DS-GA 1003 HW4 Kernal Methods

Yiyan Chen

February 2018

1 Introduction

2 [Optional]Kernel Matrices

3 Kernel Ridge Regression

3.1

$$\frac{\partial_w J(w)}{\partial w} = 2X^\top (Xw - y) + 2\lambda w$$

To minimize $J(w)$:

$$\begin{aligned}
2X^\top (Xw - y) + 2\lambda w &= 0 \\
X^\top Xw - X^\top y + \lambda w &= 0 \\
X^\top wX + \lambda Iw &= X^\top y \\
(X^\top X + \lambda I)w &= X^\top y
\end{aligned}$$

Based on the positive semidefinite matrix theory: if M can be factorized as $= R^\top R$ for some matrix R, M is psd.
Hence, $X^\top X$ is a psd matrix. For $\lambda > 0$, $X^\top X + \lambda I$ is a spd, so that $X^\top X + \lambda I$ is invertible. Hence:

$$w = ((X^\top X) + \lambda I)^{-1} X^\top y$$

3.2

$$\begin{aligned}
w &= \frac{1}{\lambda}(X^\top y - X^\top Xw) \\
&= \frac{1}{\lambda} X^\top (y - Xw) \\
w &= X^\top \alpha \\
\alpha &= \frac{1}{\lambda}(y - Xw)
\end{aligned}$$

3.3

$$\begin{bmatrix} x_1 & x_2 \dots x_n \end{bmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \alpha_1(x_1) \dots \alpha_n(x_n) = \sum_{i=1}^{n} \alpha_i x_i$$

w is the span of the data

3.4

Based on the α expression in the second question:

$$\begin{aligned} \lambda\alpha &= y - Xw \\ \lambda\alpha &= y - XX^\top\alpha \\ (\lambda I + XX^\top)\alpha &= y \\ \alpha &= (\lambda I + XX^\top)^{-1} y \end{aligned}$$

3.5

$$Xw = X(X^\top\alpha) = XX\top(\lambda I + XX^\top)^{-1} y$$

3.6

$$\begin{aligned} f(x) &= x^\top w^* \\ &= x^\top X\top(\lambda I + XX^\top)^{-1} y \\ &= x^\top \begin{bmatrix} x_1 \dots x_n \end{bmatrix} (\lambda I + XX^\top)^{-1} y \\ &= k_x(\lambda I + XX^\top)^{-1} y \end{aligned}$$

4 Optional

5 Kernelized Pegasos

5.1

$$
\begin{aligned}
y_i < w^{(t)}, x_j > &= y_j < \sum_{i=1}^{n} \alpha_i^{(t)} x_i, x_j > \\
&= y_i (\sum_{i=1}^{n} \alpha_i^{(t)} x_i)^\top x_j \\
&= y_i (\alpha_1^{(t)} x_1 + \dots \alpha_n^{(t)} x_n)^\top x_j \\
&= y_j (x_1^\top \dots x_n^\top)(\alpha_1^{(t)} \dots \alpha_n^{(t)})^\top x_j \\
&= y_j (< x_1, x_j >, \dots, < x_n, x_j >) \alpha^{(t)} \\
&= y_j K_{j.} \alpha^{(t)}
\end{aligned}
$$

5.2

If the selected point does not have a margin violation: η is the step size

$$
\begin{aligned}
y_j w_t^\top x_j &\geq 1 \\
obj &= \frac{\lambda}{2} ||w||^2 \\
\Delta obj &= 2\lambda w \\
w^{(t+1)} &= w^{(t)} - \eta \lambda w^{(t)} \\
&= (1 - \eta\lambda) w^{(t)} \\
&= (1 - \eta\lambda) \sum_{i=1}^{n} \alpha_i^{(t)} x_i \\
&= \sum_{i=1}^{n} (1 - \eta\lambda) \alpha_i^{(t)} x_i \\
&= \sum_{t=1}^{n} \alpha^{(t+1)} x_i
\end{aligned}
$$

Hence:

$$
\alpha^{(t+1)} = (1 - \eta\lambda) \alpha^{(t)}
$$

5.3

If the selected point have a margin violation:

$$
\begin{aligned}
obj &= \frac{\lambda}{2}||w||^2 + \max(0, 1 - y_i w^\top x_i) \\
\Delta obj &= \lambda w^{(t)} - y_j x_j \\
w^{(t+1)} &= w^{(t)} - \eta(\lambda w^{(t)} - y_j x_j) \\
&= w^{(t)} - \eta(\lambda w^{(t)} - y_j x_j) \\
&= (1 - \eta\lambda) w^{(t)} + \eta y_j x_j \\
&= (1 - \eta\lambda) \sum_{i=1}^{n} \alpha_i^{(t)} x_i + \eta y_j x_j
\end{aligned}
$$

Since $w^{(t+1)} = \sum_{i=1}^{n} \alpha_i^{(t+1)} x_i$:
We can see $\alpha_i = (1 - \eta\lambda)\alpha_i$ for $i = 1 \dots n$, and $\alpha_j = \alpha_\eta y_j$

Algorithm 1 Kernelized Pegasos Algorithm

input: kernel matrix K and the labels $y_1 \dots y_n \in \{-1, 1\}$
$\alpha^{(1)} = (0, \dots, 0) \in R^n$
$t = 0$
repeat:
$t = t + 1$
$\eta^{(t)} = 1/(t\lambda)$
$\alpha^{(t+1)} = (1 - \eta^{(t)}\lambda)\alpha^{(t)}$
randomly choose j in $1, \dots, n$
if $y_j K_{j.} \alpha^{(t)} < 1$ **then**
    $\alpha_j^{(t+1)} = \alpha_j^{(t+1)} + \eta^{(t)} y_j$
Until bored
Return $\alpha^{(t)}$

6 Kernel Methods: Let's Implement

6.1 Review

6.2 Kernels and Kernel Machines

6.2.1

```
def linear_kernel(X1, X2):
    """
    Computes the linear kernel between two sets of vectors.
    Args:
        X1 - an n1xd matrix with vectors x1_1,...,x1_n1 in the rows
        X2 - an n2xd matrix with vectors x2_1,...,x2_n2 in the rows
    Returns:
        matrix of size n1xn2, with x1_i^T x2_j in position i,j
    """
    return np.dot(X1,np.transpose(X2))

def RBF_kernel(X1,X2,sigma):
    """
    Computes the RBF kernel between two sets of vectors
    Args:
        X1 - an n1xd matrix with vectors x1_1,...,x1_n1 in the rows
        X2 - an n2xd matrix with vectors x2_1,...,x2_n2 in the rows
        sigma - the bandwidth (i.e. standard deviation) for the RBF/Gaussian kernel
    Returns:
        matrix of size n1xn2, with exp(-||x1_i-x2_j||^2/(2 sigma^2)) in position i,j
    """
    #TODO
    dis = distance.cdist(X1,X2,'sqeuclidean')
    return np.exp(-dis/(2*sigma**2))

def polynomial_kernel(X1, X2, offset, degree):
    """
    Computes the inhomogeneous polynomial kernel between two sets of vectors
    Args:
        X1 - an n1xd matrix with vectors x1_1,...,x1_n1 in the rows
        X2 - an n2xd matrix with vectors x2_1,...,x2_n2 in the rows
        offset, degree - two parameters for the kernel
    Returns:
        matrix of size n1xn2, with (offset + <x1_i,x2_j>)^degree in position i,j
    """
    #TODO
    return (offset+linear_kernel(X1,X2))**degree
```

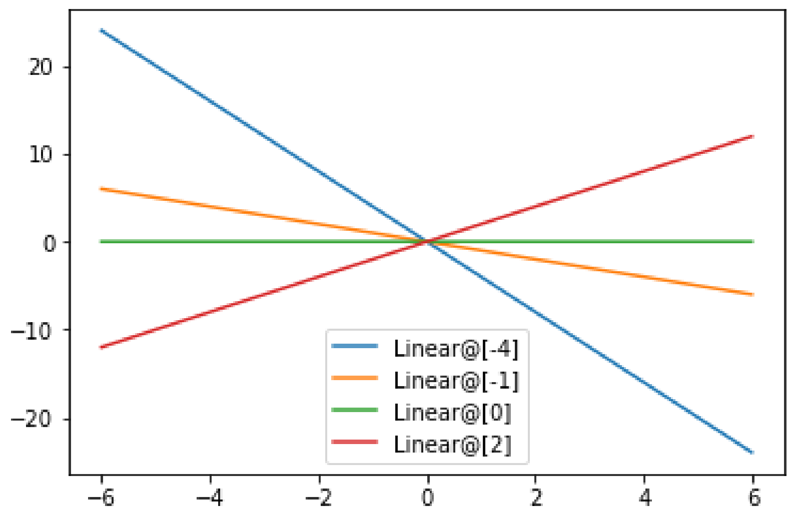
6.2.2

```
x_0 = np.array([-4,-1,0,2]).reshape(-1,1)
linear_kernel(x_0, x_0)
```

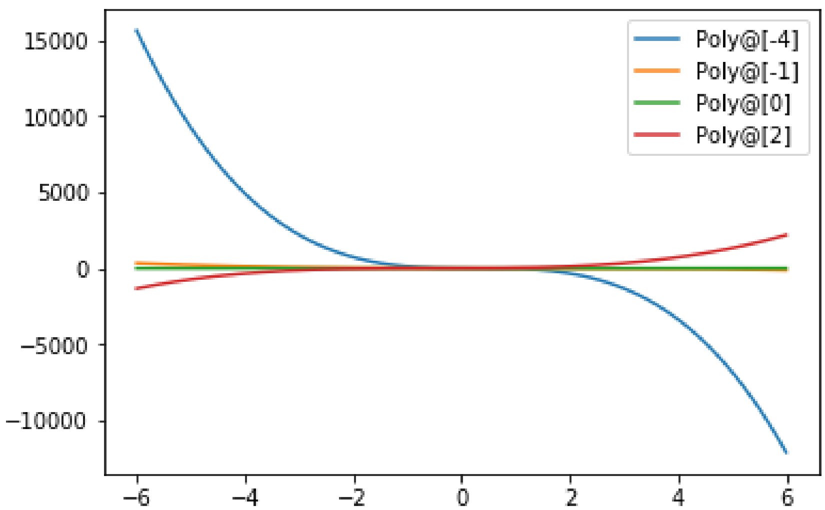
```
array([[16,  4,  0, -8],
       [ 4,  1,  0, -2],
       [ 0,  0,  0,  0],
       [-8, -2,  0,  4]])
```

6.2.3

(a)

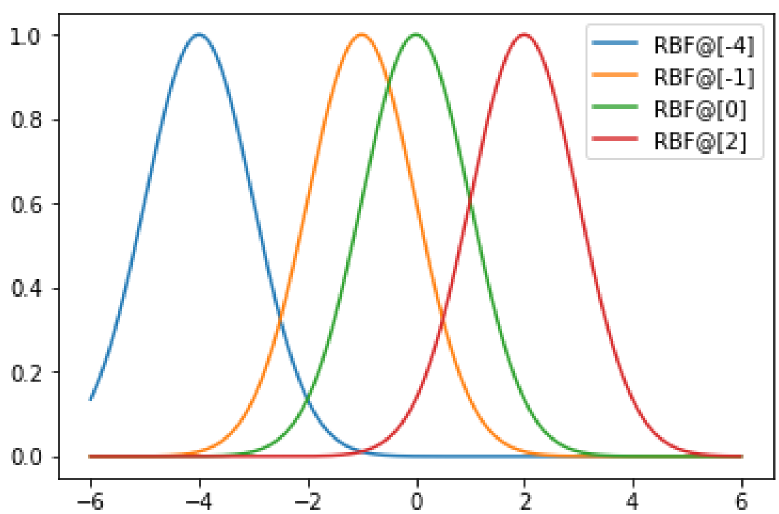


(b)



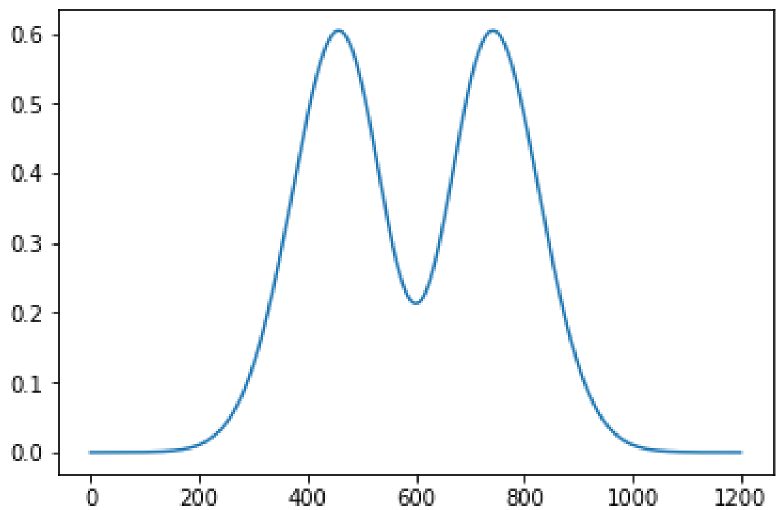
15000
10000
5000
0
−5000
−10000
−6
−4
−2
0
2
4
6
Poly@[-4]
Poly@[-1]
Poly@[0]
Poly@[2]

(c)



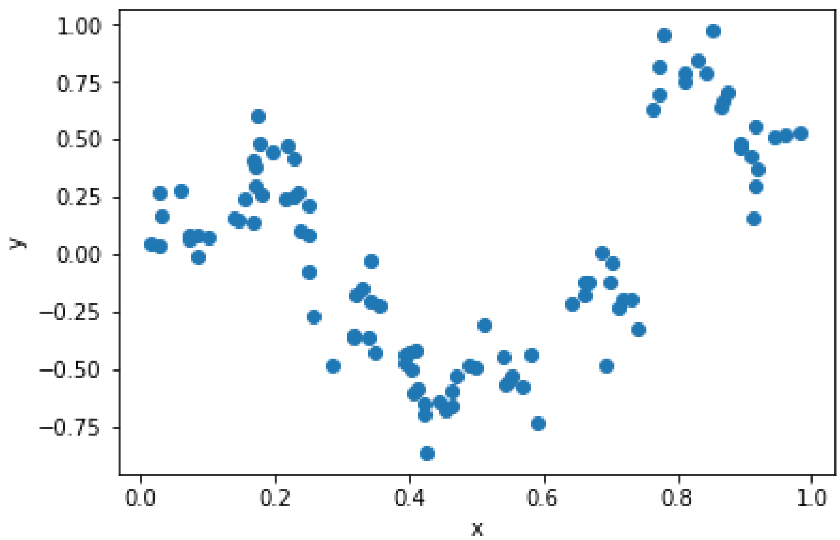
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6
RBF@[-4]
RBF@[-1]
RBF@[0]
RBF@[2]

(d)


0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
200
400
600
800
1000
1200

6.3 Kernel Ridge Regression

6.3.1

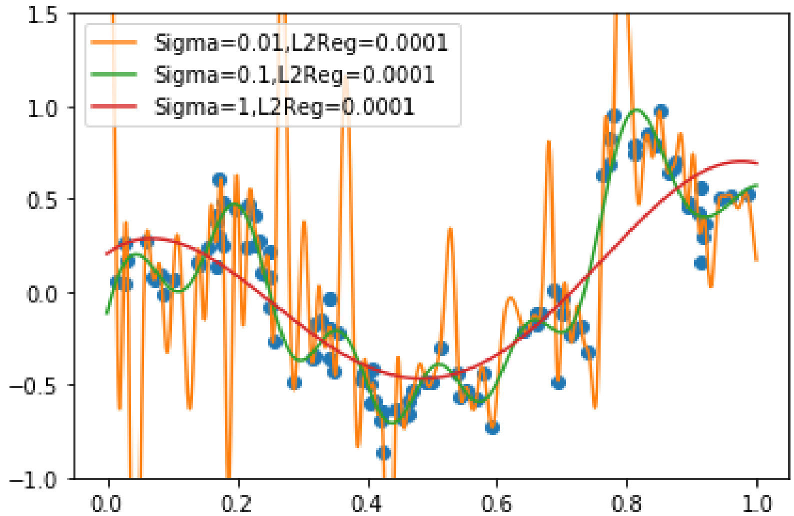


We can see the relationship between x and y is not linear.

6.3.2

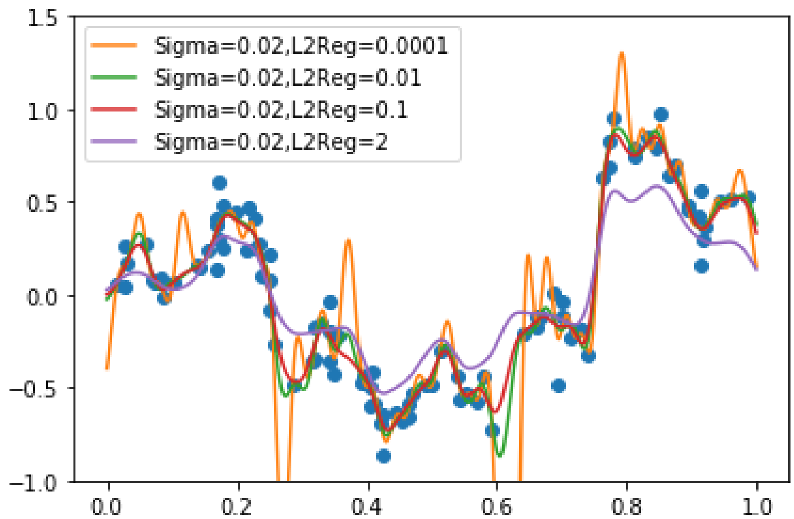
```
def train_kernel_ridge_regression(X, y, kernel, l2reg):
    # TODO
    knl = kernel(X,X)
    I = np.identity(knl.shape[0])
    alpha = np.dot(np.linalg.inv(l2reg*I+knl),y)
    return Kernel_Machine(kernel, X, alpha)
```

6.3.3



Smaller sigma values I think would more likely to overfit, in this case, when the sigma equals to 0.01. And bigger sigma values would less likely to overfit since the curve is smoother, such as when sigma equals to 1.

6.3.4



When the lambda is bigger, the prediction line is smoother. So when lambda goes to infinity, the prediction function will get smooth, which is less likely to overfit.

6.3.5

The kernel type: RBF

	param_kernel	param_l2reg	param_sigma	mean_test_score	mean_train_score
5012	RBF	0.176777	0.060	0.013805	0.014446
4992	RBF	0.170755	0.060	0.013806	0.014391
5032	RBF	0.183011	0.060	0.013807	0.014503
4972	RBF	0.164938	0.060	0.013808	0.014337
5052	RBF	0.189465	0.060	0.013810	0.014562
4952	RBF	0.159320	0.060	0.013813	0.014284
4713	RBF	0.105112	0.065	0.013814	0.014479
4693	RBF	0.101532	0.065	0.013814	0.014432
4733	RBF	0.108819	0.065	0.013815	0.014528
5072	RBF	0.196146	0.060	0.013816	0.014623

Show that make a small change in l2reg:

	param_kernel	param_l2reg	param_sigma	mean_test_score	mean_train_score
250	RBF	0.176777	0.06	0.013805	0.014446
249	RBF	0.170755	0.06	0.013806	0.014391
251	RBF	0.183011	0.06	0.013807	0.014503
248	RBF	0.164938	0.06	0.013808	0.014337
252	RBF	0.189465	0.06	0.013810	0.014562
247	RBF	0.159320	0.06	0.013813	0.014284
253	RBF	0.196146	0.06	0.013816	0.014623
246	RBF	0.153893	0.06	0.013819	0.014233
254	RBF	0.203063	0.06	0.013824	0.014686
245	RBF	0.148651	0.06	0.013828	0.014184
255	RBF	0.210224	0.06	0.013834	0.014750
244	RBF	0.143587	0.06	0.013838	0.014136
256	RBF	0.217638	0.06	0.013847	0.014817
243	RBF	0.138696	0.06	0.013849	0.014089
257	RBF	0.225313	0.06	0.013862	0.014885
242	RBF	0.133972	0.06	0.013863	0.014043
241	RBF	0.129408	0.06	0.013877	0.013999

Show that make a small change in sigma:

	param_kernel	param_l2reg	param_sigma	mean_test_score	mean_train_score
12	RBF	0.176777	6.000000e-02	0.013805	0.014446
11	RBF	0.176777	5.500000e-02	0.013984	0.013652
13	RBF	0.176777	6.500000e-02	0.014045	0.015379
10	RBF	0.176777	5.000000e-02	0.014404	0.012841
14	RBF	0.176777	7.000000e-02	0.014832	0.016613
9	RBF	0.176777	4.500000e-02	0.014883	0.011975
8	RBF	0.176777	4.000000e-02	0.015282	0.011135
7	RBF	0.176777	3.500000e-02	0.015656	0.010365
15	RBF	0.176777	7.500000e-02	0.016159	0.018182
6	RBF	0.176777	3.000000e-02	0.016303	0.009675
3	RBF	0.176777	1.500000e-02	0.016778	0.008280
4	RBF	0.176777	2.000000e-02	0.016992	0.008708

The kernel type: polynomial Show that make a small change in l2reg:

	param_kernel	param_l2reg	param_degree	param_offset	mean_test_score	mean_train_score
57	polynomial	0.001586	14	-6	0.024660	0.031331
56	polynomial	0.001480	14	-6	0.024672	0.032009
61	polynomial	0.002093	14	-6	0.024854	0.030574
52	polynomial	0.001122	14	-6	0.024924	0.036988
51	polynomial	0.001047	14	-6	0.024956	0.037785
50	polynomial	0.000977	14	-6	0.025052	0.039320
54	polynomial	0.001289	14	-6	0.025418	0.033748
66	polynomial	0.002960	14	-6	0.025564	0.029528
64	polynomial	0.002577	14	-6	0.025618	0.029612
63	polynomial	0.002405	14	-6	0.025655	0.029776
62	polynomial	0.002244	14	-6	0.025682	0.029852
65	polynomial	0.002762	14	-6	0.025908	0.029354
68	polynomial	0.003401	14	-6	0.025955	0.029188
70	polynomial	0.003906	14	-6	0.026151	0.028849
72	polynomial	0.004487	14	-6	0.026545	0.028945
76	polynomial	0.005921	14	-6	0.026723	0.029185

Show that make a small change in offset:

	param_kernel	param_l2reg	param_degree	param_offset	mean_test_score	mean_train_score
4	polynomial	0.001586	14	-6	0.024660	0.031331
5	polynomial	0.001586	14	-5	0.027410	0.028653
9	polynomial	0.001586	14	-1	0.029786	0.043859
8	polynomial	0.001586	14	-2	0.032687	0.036030
6	polynomial	0.001586	14	-4	0.032818	0.030837
7	polynomial	0.001586	14	-3	0.059947	0.050606
0	polynomial	0.001586	14	-10	3.839574	0.800597
3	polynomial	0.001586	14	-7	17.180287	8.935204
1	polynomial	0.001586	14	-9	71.841665	73.304396
2	polynomial	0.001586	14	-8	108623.612032	71030.233578

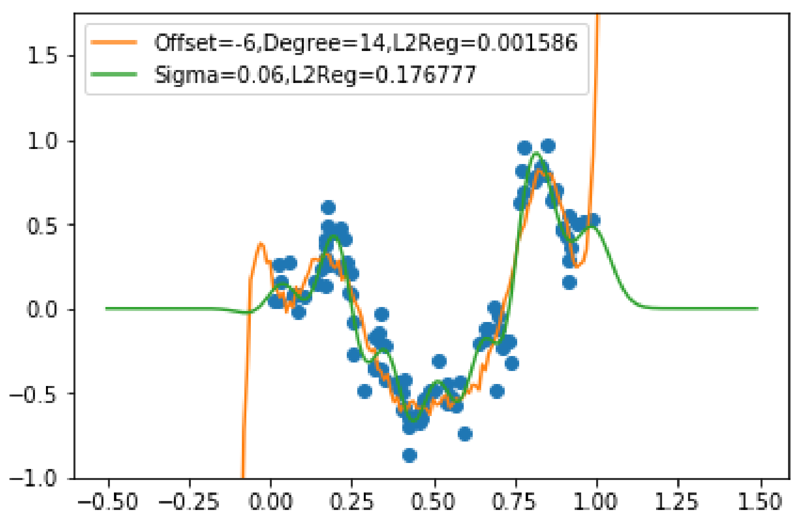
Show that make a small change in degree:

	param_kernel	param_l2reg	param_degree	param_offset	mean_test_score	mean_train_score
10	polynomial	0.001586	14	-6	0.024660	0.031331
9	polynomial	0.001586	13	-6	0.029886	0.029703
8	polynomial	0.001586	12	-6	0.030445	0.031115
6	polynomial	0.001586	10	-6	0.031442	0.038694
5	polynomial	0.001586	9	-6	0.031482	0.039292
4	polynomial	0.001586	8	-6	0.032301	0.039440
0	polynomial	0.001586	4	-6	0.034799	0.052644
1	polynomial	0.001586	5	-6	0.040457	0.043195
2	polynomial	0.001586	6	-6	0.043192	0.042639
3	polynomial	0.001586	7	-6	0.044805	0.044029
7	polynomial	0.001586	11	-6	0.109984	0.102572
13	polynomial	0.001586	17	-6	2.393718	0.409655
11	polynomial	0.001586	15	-6	7.396540	3.599513
14	polynomial	0.001586	18	-6	37.030094	30.379074
15	polynomial	0.001586	19	-6	491.993856	462.050327
12	polynomial	0.001586	16	-6	1649.805109	1441.131048

The kernel type: linear Show that make a small change in l2reg:

	param_kernel	param_l2reg	mean_test_score	mean_train_score
3393	linear	3.904127	0.16451	0.206560
3394	linear	3.917681	0.16451	0.206561
3392	linear	3.890620	0.16451	0.206560
3395	linear	3.931282	0.16451	0.206561
3391	linear	3.877159	0.16451	0.206560
3396	linear	3.944931	0.16451	0.206561
3390	linear	3.863745	0.16451	0.206559
3397	linear	3.958627	0.16451	0.206562
3389	linear	3.850378	0.16451	0.206559
3398	linear	3.972370	0.16451	0.206562
3388	linear	3.837056	0.16451	0.206558
3399	linear	3.986161	0.16451	0.206562
3387	linear	3.823781	0.16451	0.206558
3386	linear	3.810552	0.16451	0.206558
3385	linear	3.797368	0.16451	0.206557
3384	linear	3.784231	0.16451	0.206557

6.3.6



Both of the kernels perform well in the regime of our data.

6.3.7

Bayes Decision Function:

$$E(\epsilon^2) = Var(\epsilon) - E(\epsilon)^2 = 0.1^2 - 0 = 0.01$$

$$
\begin{aligned}
f^* &= argminEL(\hat{y}, y) = argminEL(\hat{y}, f(x) + \epsilon) \\
EL(\hat{y}, f(x) + \epsilon) &= E[((\hat{y} - f(x))^2 - 2\epsilon(\hat{y} - f(x)) + \epsilon^2] \\
&= E(\hat{y} - f(x))^2 - 2E(\epsilon)E(\hat{y} - f(x)) + \epsilon^2 \\
&= E(\hat{y} - f(x))^2 - 2E(\epsilon)E(\hat{y} - f(x)) + E(\epsilon^2) \\
&= E(\hat{y} - f(x))^2 + 0 + 0.01 \\
&= E(\hat{y} - f(x))^2 + 0.01
\end{aligned}
$$

Bayes Risk: To minimize the Bayes Decision Function, $E(\hat{y} - f(x))^2 = 0$, that is, $\hat{y} = f(x)$

$$minEL(\hat{y}, f(x) + \epsilon) = 0.01$$

7 Representer Theorem

7.1

$$||x - m_0||^2 = ||x||^2 - ||m_0||^2 = < x - m_0, x - m_0 >$$
$$\text{Since } ||x||^2 \geq ||m_0||^2$$
$$< x - m_0, x - m_0 > \geq 0 \text{ and } < x - m_0, x - m_0 > = 0 \iff x - m_0 = 0$$

Hence, we have $||m_0|| = ||x||$ only when $m_0 = x$

7.2

If $||w|| = ||w^*||$, then from the previous problem we know that $w = w^*$. So if $w^* = \sum_{i=1}^{n} \alpha_i \psi(x_i)$, then must$w = \sum_{i=1}^{n} \alpha_i \psi(x_i)$
If $||w|| < ||w^*||$, let M = $span(x_1, \ldots, x_n)$, and w as the projection of w^* on M. $w^\perp = w^* - w$. $< w, x_i > = < w^* - w^\perp, x_i > = < w^*, x_i >$ since $< w^\perp, x_i >$ is zero. R is strictly increasing, $R(||w||) < R(||w^*||)$, loss term here doesn't change. So $J(w) < J(w^*)$. This is a contradiction due to the assumption that w^* is an optimal minimizer. In conclusion, all minimizers have this form.

8 Ivanov and Tikhonov Regularization

8.1 Tikhonov optional implies Ivanov optimal

First to find the r value:
Suppose there is a minimizer $f^{**} < f^*$, then it satisfies the (2) condition, so $\phi(f^{**}) < \phi(f^*)$. The (1) has to be contracted, which means $\Omega(f^{**} > \Omega(f^*)$. Due to the condition we need to satisfies in the (1), $\Omega(f) < \Omega(f^{**}$ in order to find the optimal f^*. So $r = \Omega(f^*)$ in which f^* is the minimizer.
Proof:
Suppose there is another minimizer f^{**}. $\phi(f^{**}) < \phi(f^*)$ and $\Omega(f^{**}) \leq \Omega(f^*)$. In (1), $\phi(f^{**} + \lambda\Omega(f^{**}) < \phi(f^*) + \lambda\Omega(f^*)$ which is a contradiction.