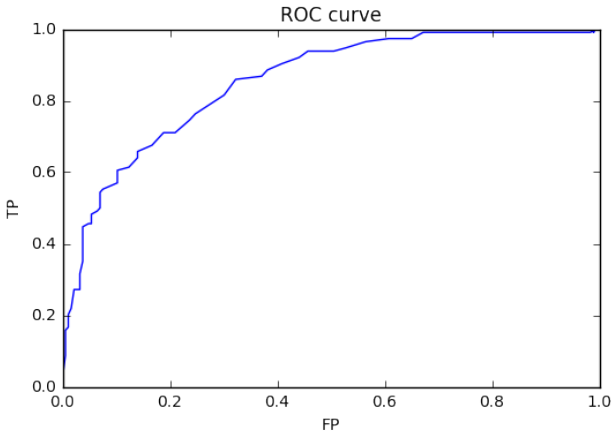


Figure 2: ROC curve of the diabetis

```
ROC curve of the classification of flare-solar
AUC = 0.846986417657
```

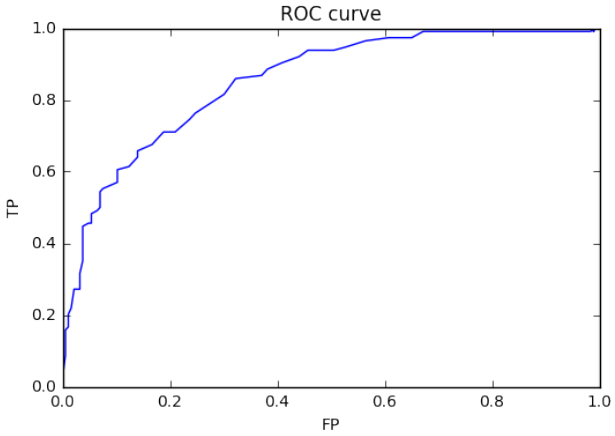


Figure 3: ROC curve of the flare-solar

```
ROC curve of the classification of image
AUC = 0.846986417657
```

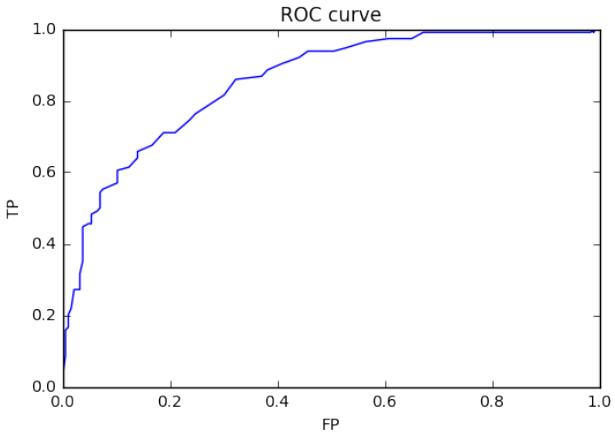


Figure 4: ROC curve of the of image

```
ROC curve of the classification of ringnorm
AUC = 0.846986417657
```

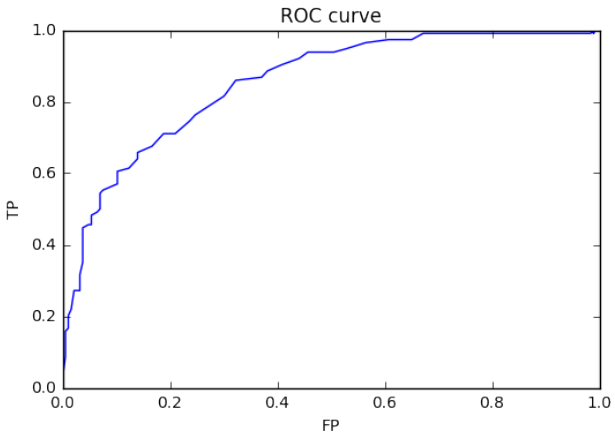


Figure 5: ROC curve of the ringnorm