Machine Learning HW04

Ben Zhang, bz957

March 3, 2018

2 Kernel Matrices

2.1. $X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$ $X^T = [x_1^T, x_2^T \cdots x_m^T]$

$K = XX^T = \begin{bmatrix} x_1x_1^T, x_1x_2^T & \cdots & x_1x_m^T \\ \vdots \quad x_2x_2^T & \ddots & \vdots \\ x_mx_1^T & \cdots & x_mx_m^T \end{bmatrix}$

$\because d(x,y) = \|x-y\| = \sqrt{\langle x-y, x-y\rangle} = \sqrt{x^Tx - 2x^Ty + y^Ty}$

according K, we know the vector lengths through

$$\|x_i\| = \sqrt{x_i^Tx_i}$$

and we know every pairwise distance through

$$d(x_i, x_j) = \sqrt{x_i^Tx_i - 2x_i^Tx_j + x_j^Tx_j}$$

3 Kernel Ridge Regression

3.1 $J'(w;v) = \left[(2w^T X^T X - 2X^T y) + 2\lambda I w\right] v$

$\Rightarrow \nabla J'(w;v) = 2(X^T X w - X^T y + \lambda I w)$

$\because \nabla J'(w;v) = 0, \quad X^T X w + \lambda I w = X^T y$

$w = (X^T X + \lambda I)^{-1} X^T y$ is the minimizer of $J(w)$.

for any $a \in R^d$, $a X^T X a^T = (X a^T)^T (X a^T)$

$= \| X a^T \| \geq 0$

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

for $\lambda > 0$

$\Rightarrow X^T X + \lambda I$ is invertible

3.2. $X^T X w + \lambda I w = X^T y$

$\lambda I w = X^T y - X^T X w$

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

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

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

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

3.3.

$\because w = X^T\alpha \Rightarrow w = \sum_{i=1}^{n} x_i^T \alpha_i + \dots x_n^T \alpha_n$,

$\alpha_i \in R$, is a scalar of x_i,

$\Rightarrow$ w is $\in R^d$ (same dimension as x_i), decided by x_i

so w is in the span of the data

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

$\because w = X^T\alpha \Rightarrow \alpha = \frac{y - XX^T\alpha}{\lambda}$

$\lambda\alpha = y - XX^T\alpha$

$(\lambda I + XX^T)\alpha = y$

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

3.5 $Xw = X \cdot X^T\alpha = X \cdot X^T(\lambda I + XX^T)^{-1} y$

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

$= x^T \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \cdot (\lambda I + XX^T)^{-1} y$

$= \begin{pmatrix} x^T \cdot x_1 \\ \vdots \\ x^T x_n \end{pmatrix} \cdot (\lambda I + XX^T)^{-1} y$

let $k_x = \begin{pmatrix} x^T x_1 \\ \vdots \\ x^T x_n \end{pmatrix} \Rightarrow = k_x \cdot (\lambda I + XX^T)^{-1} y$

4 Pegasos and SSGD for '2-regularized ERM1

4.1 $J_i(w) = \frac{\lambda}{2}\|w\|_2^2 + l_i(w) \Rightarrow g_i(w) = \lambda w + v_i(w)$

4.2. $Eg_i(w) = \frac{1}{n}\sum_{i=1}^{n} g_i(w) = \frac{1}{n}\sum_{i=1}^{n}(\lambda w + v_i(w))$

$= \lambda w + \frac{1}{n}\sum_{i=1}^{n} v_i(w)$

$\nabla J(w) = \lambda w + 2\frac{1}{n}\sum_{i=1}^{n} l_i(w)$

$= \lambda w + \frac{1}{n}\sum_{i=1}^{n} 2l_i(w)$

$= \lambda w + \frac{1}{n}\sum_{i=1}^{n} v_i(w)$

$\Rightarrow Eg_i(w) = \nabla J(w)$

4.3. $w^{(t+1)} = w^{(t)} - \eta^{(t)} \cdot g_i(w^{(t)})$

$= w^{(t)} - 1/\lambda t \cdot (\lambda w^t + v_i(w^t))$

$= \frac{t-1}{t} w^t - \frac{1}{\lambda t} v^{(t)}$

$= \frac{t-1}{t}\left[\frac{t-2}{t-1} w^{t-1} - \frac{1}{\lambda(t-1)} v^{(t-1)}\right] - \frac{1}{\lambda t} v^{(t)}$

$\vdots$

$= \frac{1}{t} w' - \frac{1}{\lambda t}\sum_{\tau=1}^{t} v^{(\tau)}$

$\because w' = (0, \ldots, 0) \in R^d \quad \therefore w^{(t+1)} = -\frac{1}{\lambda t}\sum_{\tau=1}^{t} v^{(\tau)}$

4.3 $\because \theta^{(t)} = \sum_{\tau=1}^{t-1} v^{(\tau)}$,

$v^{(t)}$ is a subgradient of $J_i(w^{(t)})$

$$\Rightarrow w^{(t+1)} = -\frac{1}{\lambda t}\sum_{i=1}^{t}\theta^{(t)}$$

according the algorithm,

if $y_i \langle w^{(t)}, x_i \rangle < 1$,

we have $\theta^{(t+1)} = \theta^{(t)} + y_j x_j$

$y_j x_j$ means we have s accesses if x_j has s nonzero entries.

otherwise, let $\theta^{(t+1)} = \theta^{(t)}$

$\therefore$ the total accesses in every pass through the loop is $O(s)$, i.e. $\leq s$

5 Kernelized Pegasos

5.1

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

$$= y_j \cdot \sum_{i=1}^{n} \alpha_i^{(t)} \cdot \langle x_i, x_j \rangle$$

$$= y_j \cdot [\alpha_1 \cdot \langle x_1, x_j \rangle + \cdots \alpha_n \cdot \langle x_n, x_j \rangle]$$

$$= y_j \cdot (\langle x_1, x_j \rangle \cdots \langle x_n, x_j \rangle) \cdot (\alpha_1^{(t)}, \cdots, \alpha_n^{(t)})^T$$

$$= y_j \cdot K_j \cdot \alpha^{(t)}$$

5.2 if $y_j \langle w^{(t)}, x_j \rangle \geq 1$, i.e. no margin violation

according Algorithm 2, $w^{(t+1)} = (1 - \eta^{(t)} \lambda) w^{(t)}$

$\because w^{(t)} = \sum_{i=1}^{n} \alpha_i^{(t)} x_i$

$\Rightarrow w^{(t+1)} = (1 - \eta^{(t)} \lambda) \cdot \sum_{i=1}^{n} \alpha_i^{(t)} x_i$

if we wanna $w^{(t+1)} = \sum_{i=1}^{n} \alpha_i^{(t+1)} x_i$

$$\Rightarrow \sum_{i=1}^{n} \alpha_i^{(t+1)} x_i = (1 - \eta^{(t)} \lambda) \cdot \sum_{i=1}^{n} \alpha_i^{(t)} \cdot x_i$$

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

5.3 if $y_i \langle w^{(t)}, x_j \rangle < 1$

$$w^{(t+1)} = (1 - \eta^{(t)}\lambda) w^{(t)} + \eta^{(t)} y_i x_i$$

$$\sum_{i=1}^{n} \alpha_i^{t+1} x_i = (1 - \eta^{(t)}\lambda) \sum_{i=1}^{n} \alpha_i^{(t)} x_i + \eta^{(t)} y_j x_j$$

$$\text{for } i \text{ in } [1,n],\ \alpha_i^{t+1} = \begin{cases} (1 - \eta^{(t)}\lambda) \cdot \alpha_i^t, & i \neq j \\ (1 - \eta^t \lambda)\alpha_i^t + \eta^{(t)} y_j, & i = j \end{cases}$$

5.3. Algorithm 2:

input: kernel matrix K, the labels $y_1, \cdots y_n \in \{-1, 1\}$.

$w^{(1)} = (0, \cdots, 0) \in R^d$

$t = 0$

repeat

$t = t + 1$

$\eta^{(t)} = 1/t\lambda$

randomly choose j in $1, \ldots, n$.

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

if $y_i K_{j,} \alpha^{(t+1)} < 1$

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

until bored

return $w^{(t)}$

5.4. Algorithm

let $\alpha = sX$

$X = (0, 0, \dots 0)' \in R^n$

$t = 0$, $s = 1$

repeat $t = t + 1$

$\eta^{(t)} = 1/\lambda t$

if $s \cdot y_i K_{j:}^T X < 1$:

$X = (1 - \eta^{(t)} \lambda) X + \frac{1}{\lambda t} y_i$

else:

$s = (1 - \frac{1}{t}) s$

through the algorithm, we only need to update α when margin violation occurs otherwise only update s, saving a lot time

6.2 Kernels and Kernel Machines

6.2.1

```
def linear_kernel(X1, X2):

    return np.dot(X1,np.transpose(X2))

def RBF_kernel(X1,X2,sigma):

    return np.exp(-distance.cdist(X1, X2, 'sqeuclidean')/(2*sigma**2))


def polynomial_kernel(X1, X2, offset, degree):

    return (offset + np.dot(X1, np.transpose(X2)))**degree
```

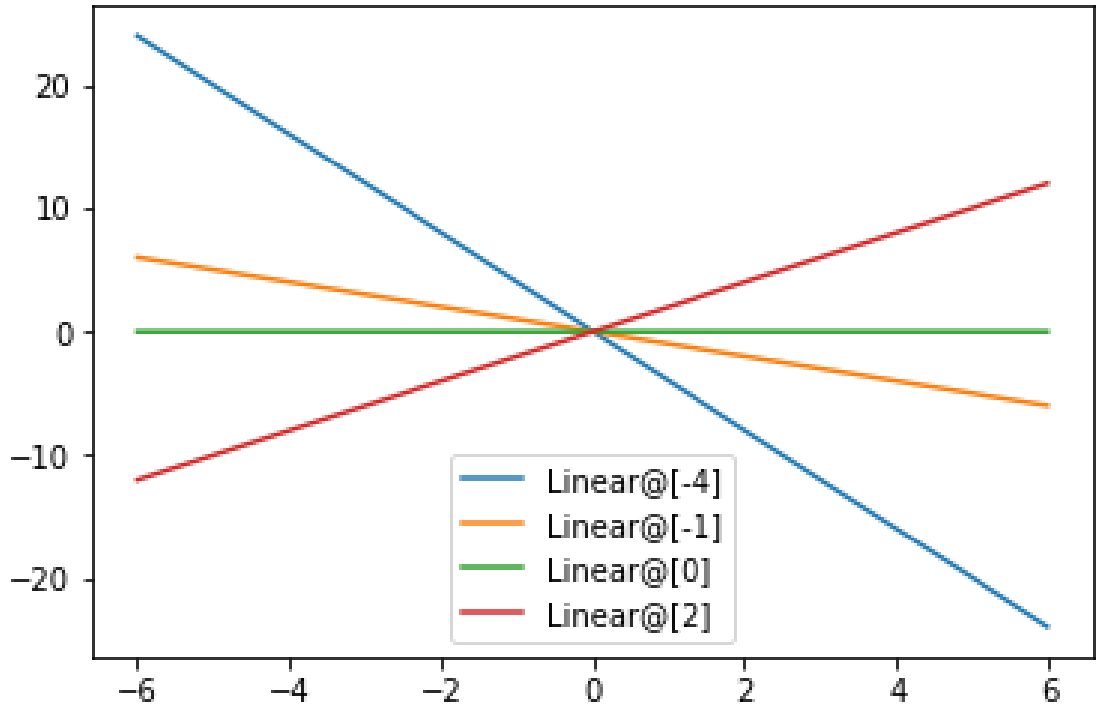
6.2.2

```
X1=np.array([-4,-1,0,2]).reshape(-1,1)
output = linear_kernel(X1, X1)
output
```

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

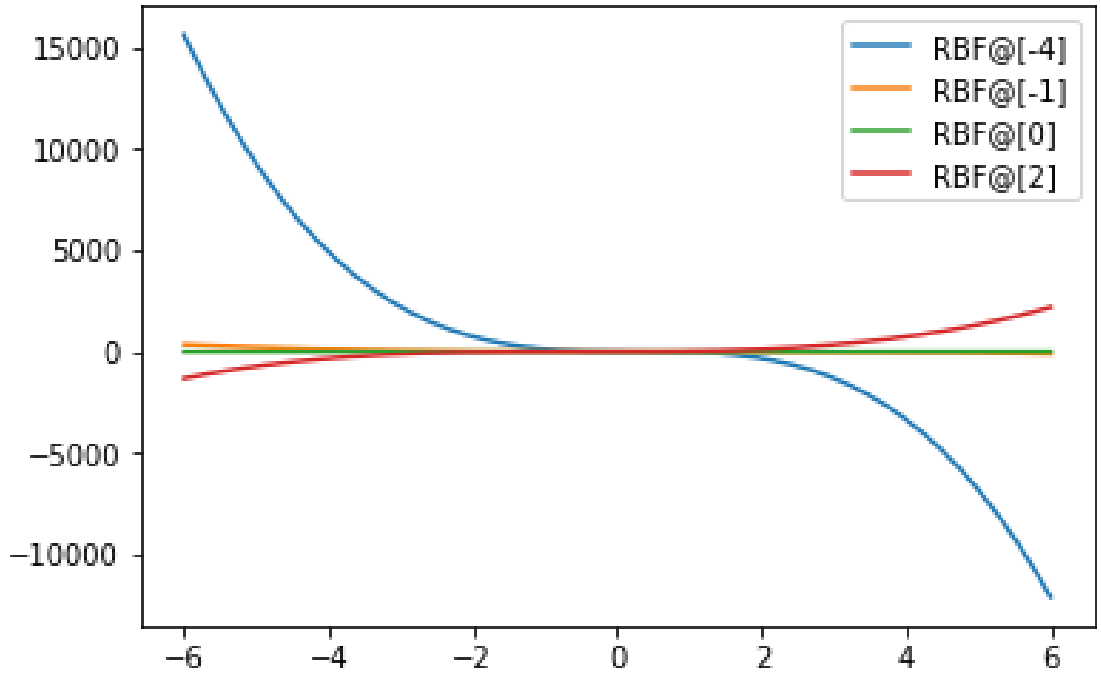
6.2.3

```
plot_step = .01
xpts = np.arange(-6.0, 6, plot_step).reshape(-1,1)
prototypes = np.array([-4,-1,0,2]).reshape(-1,1)

# Linear kernel
y = linear_kernel(prototypes, xpts)
for i in range(len(prototypes)):
    label = "Linear@"+str(prototypes[i,:])
    plt.plot(xpts, y[i,:], label=label)
plt.legend(loc = 'best')
plt.show()
```

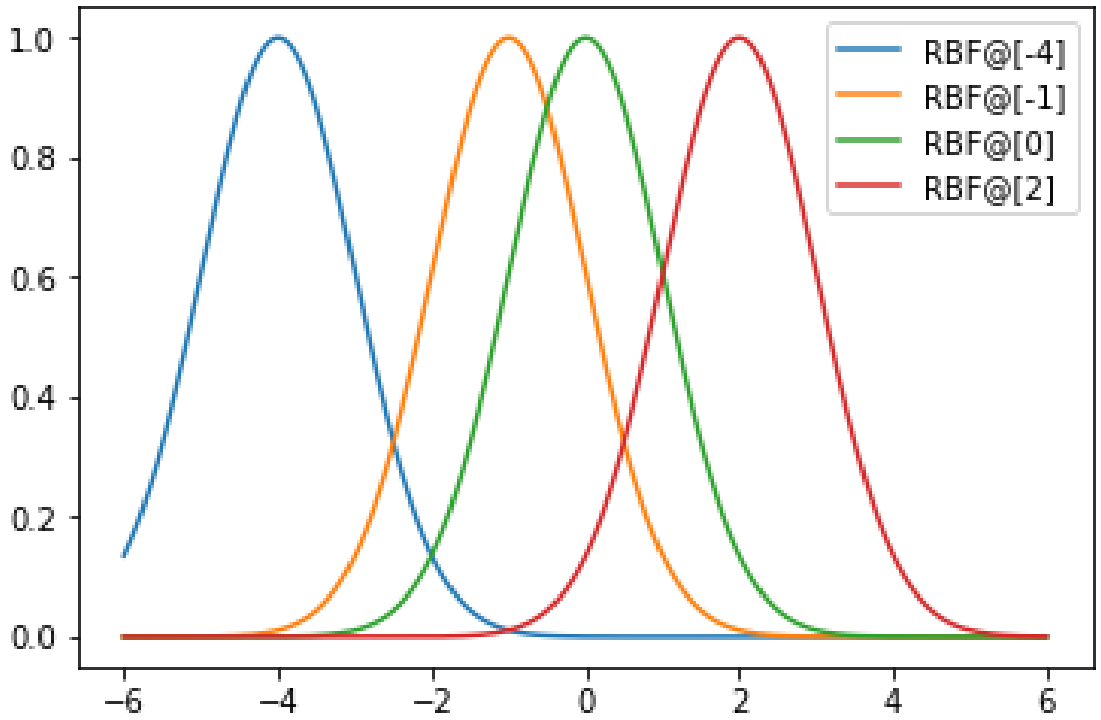



```
plot_step = .01
xpts = np.arange(-6.0, 6, plot_step).reshape(-1,1)
prototypes = np.array([-4,-1,0,2]).reshape(-1,1)
```

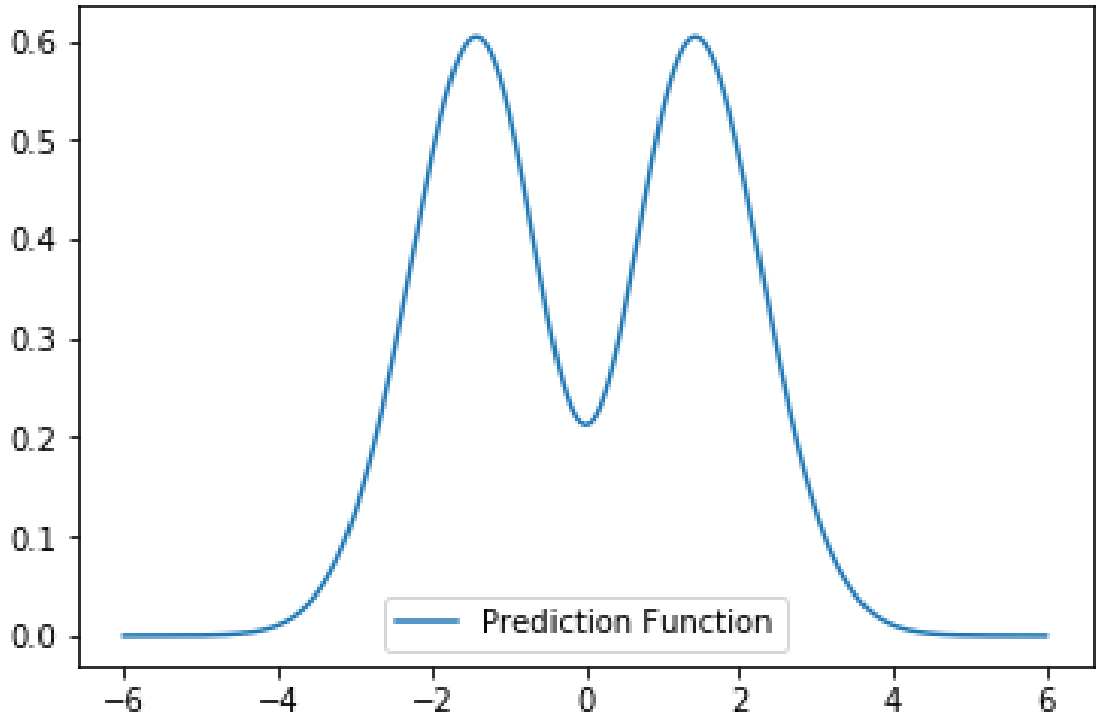
```
y = polynomial_kernel(prototypes, xpts, 1, 3)
for i in range(len(prototypes)):
    label = "RBF@"+str(prototypes[i,:])
    plt.plot(xpts, y[i,:], label=label)
plt.legend(loc = 'best')
plt.show()
```



```
plot_step = .01
xpts = np.arange(-6.0, 6, plot_step).reshape(-1,1)
prototypes = np.array([-4,-1,0,2]).reshape(-1,1)

y = RBF_kernel(prototypes, xpts,1)
for i in range(len(prototypes)):
    label = "RBF@"+str(prototypes[i,:])
    plt.plot(xpts, y[i,:], label=label)
plt.legend(loc = 'best')
plt.show()
```



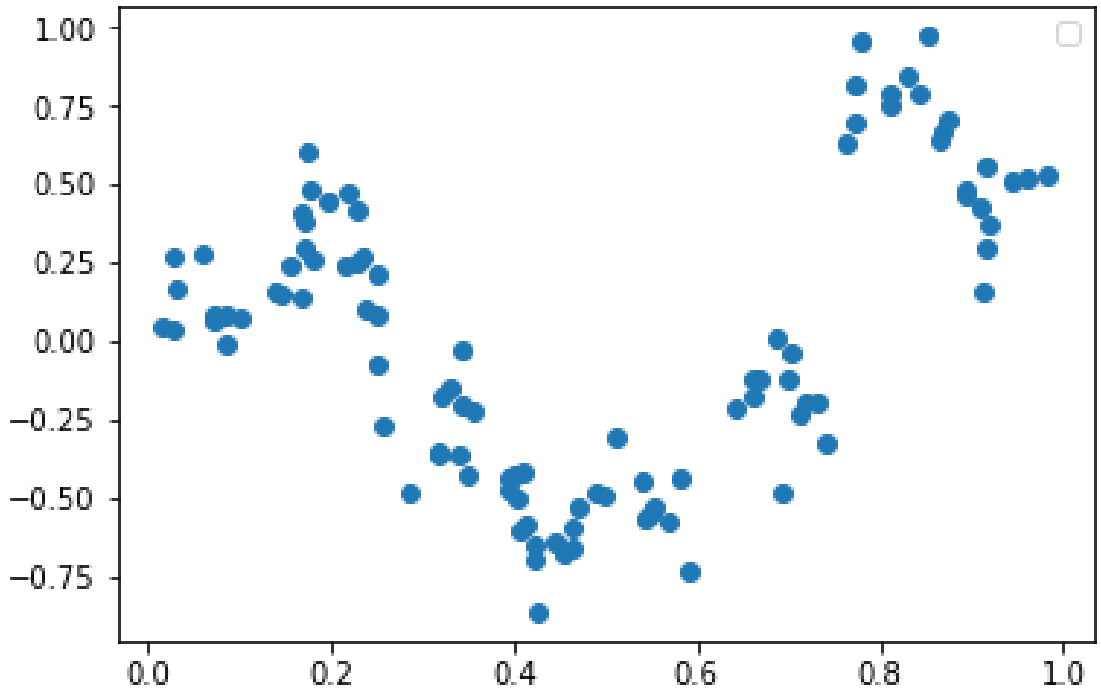

```
class Kernel_Machine(object):
    def __init__(self, kernel, prototype_points, weights):
        self.kernel = kernel
        self.prototype_points = prototype_points
        self.weights = weights

    def predict(self, X):

        return np.dot(self.weights.T, self.kernel(self.prototype_points,
                                                  X)).reshape(-1,1)

prototype_p = np.array([-1, 0, 1]).reshape(-1,1)
weight = np.array([1, -1, 1]).reshape(-1,1)
k = functools.partial(RBF_kernel,sigma=1)
RBF_K = Kernel_Machine(k,prototype_p,weight)

preds = RBF_K.predict(xpts)

plt.plot(xpts, preds, label="Prediction␣Function")
plt.legend(loc = 'best')
plt.show()
```



6.3 Kernel Ridge Regression

6.3.1

```
data_train,data_test = np.loadtxt("krr-train.txt"),np.loadtxt("krr-test.txt")
x_train, y_train = data_train[:,0].reshape(-1,1),data_train[:,1].reshape(-1,1)
x_test, y_test = data_test[:,0].reshape(-1,1),data_test[:,1].reshape(-1,1)
plt.plot(x_train, y_train, 'o')
plt.legend(loc = 'best')
plt.show()
```

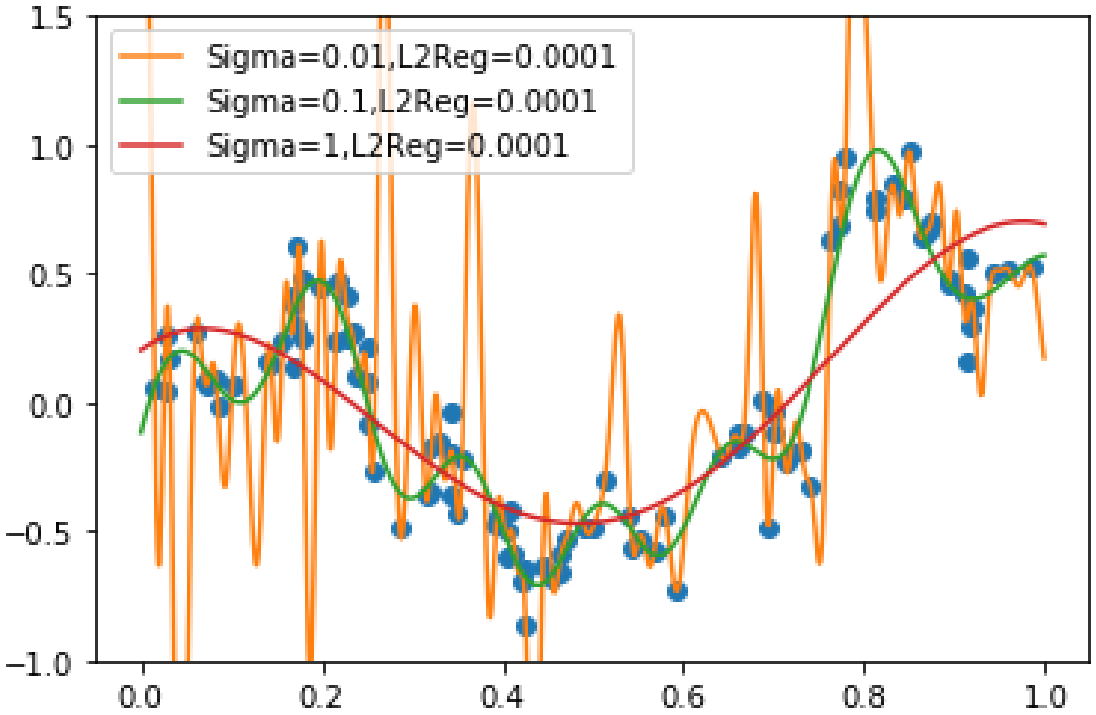


6.3.2

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

6.3.3

```
plot_step = .001
xpts = np.arange(0 , 1, plot_step).reshape(-1,1)
plt.plot(x_train,y_train,'o')
l2reg = 0.0001
for sigma in [.01,.1,1]:
    k = functools.partial(RBF_kernel, sigma=sigma)
    f = train_kernel_ridge_regression(x_train, y_train, k, l2reg=l2reg)
    label = "Sigma="+str(sigma)+",L2Reg="+str(l2reg)
    plt.plot(xpts, f.predict(xpts), label=label)
plt.legend(loc = 'best')
plt.ylim(-1,1.5)
plt.show()
```

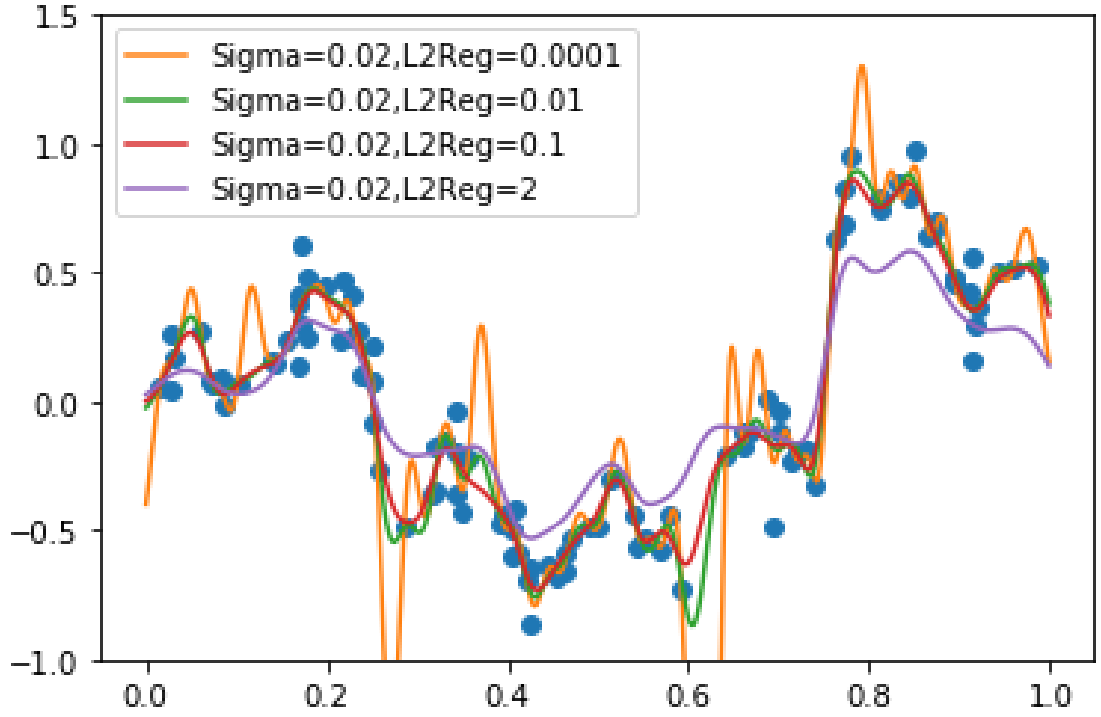


Sigma=0.01 is the most over fitting. Sigma=1 is less over fitting.

6.3.4

```
plot_step = .001
xpts = np.arange(0 , 1, plot_step).reshape(-1,1)
plt.plot(x_train,y_train,'o')
sigma= .02
for l2reg in [.0001,.01,.1,2]:
    k = functools.partial(RBF_kernel, sigma=sigma)
    f = train_kernel_ridge_regression(x_train, y_train, k, l2reg=l2reg)
    label = "Sigma="+str(sigma)+",L2Reg="+str(l2reg)
```

```
    plt.plot(xpts, f.predict(xpts), label=label)
plt.legend(loc = 'best')
plt.ylim(-1,1.5)
plt.show()
```



When $\lambda \to \infty$, the alpha is compressed to zero, which means the predicted function becomes flatter, y $\to$ 0.

6.3.5

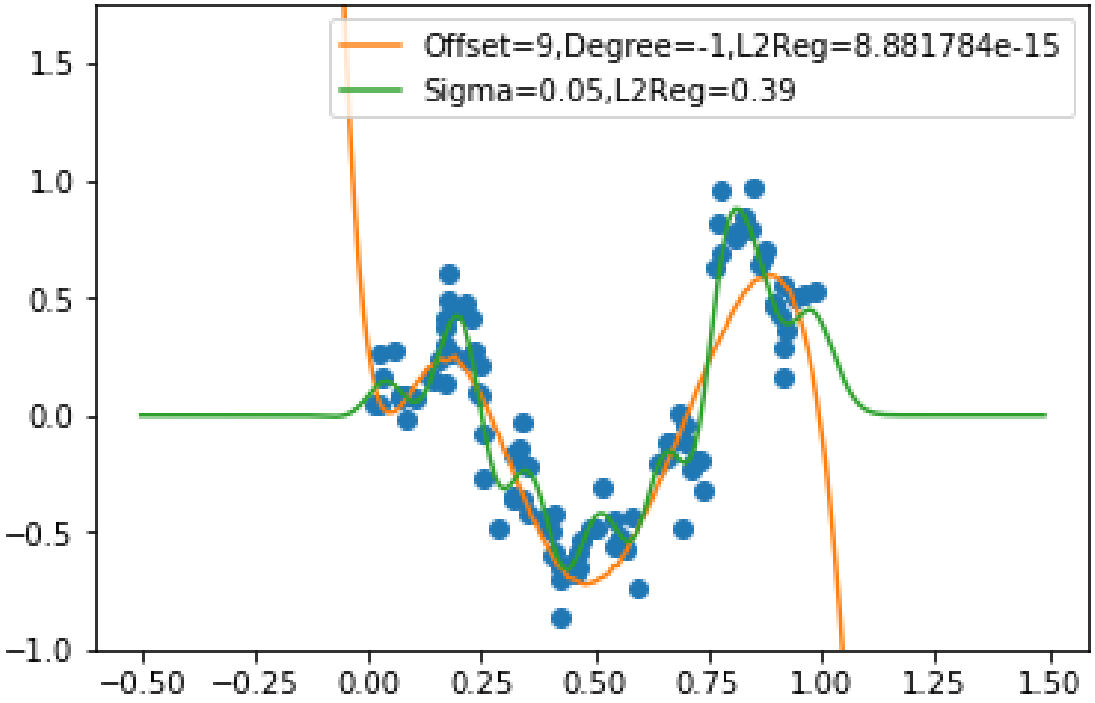
```
param_grid = [{'kernel': ['RBF'],'sigma':[.05,1,2], 'l2reg': np.arange(-0.4,0.4,0.01)}
              {'kernel':['polynomial'],'offset':np.arange(-5,20,1),
                                'degree':np.arange(-5,20,1),'l2reg':
                                        np.arange(-1,1,0.01)},#'l2reg':[10, .1, .01]
              {'kernel':['linear'],'l2reg': np.arange(-1,10,0.1)}]
kernel_ridge_regression_estimator = KernelRidgeRegression()
grid = GridSearchCV(kernel_ridge_regression_estimator,
                    param_grid,
                    cv = predefined_split,
                    scoring = make_scorer(mean_squared_error,greater_is_better = False
                   )
grid.fit(np.vstack((x_train,x_test)),np.vstack((y_train,y_test)))

pd.set_option('display.max_rows', 100)
df = pd.DataFrame(grid.cv_results_)
# Flip sign of score back, because GridSearchCV likes to maximize,
# so it flips the sign of the score if "greater_is_better=FALSE"
df['mean_test_score'] = -df['mean_test_score']
df['mean_train_score'] = -df['mean_train_score']
cols_to_keep = ["param_degree", "param_kernel","param_l2reg" ,"param_offset",
         "param_sigma",
        "mean_test_score","mean_train_score"]
df_toshow = df[cols_to_keep].fillna('-')
df_toshow.sort_values(by=["mean_test_score"])
```

	param_kernel	param_degree	param_offset	param_sigma	param_l2reg	mean_test_score	mean_train_score
0	Best RBF	-	-	0.05	3.900000e-01	0.013873	0.014243
1	Nearby RBF	-	-	0.05	3.600000e-01	0.013876	0.014056
2	Best Polynomial	-1	9	-	8.881784e-15	0.029206	0.045620
3	Nearby Polynomial	-1	17	-	8.881784e-15	0.029265	0.041863
4	Best Linear	-	-	-	4.100000e+00	0.164509	0.206561
5	Nearby Linear	-	-	-	4.000000e+00	0.164515	0.206529

6.3.6

```
## Plot the best polynomial and RBF fits you found
plot_step = .01
xpts = np.arange(-.5 , 1.5, plot_step).reshape(-1,1)
plt.plot(x_train,y_train,'o')
#Plot best polynomial fit
offset= 9
degree = -1
l2reg = 8.881784e-15
k = functools.partial(polynomial_kernel, offset=offset, degree=degree)
f = train_kernel_ridge_regression(x_train, y_train, k, l2reg=l2reg)
label = "Offset="+str(offset)+",Degree="+str(degree)+",L2Reg="+str(l2reg)
plt.plot(xpts, f.predict(xpts), label=label)
#Plot best RBF fit
sigma = 0.05
l2reg= 3.900000e-01
k = functools.partial(RBF_kernel, sigma=sigma)
f = train_kernel_ridge_regression(x_train, y_train, k, l2reg=l2reg)
label = "Sigma="+str(sigma)+",L2Reg="+str(l2reg)
plt.plot(xpts, f.predict(xpts), label=label)
plt.legend(loc = 'best')
```

```
plt.ylim(-1,1.75)
plt.show()
```



The RBF kernel predicted function performs better than the polynomial one. Both functions work well int the area $x \in [0,1]$. Outside of this area, RBF kernel predicted function $\to \infty$, while the polynomial one $\to +\infty or -\infty$

6.3.7

6.3.7.

according HW1, we have Bayes decision function

$$f^* = E(y|x).$$

$\Rightarrow$ Bayes decision function here

is $f^* = E(f(x)+\varepsilon|x)$

Bayes risk:

$$E(l(\hat{y},y)) = E\left(E(f(x)+\varepsilon|x)^2 - 2E(f(x)+\varepsilon|x)\cdot y + y^2\right)$$

$$= E\left((f(x)+\varepsilon)^2 - 2E(f(x)+\varepsilon)\cdot E(y) + E(y^2)\right)$$

$$= E(f(x)^2) + 2E(f(x))\cdot E(\varepsilon) + E(\varepsilon^2) - 2E(f(x))\cdot E(y) - 2E(\varepsilon)\cdot E(y) + E(y^2)$$

$$= E(f(x)^2) + E(y^2) - 2E(f(x))\cdot E(y) + E(\varepsilon^2)$$

$$\boxed{E(f(x)) = E(y)} \quad = E(\varepsilon^2)$$

$\because Var(\varepsilon) = E(\varepsilon^2) - E^2(\varepsilon) = 0.1^2$

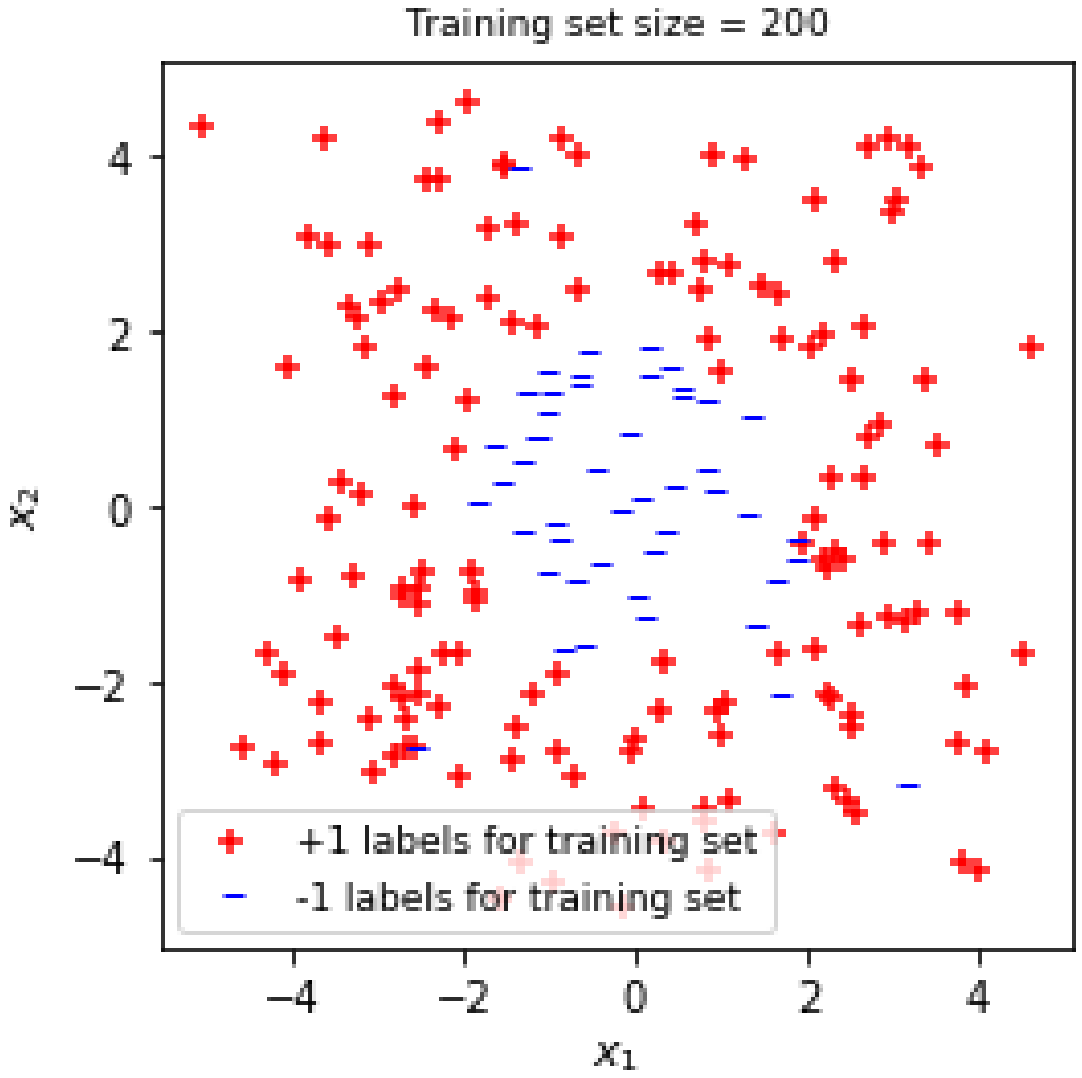
$\Rightarrow E(\varepsilon^2) = 0.1^2$

$\Rightarrow$ Bayes risk is 0.1^2

6.4 Kernelized Support Vector Machines with Kernelized Pegasos

6.4.1

```
# Load and plot the SVM data
#load the training and test sets
data_train,data_test = np.loadtxt("svm-train.txt"),np.loadtxt("svm-test.txt")
x_train, y_train = data_train[:,0:2], data_train[:,2].reshape(-1,1)
x_test, y_test = data_test[:,0:2], data_test[:,2].reshape(-1,1)

#determine predictions for the training set
yplus = np.ma.masked_where(y_train[:,0]<=0, y_train[:,0])
xplus = x_train[~np.array(yplus.mask)]
yminus = np.ma.masked_where(y_train[:,0]>0, y_train[:,0])
xminus = x_train[~np.array(yminus.mask)]

#plot the predictions for the training set
figsize = plt.figaspect(1)
f, (ax) = plt.subplots(1, 1, figsize=figsize)

pluses = ax.scatter (xplus[:,0], xplus[:,1], marker='+', c='r', label = '+1 labels fo
minuses = ax.scatter (xminus[:,0], xminus[:,1], marker=r'$-$', c='b', label = '-1 lab

ax.set_ylabel(r"$x_2$", fontsize=11)
ax.set_xlabel(r"$x_1$", fontsize=11)
ax.set_title('Training set size = %s'% len(data_train), fontsize=9)
ax.axis('tight')
ax.legend(handles=[pluses, minuses], fontsize=9)
plt.show()
```



This is quadratically seperable. RBF will achieve better seperations

7 Representer Theorem

7.1

$\because \|m_0\|^2 + \|x - m_0\|^2 = \|x\|^2$

$\therefore \|m_0\|^2 \leq \|x\|^2$ because $\|x - m_0\|^2 \geq 0$

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

$\Rightarrow x = m_0$

then we have $\|m_0\|^2 = \|x\|^2$

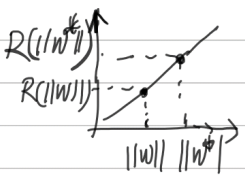
so we have $\|m_0\| = \|x\|$ only when $m_0 = x$

7.2. let $\theta = \text{Span}(\psi(x_1), \cdots \psi(x_n))$, $w = \text{Proj}_\theta w^*$

for some minimizer w^* of $J(w)$.

for Loss term, $\langle w, \psi(x_i) \rangle = \langle w^*, \psi(x_i) \rangle$ according the Projection Theorem.

① if $\|w\| < \|w^*\|$, and R term is strictly increasing,



$\Rightarrow R(\|w^*\|) > R(\|w\|)$,

then w^* is not the minimizer.

② if $\|w\| = \|w^*\|$, w^* is the minimizer,

from 7.1, we have $w^* = w = \sum_{i=1}^{n} \alpha_i \psi(x_i)$ form

$\Rightarrow$ if R is strictly increasing, that all minimizer has $\sum_{i=1}^{n} \alpha_i \psi(x_i)$

7.3.

R, L are convex $\Rightarrow \langle w, \phi(x_i) \rangle = \phi(x_i)^T w$ is convex by w.

$\therefore$ $L(\langle w_1, \phi(x_1) \rangle \cdots \langle w_n, \phi(x_n) \rangle)$ is convex too.

$\|w\| = \sqrt{w^T w}$ is convex

$\Rightarrow$ $R(\|w\|)$ is convex,

$\Rightarrow$ $J(w) = R(\|w\|) + L(\langle w, \phi(x_1) \rangle, \cdots \langle w_n, \phi(x_n) \rangle)$ is convex

8 Ivanov and Tikhonov Regularization

8.1. assume $r = \Omega(f^*)$

suppose f^* is not the optimum in (2), there is $\hat{f}$

$\Rightarrow \Omega(\hat{f}) \le \Omega(f^*)$, $\phi(\hat{f}) \le \phi(f^*)$. (3)

$\because$ (1), (2) have same item $\arg\min_{f \in F} \phi(f)$,

from (3) $\Rightarrow [\phi(\hat{f}) + \lambda\Omega(\hat{f})] < [\phi(f^*) + \lambda\Omega(f^*)]$

contradicts (1).

$\Rightarrow f^*$ is also an Ivanov solution.

8.2.1

$$L(w, \lambda) = \phi(w) + \lambda(\Omega(w) - r)$$

8.2.2

We need to maximum $L(w, \lambda)$,

$\Rightarrow \max(\min(\phi(w) + \lambda(\Omega(w) - r))$

$= g(\lambda) = \min(\phi(w) + \lambda(\Omega(w) - r))$