Machine Learning and Computational Statistics
Homework 3: SVM and Sentiment Analysis

2.Kernel Matrices

1

$$K = X^T X = \begin{bmatrix} < x_1, x_1 > & < x_1, x_2 > & \dots & < x_1, x_n > \\ \multicolumn{4}{c}{\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots} \\ < x_n, x_1 > & < x_n, x_2 > & \dots & < x_n, x_n > \end{bmatrix}$$

Knowing K is equivalent to knowing $< x_i, x_j >$ for $0 \leq i, j \leq m$
for any x_i

$$\|x_i, x_i\| = \sqrt{< x_i, x_i >}$$

Therefore, knowing K is equivalent to knowing the vector lengths for any x_i, x_j

$$\begin{aligned}
& d(x_i, x_j) \\
& = \|x_i - x_j\| \\
& = \sqrt{\|x_i - x_j\|^2} \\
& = \sqrt{\|(x_i - x_j)^T (x_i - x_j)\|} \\
& = \sqrt{\|x_i^T x_i\| + \|x_j^T x_j\| - 2\|x_i^T x_j\|} \\
& = \sqrt{< x_i, x_i > + < x_j, x_j > -2 < x_i, x_j >}
\end{aligned}$$

Therefore, knowing K is equivalent to knowing the the set of pairwise distances among the vectors in S.

3 Kernel Ridge Regression

1

$$
\begin{aligned}
J(w) &= ||Xw - y||^2 + \lambda ||w||^2 \\
&= (Xw - y)^T (Xw - y) + \lambda w^T w \\
&= w^T X^T X w + y^T y - 2y^T X w + \lambda w^T w \\
\frac{d\ J(w)}{dw} &= 2X^T X w - 2X^T y + 2\lambda I w = 0 \\
X^T X w + \lambda I w &= X^T y
\end{aligned}
$$

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

For any vector $b \neq 0$,

$$
\begin{gathered}
b^T X^T X b = (Xb)^T (Xb) \geq 0 \\
b^T I b > 0
\end{gathered}
$$

Then,

$$b^T (X^T X + \lambda I) b = b^T X^T X b + b^T I > 0$$

Therefore, $X^T X + \lambda I$ is a spd, and spd is invertible.

2

$$w = \frac{1}{\lambda}(X^T y - X^T X w) = X^T \frac{1}{\lambda}(y - Xw) = X^T \alpha$$

where $\alpha = \frac{1}{\lambda}(y - Xw)$

3

w is the linear combination of vector x_i for each i, which means that w is in the span of x_i for each i.

4

$$
\begin{gathered}
\alpha = \frac{1}{\lambda}(y - Xw) = \frac{1}{\lambda}(y - XX^T \alpha) \\
\alpha = (\lambda I + XX^T)^{-1} y
\end{gathered}
$$

5

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

6

$$x^T w^* = x^T X^T (\lambda I + XX^T)^{-1} y = k_x^T (\lambda I + XX^T)^{-1} y$$

4 Pegasos and SSGD for ‘ 2 -regularized ERM

1

$$g_i(w) = \lambda w + v_i(w)$$

2

According to the definition of the subgradient,

$$J_i(z) \geq J_i(x) + g_i^T(z - x)$$

Then,

$$E(J_i(z)) \geq E(J_i(x)) + E(g_i^T)(z - x)$$
$$J(z) \geq J(x) + E(g_i^T)(z - x)$$

Therefore, the expectation of our subgradient of a randomly chosen $J_i(w)$ is in the subdifferential of J.

5. Kernelized Pegasos

1

$$y_j \left\langle w^{(t)}, x_j \right\rangle = y_j \left\langle \sum_{i=1}^{n} \alpha_i^{(t)} x_i, x_j \right\rangle = y_j \sum_{i=1}^{n} \left\langle x_i, x_j \right\rangle \alpha_i^{(t)} = y_j K_j \alpha^{(t)}$$

2

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

3

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

$$\alpha_j^{t+1} = \alpha_j^t + \eta^{(t)} y_j$$

Algorithm 1: Pegasos Algorithm

input: Training set Kernel matrix K and $y_1, y_2, ..., y_n \in \{-1, 1\}$ and $\lambda > 0$.
$w^{(1)} = (0, \ldots, 0) \in \mathbf{R}^d$
$t = 0$ # step number
repeat
  $t = t + 1$
  $\eta^{(t)} = 1/\left(t\lambda\right)$ # step multiplier
  randomly choose j in $1, \ldots, n$
  if $y_j K_j \alpha^{(t)} < 1$
    $\alpha^{t+1} = (1 - \frac{1}{t})\alpha^t$
    $\alpha_j^{t+1} = \alpha_j^t + \eta^{(t)} y_j$
  else
    $\alpha^{t+1} = (1 - \frac{1}{t})\alpha^t$
until bored
return $\alpha^{(t)}$

4

Algorithm 2: Pegasos Algorithm

input: Training set Kernel matrix K and $y_1, y_2, ..., y_n \in \{-1, 1\}$ and $\lambda > 0$.
$w^{(1)} = (0, \ldots, 0) \in \mathbf{R}^d$
$t = 0$ # step number
repeat
  $t = t + 1$
  $\eta^{(t)} = 1/\,(t\lambda)$ # step multiplier
  randomly choose j in $1, \ldots, n$
  if $t == 2$
    $l = 1$
  $l* = 1 - \frac{1}{t}$
  if $y_j K_j \alpha^{(t)} < 1/l$
    $\alpha_j^{t+1} = \alpha_j^t + \eta^{(t)} y_j$
until bored
return $\alpha^{(t)}/l$

6.Kernel Methods: Letâs Implement

6.2 Kernels and Kernel Machines

1

```python
### Kernel function generators
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
    """
    dis = distance.cdist(X1, X2, 'euclidean')
    return np.exp(-dis ** 2 /(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
    """
    return (offset + linear_kernel(X1, X2)) ** degree
```

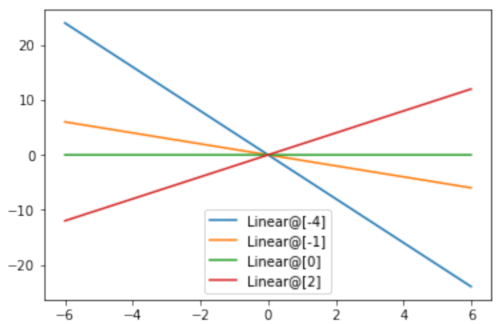
2

```
x = [-4,-1,0,2]
kernel_matrix = np.zeros((len(x), len(x)))
for i in range(len(x)):
    for j in range(len(x)):
        kernel_matrix[i,j] = linear_kernel(x[i], x[j])
```

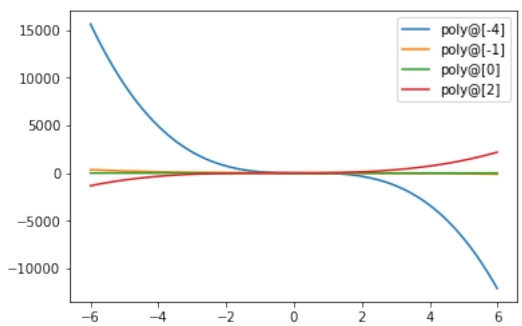
```
array([[ 16.,   4.,   0.,  -8.],
       [  4.,   1.,   0.,  -2.],
       [  0.,   0.,   0.,   0.],
       [ -8.,  -2.,   0.,   4.]])
```

3

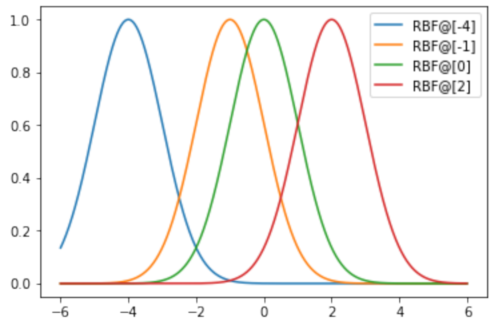
a



b

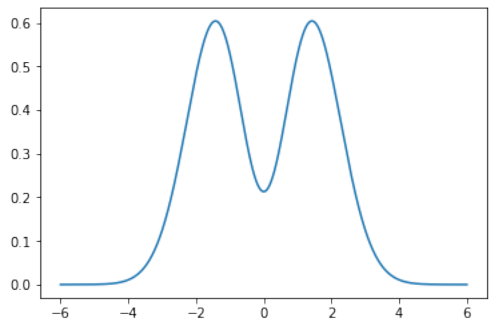


c



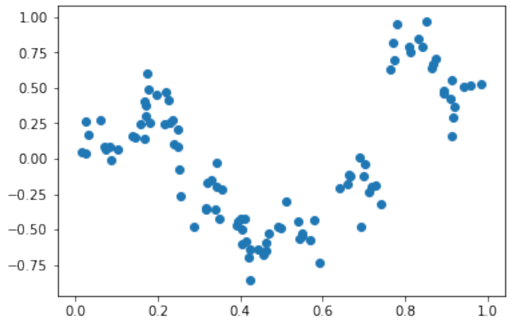
d

```
def predict(self, X):
        """
        Evaluates the kernel machine on the points given by the rows of
 ↪   X
        Args:
            X - an nxd matrix with inputs x_1,...,x_n in the rows
        Returns:
            Vector of kernel machine evaluations on the n points in X.
 ↪   Specifically, jth entry of return vector is
                Sum_{i=1}^R w_i k(x_j, mu_i)
        """
        # TODO
        #weights 3x1 X^TX 3x1000
        preds = self.weights.T.dot(self.kernel(self.prototype_points,
         ↪   X))
        return preds.T
```



6.3 Kernel Ridge Regression

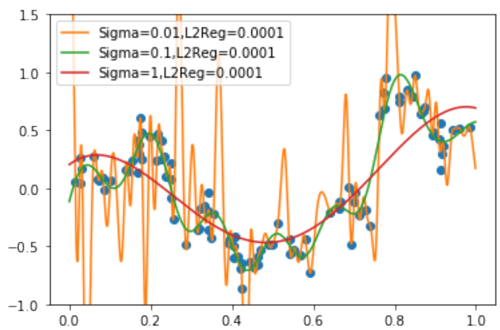
1



2

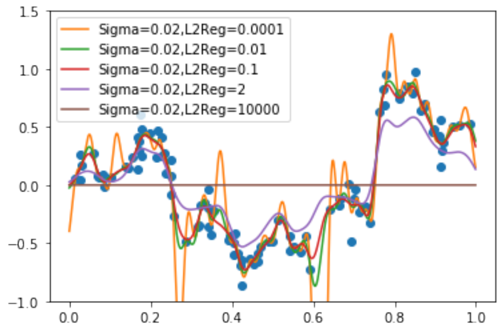
```
def train_kernel_ridge_regression(X, y, kernel, l2reg):
    # TODO
    alpha = inv((l2reg * np.identity(X.shape[0]) + kernel(X,X))).dot(y)
    return Kernel_Machine(kernel, X, alpha)
```

3



Sigma = 0.01 would be most likely to overfit, Sigma = 1 would be less likely to overfit.

4



When $\lambda \to \infty$, the prediction function is a straight line overlapped with x axis.

5

Tuning the hyperparameters for linear kernel,

	param_kernel	param_l2reg	mean_test_score	mean_train_score
2	linear	1.000	0.164540	0.206506
3	linear	0.100	0.164565	0.206501
4	linear	0.010	0.164569	0.206501
5	linear	0.001	0.164569	0.206501
1	linear	10.000	0.164591	0.206780
0	linear	100.000	0.166435	0.209156

Zoom in further to get better results,

	param_kernel	param_l2reg	mean_test_score	mean_train_score
1	linear	3	0.164512	0.206538
2	linear	5	0.164513	0.206592
3	linear	7	0.164534	0.206661
0	linear	1	0.164540	0.206506
4	linear	10	0.164591	0.206780

The best hyperparameter for linear kernel is $param_l 2reg = 3$. Making small change in any one of the hyperparameters in either direction will cause the performance to get worse.

Tuning the hyperparameters for rbf kernel,

	param_kernel	param_l2reg	param_sigma	mean_test_score	mean_train_score
2	RBF	0.0800	0.05	0.015505	0.011973
5	RBF	0.0625	0.05	0.015869	0.011751
8	RBF	0.0400	0.05	0.016507	0.011393
6	RBF	0.0400	0.10	0.020406	0.022413
3	RBF	0.0625	0.10	0.021270	0.023245
0	RBF	0.0800	0.10	0.021797	0.023710
1	RBF	0.0800	0.20	0.031673	0.037145
4	RBF	0.0625	0.20	0.031931	0.036712
7	RBF	0.0400	0.20	0.032221	0.035914

Zoom in further to get better results,

	param_kernel	param_l2reg	param_sigma	mean_test_score	mean_train_score
10	RBF	0.40	0.05	0.013876	0.014305
13	RBF	0.60	0.05	0.014173	0.015527
7	RBF	0.20	0.05	0.014264	0.013010
0	RBF	0.07	0.08	0.015447	0.017194
4	RBF	0.08	0.05	0.015505	0.011973
1	RBF	0.07	0.05	0.015702	0.011851
3	RBF	0.08	0.08	0.015735	0.017553
6	RBF	0.20	0.08	0.018195	0.020335
2	RBF	0.07	0.01	0.018631	0.005354
5	RBF	0.08	0.01	0.018648	0.005545
8	RBF	0.20	0.01	0.019216	0.007551
9	RBF	0.40	0.08	0.020456	0.022718
11	RBF	0.40	0.01	0.021553	0.011029
12	RBF	0.60	0.08	0.021856	0.024255
14	RBF	0.60	0.01	0.024866	0.014914

The best hyperparameters for rbf kernel are sigma $= 0.05$, l2reg$= 0.4$. Making small change in any one of the hyperparameters in either direction will cause the performance to get worse.
Tuning the hyper parameters for polynomial kernel,

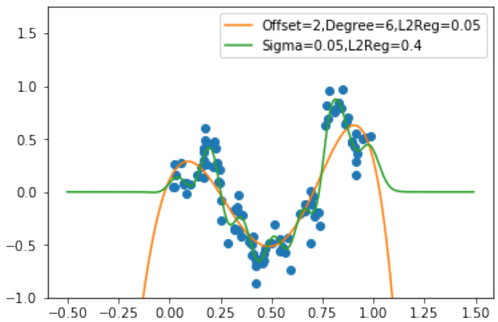
	param_degree	param_kernel	param_l2reg	param_offset	mean_test_score	mean_train_score
14	6	polynomial	0.010	1	0.032700	0.049496
17	6	polynomial	0.005	1	0.034082	0.045336
2	7	polynomial	0.050	1	0.034570	0.055279
20	8	polynomial	0.050	1	0.037000	0.049600
6	7	polynomial	0.005	-1	0.037116	0.046015
5	7	polynomial	0.010	1	0.037157	0.045356
8	7	polynomial	0.005	1	0.039204	0.044079
26	8	polynomial	0.005	1	0.039882	0.043413
23	8	polynomial	0.010	1	0.040309	0.044769
3	7	polynomial	0.010	-1	0.041585	0.049871
...	...	...	...	...	...	...

Zoom in further to get better results,

	param_degree	param_kernel	param_l2reg	param_offset	mean_test_score	mean_train_score
54	6	polynomial	0.050	2	0.032504	0.049690
41	6	polynomial	0.100	3	0.032528	0.048157
48	6	polynomial	0.080	3	0.032669	0.046628
60	6	polynomial	0.010	1	0.032700	0.049496
55	6	polynomial	0.050	3	0.033735	0.044486
47	6	polynomial	0.080	2	0.034064	0.054881
67	6	polynomial	0.005	1	0.034082	0.045336
18	7	polynomial	0.050	1	0.034570	0.055279
5	7	polynomial	0.100	2	0.034673	0.044425
12	7	polynomial	0.080	2	0.035492	0.043866
...	...	...	...	...	...	...
73	8	polynomial	0.100	0	0.128783	0.165052
80	8	polynomial	0.080	0	0.128891	0.165042
87	8	polynomial	0.050	0	0.129061	0.165031
94	8	polynomial	0.010	0	0.129302	0.165024
101	8	polynomial	0.005	0	0.129334	0.165023
78	8	polynomial	0.080	-2	0.138769	0.114552
93	8	polynomial	0.010	-1	0.203748	0.162364
100	8	polynomial	0.005	-1	1.141505	0.689108
85	8	polynomial	0.050	-2	8.243975	4.501527
16	7	polynomial	0.050	-1	20.147533	15.474952

The best hyperparameters for polynomial kernel are offset$=2$, degree $=6$, l2reg $=0.05$. Making small change in any one of the hyperparameters in either direction will cause the performance to get worse.

6



The prediction function with rbf kernel turns out to better fit the data than the one with polynomial kernel. Also, we can see that outside of region of the training points, the polynomial acts very unreasonable.

7

Let Bayes decision function be $f^*(x)$,

$$f^*(x) = E(f(x) + \epsilon) = f(x)$$

The Bayes decision function is $f(x)$, the bayes risk is,

$$\begin{aligned}
E(l(\hat{y}, y)) &= E(\hat{y}^2) + E(y^2) - 2E(\hat{y}y) \\
&= Var(\hat{y}) + E(\hat{y})^2 + Var(y) + E(y)^2 + -2(cov(\hat{y}, y) + E(y)E(\hat{y})) \\
&= 0 + E(\hat{y})^2 + Var(\epsilon) + E(y)^2 - 0 + E(\hat{y})^2 \\
&= 0.01
\end{aligned}$$

7.Representer Theorem

1

When $\|m_0\| = \|x\|$,

$$\|x - m_0\|^2 = 0$$

that is,

$$< x - m_0, x - m_0 >= 0$$

Because the positive-definiteness of the inner product, the above equation imlies,

$$m_0 = x$$

2

Suppose J has a minimizer w, and let $w^* = Proj_M w$, it can be shown that,

$$L(< w, x_1 >, ..., < w, x_n >) = L(< w_*, x_1 >, ..., < w_*, x_n >)$$

Because w^* is the projection of w,

$$\|w^*\| \leq \|w\|$$

In the case $\|w^*\| < \|w\|$, because R is strictly increasing,

$$R(\|w^*\|) < R(\|w\|)$$

Therefore, we can conclude that,

$$J(w^*) < J(w)$$

In the case $\|w^*\| = \|w\|$, because R is strictly increasing,

$$R(\|w^*\|) = R(\|w\|)$$

Therefore, we can conclude that,

$$J(w^*) = J(w)$$

All minimizer has the form claimed in both cases.

8.Ivanov and Tikhonov Regularization

8.1 Tikhonov optimal implies Ivanov optimal

$$r = \Omega(f^*)$$

Suppose Ivanov solution $f \neq f^*$,
We have

$$\Phi(f) < \Phi(f^*)$$

and,

$$\Omega(f) \leq r = \Omega(f^*)$$

Therefore,

$$\Phi(f) + \Omega(f) < \Phi(f^*) + \Omega(f^*)$$

Which contracdicts that f^* is the Tikhonov regularization solution.
Therefore, f^* is also an Ivanov solution.