Stat 542 HW9

Harris Nisar (nisar2)

April 13, 2022

1 About HW9

In this HW, we use two examples to demonstrate how the RKHS and Representer Theorem are used in practice. In both questions, we will use the radial basis kernel, defined as

$$K(\mathbf{x}, \mathbf{z}) = e^{-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}}.$$

You are not required to tune the parameter σ. The information will be given.

2 Question 1 [45 Points] Kernel Ridge Regression

Let's first generate a set of data using the following code (or some similar code in Python). I just saved the data from the R code and loaded it here:

```
[2]: train = np.array(pd.read_csv('./data/train-q1.csv'))
     test = np.array(pd.read_csv('./data/test-q1.csv'))
     train_x = train[:,0:2]
     train_y = train[:,2]
     test_x = test[:,0:2]
     test_y = test[:,2]
```

[5 Points] As a comparison, we can first fit a ridge regression to this data. Use the `train` part to fit a ridge regression using the `glmnet()` package with 10-fold cross-validation. Use `lambda.min` as your tuning parameter to predict the testing data. What is the prediction error?

Using the following code:

```
library(glmnet)
ridge.fit = cv.glmnet(x = train[, 1:2], y = train[, 3])
mean((predict(ridge.fit, test[, 1:2], s = "lambda.min") - test[, 3])^2)
```

I got a prediction error of 3.226

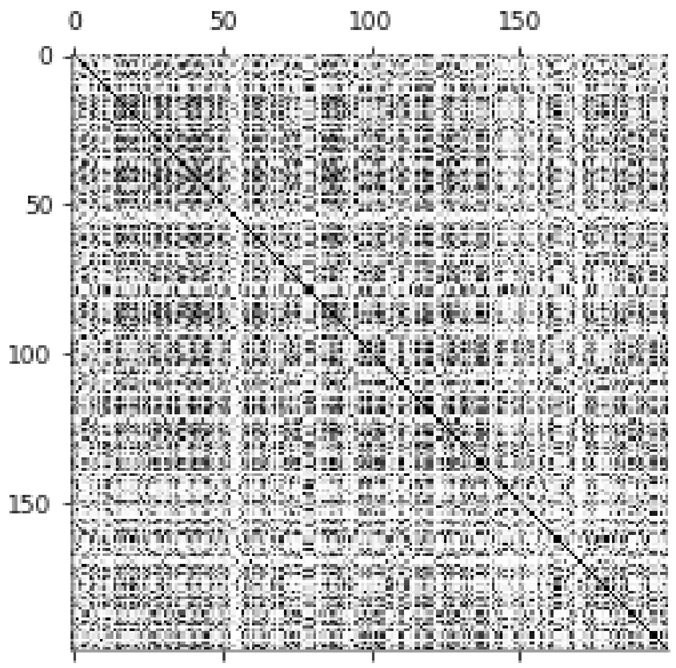
To fit a kernel ridge regression, we use the following formulation:

$$\underset{\boldsymbol{\alpha}}{\text{minimize}} \quad \frac{1}{n}\|\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}\|^2 + \lambda \boldsymbol{\alpha}^{\mathrm{T}} \mathbf{K} \alpha$$

It should be noted that you could add an α_0 term to this regression to rigorously add the intercept term. However, the theoretical mean of Y is zero. Hence, it is not a crucial issue, and not required here. Following our derivation in the class, perform the following to complete this kernel ridge regression:

- [10 Points] Compute the 200 $\times$ 200 kernel matrix $\mathbf{K}$. Use a Gaussian kernel and $\sigma = 1$.

[3]:
```
def gaus(x,z,sigma=1):
    exp_this = -np.sum((x-z)**2)/(2*sigma**2)
    return np.exp(exp_this)

k = np.zeros((200,200))

for i, row in enumerate(train_x):
    for j, row2 in enumerate(train_x):
        k[i,j] = gaus(row, row2)

plt.matshow(k)
plt.show()
```



- [10 Points] Based on the kernel ridge regression derivation, get the solution $\boldsymbol{\alpha}$. Use $\lambda = 0.01$

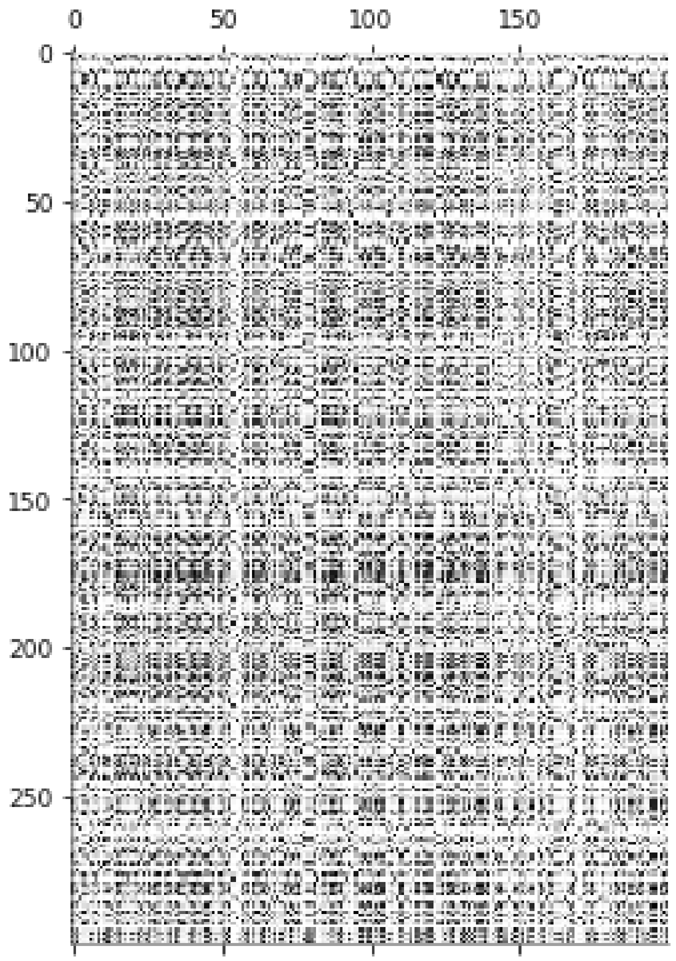
[4]:
```
lam = 0.01
n = train.shape[0]
I = np.identity(n)
alpha = np.linalg.inv(k+n*lam*I)@train_y
```

- [10 Points] The prediction of kernel ridge regression involves computing the kernel between a testing data and all the training data. To be specific, if we have a new testing data z, then the prediction $\widehat{f}(z)$ is

$$\widehat{f}(z) = \sum_{i=1}^{n} \alpha_i K(z, x_i)$$

Since we have 300 testing data, this involves calculating a 300×200 testing data kernel matrix, denoted as K_{test}, and the vector of prediction is $K_{\text{test}}\boldsymbol{\alpha}$.

[5]:
```
k_test = np.zeros((300,200))

for i, row in enumerate(test_x):
    for j, row2 in enumerate(train_x):
        k_test[i,j] = gaus(row, row2)
plt.matshow(k_test)
plt.show()
```



- [10 Points] Predict the testing data using the kernel and $\widehat{\boldsymbol{\alpha}}$. What is the prediction error? Is it better than the ridge regression?

[6]:
```
preds = k_test @ alpha
prediction_error = ((preds - test_y)**2).mean()
print(f'Kernel ridge regression error: {prediction_error:0.4f}.')
print('This was less than the error from ridge regression from glmnet package␣
↪(3.226).')
```

```
Kernel ridge regression error: 0.7996.
This was less than the error from ridge regression from glmnet package (3.226).
```

2.1 Question 2 [55 Points] Non-linear SVM as Penalized Version

Recall that in HW8, we solved SVM using a penalized loss framework, with the logistic loss function:

$$L(y, f(x)) = \log(1 + e^{-yf(x)}).$$

Again, we can specify the form of $f(\cdot)$ as in a RKHS. And the penalized objective function becomes

$$\frac{1}{n}\sum_{i=1}^{n} L(y_i, K_i^{\mathrm{T}}\boldsymbol{\alpha}) + \lambda\alpha^{\mathrm{T}}\mathbf{K}\alpha$$

where $\mathbf{K}$ is the same $n \times n$ kernel matrix as before and K_i is the ith column of $\mathbf{K}$. I saved the data generated from the R code provided and loaded it here:

[7]:
```
train = np.array(pd.read_csv('./data/train-q2.csv'))
test = np.array(pd.read_csv('./data/test-q2.csv'))
train_x = train[:,0:2]
train_y = train[:,2]
test_x = test[:,0:2]
test_y = test[:,2]
```

Similar to the HW8 linear SVM penalized version, we perform the following steps to complete this model fitting:

- [15 Points] Write a penalized loss objective function `SVMfn(b, K, y, lambda)` corresponding to the form we specified. Make sure to use the $1/n$ scale in the loss function part.

[8]:
```
# TO-DO: above
def loss(y, f_x):
    return np.log(1+np.exp(-y*f_x))

def SVMfn(b, K, y, lam):
    n = len(y)

    to_return = 0
    for i in range(n):
        to_return = to_return + loss(y[i],K[:,i].T@b)

    to_return = to_return/n + lam*b.T@K@b
    return to_return
```

- [20 Points] Drive the gradient of the loss function, typeset with LaTex. Then write a gradient function `SVMgr(b, K, y, lambda)` to implement it.

$SVMfn:$

$$P(\alpha) = \frac{1}{n}\sum_{i=1}^{n} log(1 + e^{-y_i K_i^T \alpha}) + \lambda \alpha^T K \alpha$$

This becomes the thing we want to minimize, so our object is:

$$\underset{\alpha}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} log(1 + e^{-y_i K_i^T \alpha}) + \lambda \alpha^T K \alpha$$

$SVMgn:$
In order to accomplish this objective, we need to calculate the gradient of P with respect to α.

$$\begin{aligned}\nabla_\alpha P(\alpha) &= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{1 + e^{-y_i K_i^T \alpha}} e^{-y_i K_i^T \alpha} - y_i K_i + 2\lambda K\alpha \\ &= \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i K_i}{(1 + e^{-y_i K_i^T \alpha})e^{y_i K_i^T \alpha}} + 2\lambda K\alpha \\ &= \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i K_i}{1 + e^{y_i K_i^T \alpha}} + 2\lambda K\alpha\end{aligned}$$

[9]:
```python
def SVMgr(b, K, y, lam):
    grad = 0
    n = len(y)

    for i in range(n):
        grad = grad - (y[i] * K[:,i])/(1+np.exp(y[i]*K[:,i].T@b))

    grad = grad / n + 2 * lam * K @ b

    return grad

def SVMfnAndgn(b,K,y,lam):
    return (SVMfn(b,K,y,lam), SVMgr(b,K,y,lam))
```

- [20 Points] Use the `optim()` function to solve this optimization problem by supplying the objective function and gradient function. Use $\sigma = 0.2$ and $\lambda = 0.01$. Initialize your coefficients as zeros. Report the following:
  - The optimal loss function
  - A confusion table of the training data
  - Mis-classification rate on training data.
  - It could be difficult to obtain the decision line itself. However, its relatively easy to obtain the fitted label for the testing data. Hence, calculate the fitted label of the

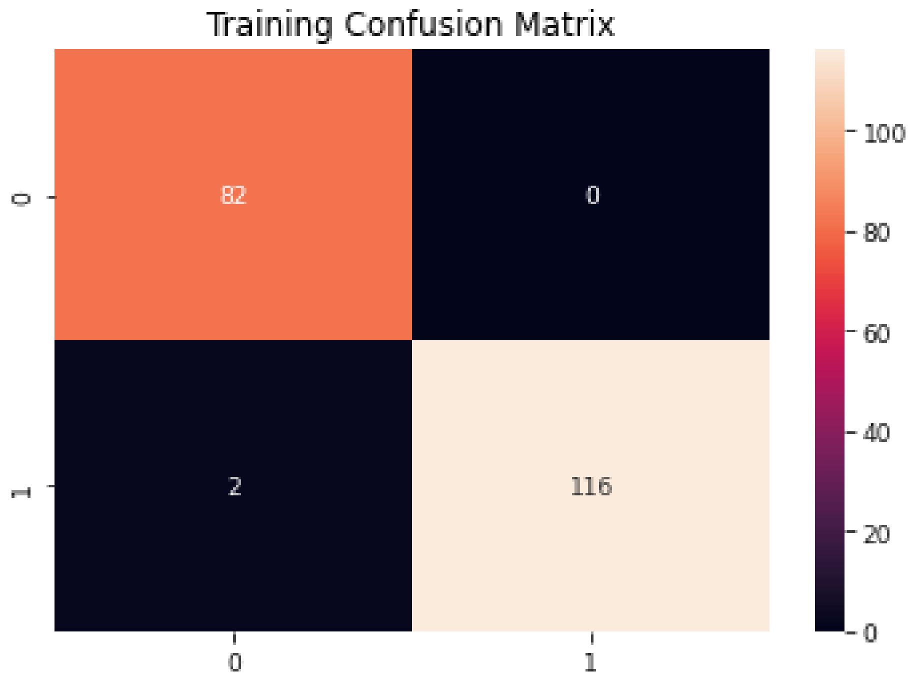
testing data and report the classification error. Plot the testing data using the fitted labels as colors. You should also add the true decision line (since you already know the true data generator). This would allow you to visualize (approximately) the decision line. You do not need to plot the original labels.

[10]:
```
# TO-DO: above

sigma = 0.2
lam = 0.01
initial_guess = np.zeros(200)

# 1. calculate ur train kernel
K_train = np.zeros((200,200))

for i, row in enumerate(train_x):
    for j, row2 in enumerate(train_x):
        K_train[i,j] = gaus(row, row2, sigma)

# 2. use that to optimize SVMfn using SVMfnAndgn
minimization_results = minimize(SVMfnAndgn, initial_guess, args=(K_train,␣
↪train_y, lam), method='BFGS', jac=True)
```

[11]:
```
# 3. report the optimal loss function
print(f'Optimal Loss: {minimization_results.fun:0.4f}')

# 4. determine training predictions
pred_train = K_train @ minimization_results.x
mask = pred_train > 0
pred_train[mask] = 1
pred_train[~mask] = -1

# 5. report the confusion table of the training data
cm = confusion_matrix(train_y, pred_train)
f = sns.heatmap(cm, annot=True, fmt='d')
f.set_title('Training Confusion Matrix')

# 6. report the misclassification rate on training data
print(f'Training misclassification rate: {np.mean(pred_train != train_y)}')

# 7. calculate ur test kernel
K_test = np.zeros((300,200))

for i, row in enumerate(test_x):
    for j, row2 in enumerate(train_x):
        K_test[i,j] = gaus(row, row2, sigma)

# 8. use that to calculate test predictions
```

```
preds_test = K_test @ minimization_results.x
mask = preds_test > 0
preds_test[mask] = 1
preds_test[~mask] = -1

# 9. report classification error
print(f'Testing misclassification rate: {np.mean(preds_test != test_y):0.4f}')
```

```
Optimal Loss: 0.6036
Training misclassification rate: 0.01
Testing misclassification rate: 0.0867
```



Note: 1) In a real problem, you can perform cross-validation to find the best tuning parameters. 2) You may also add a constant term α_0 to the problem to take care of intercept. These two are not required for the HW.