Problem 1

HW3
5241
hl3115
Haiqi Li

1. $\hat{\beta}_0 = 0$

$$\min_{\beta_1\beta_2} \sum_{i=1}^{2} (y_i - \beta_1 x_{i1} - \beta_2 x_{i2})^2 + (\beta_1^2 + \beta_2^2)\lambda$$

$$= \min_{\beta_1\beta_2} (y_1 - \beta_1 x_{11} - \beta_2 x_{12})^2 + (y_2 - \beta_1 x_{21} - \beta_2 x_{22})^2 + (\beta_1^2 + \beta_2^2)\lambda$$

2. suppose the objective fuction is J

then $\frac{\partial J}{\partial \beta_1} = 2(y_1 - \beta_1 x_{11} - \beta_2 x_{12})(-x_{11}) + 2(y_2 - \beta_1 x_{21} - \beta_2 x_{22})(-x_{21}) + 2\beta_1\lambda = 0$

$$\Rightarrow \hat{\beta}_1 = \frac{x_{11}y_1 + x_{21}y_2 - x_{11}x_{12}\beta_2 - x_{21}x_{22}\beta_2}{\lambda + x_{11}^2 + x_{21}^2} = \frac{x_1 y_1 + x_2 y_2 - x_1^2\beta_2 - x_2^2\beta_2}{\lambda + x_1^2 + x_2^2}$$

(suppose $x_{11} = x_{12} = x_1$, $x_{21} = x_{22} = x_2$)

likely $\frac{\partial J}{\partial \beta_2} = 0 \Rightarrow \hat{\beta}_2 = \frac{x_1 y_1 + x_2 y_2 - x_1^2\beta_1 - x_2^2\beta_1}{\lambda + x_1^2 + x_2^2}$

So β_1 and β_2 are symmetric $\hat{\beta}_1 = \hat{\beta}_2$

3. $\min_{\beta_1\beta_2} (y_1 - \beta_1 x_{11} - \beta_2 x_{12})^2 + (y_2 - \beta_1 x_{21} - \beta_2 x_{22})^2 + (|\beta_1| + |\beta_2|)\lambda$ ($|\beta_1|$ is absolute value of β_1)

4. like 2. $\frac{\partial J}{\partial \beta_1} = 2(y_1 - \beta_1 x_1 - \beta_2 x_1)(-x_1) + 2(y_2 - \beta_1 x_2 - \beta_2 x_2)(-x_2) + \lambda\,\mathrm{sgn}(\beta_1) = 0$

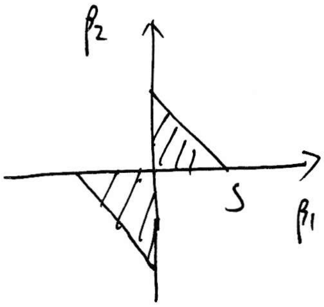
$\frac{\partial J}{\partial \beta_2} = 2(y_1 - \beta_1 x_1 - \beta_2 x_1)(-x_1) + 2(y_2 - \beta_1 x_2 - \beta_2 x_2)(-x_2) + \lambda\,\mathrm{sgn}(\beta_2) = 0$

so $\mathrm{sgn}(\beta_1) = \mathrm{sgn}(\beta_2)$

change J to $= 2(y_1 - \beta_1 x_1 - \beta_2 x_1)^2$ s.t. $|\beta_1| + |\beta_2| < s$ ($x_1 = -x_2$, $y_1 = -y_2$)

$\hat{\beta}_1 + \hat{\beta}_2 = \frac{y_1}{x_1}$

So there exists many possible solutions.



Problem 2.

1. $\hat{g}_2$ with penalty $\lambda \int g^{(4)}(x)^2 dx$ while $\hat{g}_1$ with penalty $\lambda \int g^{(3)}(x)^2 dx$

Apparently, $\hat{g}_2$ is of higher flexibility, so its RSS is likely to be small on training set.

2. It's depend on the distribution of true y_i

~~If y_i is of high noise,~~

If y_i is linear, then high flexibility mean high variance, which would cause high RSS because of overfitting. In this case, $\hat{g}_1$ is of lower RSS.

But if y_i is squiggly, then higher flexibility may be better estimation. In this case, $\hat{g}_2$ is of lower RSS.

We also have to consider about noise of model. Usually high noise would make model with higher flexibility to have higher RSS. So in high variance case, $\hat{g}_1$ may perform better in RSS. But this may be ~~trade off~~ affected by the linearity of model. So on the whole the answer is not very clear.

3. If $\lambda \to 0$ then $\hat{g}_1 \simeq \hat{g}_2$ since penalty can be neglected.

5214 HW3 Haiqi Li hl3115

March 18, 2018

```
In [1]: %config IPCompleter.greedy=True
        %config IPCompleter.debug=True
        %matplotlib inline
        import pandas as pd
        import numpy as np
        from sklearn import svm
        import math
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import cross_val_score
        import matplotlib.pyplot as plt
        from tqdm import tqdm,tqdm_notebook as tqdm
```

```
In [2]: data5=pd.read_table(r'.\train.5.txt',sep=',',header=None)
        data6=pd.read_table(r'.\train.6.txt',sep=',',header=None)
        n5=data5.shape[0]
        n6=data6.shape[0]
        nwhole=n5+n6
        _=np.array(["5","6"])
        label=np.repeat(_,[n5,n6],axis=0)
        data=pd.concat([data5,data6])
        data_train,data_test,label_train,label_test=train_test_split(data,label,test_size=0.2,r
        #Load Data and divide into train set and test set.
```

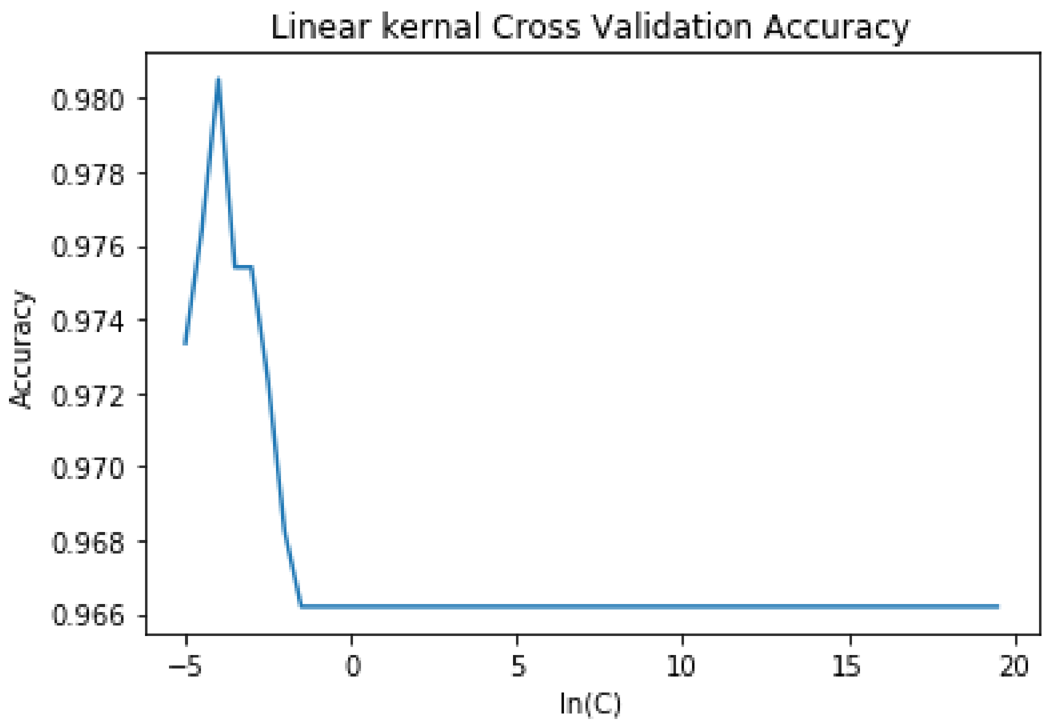
1 1.

1.1 (a)

```
In [3]: C=[math.exp(x / 2.0) for x in range(-10, 40)]
        C_len=len(C)
        linear_kernal_accuracy=[]
        for i in range(C_len):
            linear=svm.SVC(kernel='linear',C=C[i])
            scores=cross_val_score(linear,data_train,label_train,cv=5)
            score=np.mean(scores)
            linear_kernal_accuracy.append(score)
        #record different c in linear kernal model
        iter=np.arange(C_len)
```

```
plt.plot((iter-10)/2,linear_kernal_accuracy)
plt.xlabel("ln(C)")
plt.ylabel("Accuracy")
plt.title("Linear kernal Cross Validation Accuracy")
print("The best ln(C) is around:%f"%((np.argmax(linear_kernal_accuracy)-10)/2))
```

```
The best ln(C) is around:-4.000000
```



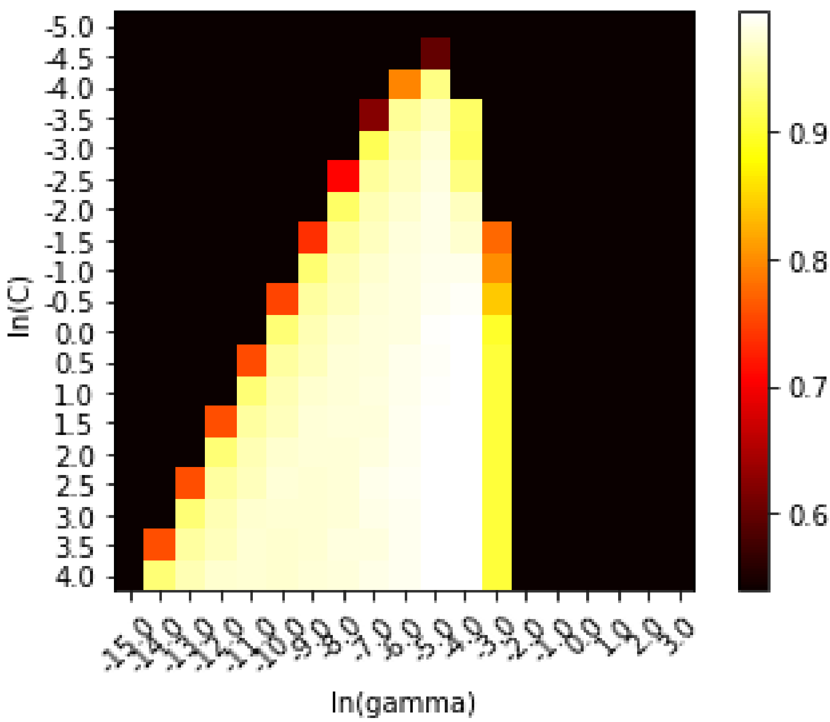
1.2 (b)

```
In [4]: C=[math.exp(x / 2.0) for x in range(-10,9)]
        C_len=len(C)
        gamma=[math.exp(x) for x in range(-15, 4)]
        gamma_len=len(gamma)
        rbf_kernal_accuracy=np.zeros((C_len,gamma_len))
        pbar=tqdm(total=C_len*gamma_len)
        for i in range(C_len):
            for j in range(gamma_len):
                pbar.update(1)
                rbf=svm.SVC(kernel='rbf',C=C[i],gamma=gamma[j])#use previous C
                scores=cross_val_score(rbf,data_train,label_train,cv=5)
                score=np.mean(scores)
                rbf_kernal_accuracy[i][j]=score
        # the exception can be neglected
```

A Jupyter Widget

In [5]:
```
plt.imshow(rbf_kernal_accuracy,cmap="hot",interpolation='nearest')
plt.colorbar()
plt.xlabel("ln(gamma)")
plt.ylabel("ln(C)")
plt.xticks(np.arange(gamma_len),np.log(gamma), rotation=45)
_=plt.yticks(np.arange(C_len), np.log(C))
inx=np.unravel_index(rbf_kernal_accuracy.argmax(),rbf_kernal_accuracy.shape)
solution_gamma,solution_C=inx
solution_C=(solution_C-10)/2
solution_gamma=solution_gamma-15
print("Best solution of ln(gamma) and ln(C) are:")
print((solution_gamma,solution_C))
```

```
Best solution of ln(gamma) and ln(C) are:
(-4, 0.5)
```



2 2

In [6]:
```
linear_SVM=svm.SVC(kernel='linear',C=math.exp(-4))
linear_SVM.fit(data_train,label_train)
```

```python
linear_predict=linear_SVM.predict(data_test)
print("Misclassification rate of linear kernel %f"%(1-np.mean(linear_predict==label_tes

rbf_SVM=svm.SVC(kernel='rbf',C=math.exp(0.5),gamma=math.exp(-4))
rbf_SVM.fit(data_train,label_train)
rbf_predict=rbf_SVM.predict(data_test)
print("Misclassification rate of rbf kernel %f"%(1-np.mean(rbf_predict==label_test)))
```

```
Misclassification rate of linear kernel 0.012295
Misclassification rate of rbf kernel 0.004098
```

Aparently,the non-linear SVM performs better than linear SVM in this case.We should try non-linear kernel SVM.