2.1 & 2.2

2. (1) $M = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$.

$M^T M = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. $\therefore$ M is orthogonal

but $M^T \neq M$. $\therefore$ M is orthogonal but not symmetric

(2) for any real, orthogonal matrix M is psd and $x \in R^n$.

$x^T M x \geq 0$.

by spectral theorem, $M \in R^{n \times n}$ can be diagonalized as $M = Q \Sigma Q^T$

$\Rightarrow \Sigma = Q^T M Q$.

$Q \in R^{n \times n}$ is an orthogonal matrix whose coloms are a set of orthonormal eigenvectors of M.

$\Rightarrow Q = \begin{pmatrix} | & & | \\ q_1 & \cdots & q_n \\ | & & | \end{pmatrix}$.

$\Rightarrow Q^T M Q = \begin{pmatrix} q_1^T M q_1 & \cdots & q_1^T M q_n \\ \vdots & & \vdots \\ q_n^T M q_1 & & q_n^T M q_n \end{pmatrix}$.

eigenvalue of psd matrix M is $\mathrm{diag}(Q^T M Q) = (q_1^T M q_1, \cdots q_n^T M q_n)$

and for any $q^T M q \geq 0$.

$\therefore$ the eigenvalues of M are all non-negative.

2.3

(3). If $M = BB^T$ for any $x \in R^n$.

$$x^T M x = x^T B B^T x$$
$$= (B^T x)^T B^T x = \|B^T x\|_2^2 \geq 0.$$

$\therefore$ Symmetric matrix M is psd.

If M is psd.

Use spetral theorem,

M can be rewritten as $Q \Sigma Q^T$

$$M = Q \Sigma Q^T$$
$$= Q \cdot \Sigma^{\frac{1}{2}} \Sigma^{\frac{1}{2}} Q^T.$$

set $B = Q \Sigma^{\frac{1}{2}}$.

$$M = BB^T.$$

3.1 & 3.2

3. ① for the real symmetric matrix M,

we have $\Sigma = Q^T M Q$. Q is the matrix of M's eigenvectors.

$$Q = \begin{pmatrix} | & & | \\ q_1 & \cdots & q_n \\ | & & | \end{pmatrix}$$

$$Q^T M Q = \begin{pmatrix} q_1^T m q_1 & \cdots & q_1^T m q_n \\ \vdots & & \vdots \\ q_n^T m q_1 & \cdots & q_n^T m q_n \end{pmatrix}$$

$$= \begin{pmatrix} q_1^T m q_1 & 0 & \cdots & 0 \\ 0 & q_2^T m q_2 & & \vdots \\ 0 & 0 & & \\ \vdots & \vdots & & \\ 0 & & & q_n^T m q_n \end{pmatrix}.$$

if M is postive definite matrix for all $i = 1 \cdots n$

$q_i^T m q_i > 0$. $\Rightarrow$ all eigenvalues will be greater 0.

② Assume it is correct. $M^{-1} = Q \Sigma^{-1} Q^T$.

$$\begin{aligned} M M^{-1} &= Q \Sigma Q^T Q \Sigma^{-1} Q^T \\ &= Q \Sigma \Sigma^{-1} Q^T \\ &= Q Q^T \\ &= 1. \end{aligned}$$

$\therefore Q \Sigma^{-1} Q^T$ is the inverse of M.

3.3 & 3.4

③ for any $x \in \mathbb{R}^n$ $(x \neq 0)$

$x^T(M+\lambda I)x = x^T M x + x^T \lambda I x$

Since M is psd and $x \neq 0$. $x^T M x \geq 0$ $x^T x > 0$.

when $\lambda > 0$ $x^T(M+\lambda I)x > 0$.

$\therefore x^T(M+\lambda I)x > 0$.

$\therefore M+\lambda I$ is psd.

$M+\lambda I = Q(\Sigma + \lambda I)Q^T$ $\therefore Q(\Sigma+\lambda I)^{-1}Q^T$ is the inverse

④ $x^T(M+N)x$

$= x^T M x + x^T N x$

$x^T M x \geq 0$. (by positive semidefinite)

$x^T N x > 0$. (by positive definite)

$\therefore x^T(M+N)x > 0$. $\therefore x^T(M+N)x$ is positive definite and invertible

Q4 – Optional

5.1 & 5.2 & 5.3

3① $J(w) = (xw-y)^T(xw-y) + \lambda w^T w$,

$\partial J(w) = 2x^Txw - 2x^Ty + 2\lambda w$.

set $\partial J(w) = 0$

$2x^Txw - 2x^Ty + 2\lambda w = 0$.

$(x^Tx + \lambda I)w = x^Ty$.

$w = (x^Tx + \lambda I)^{-1} x^Ty$.

for any R^n and $\lambda > 0$

$a^T(x^Tx + \lambda I)a = \underbrace{a^T(x^Tx)a}_{PSD} + \underbrace{a^T\lambda I a}_{\text{positive definite}}$

By 3④ proof, $a^T(x^Tx+\lambda I)a$ is invertible.

② $w = \frac{1}{\lambda}(x^Ty - x^Txw)$.

$= x^T \frac{1}{\lambda}(y - xw)$.

$\therefore w = x^T\alpha$ and $\alpha = \frac{1}{\lambda}(y - xw)$.

③ $w = x^T\alpha$

$= \begin{pmatrix} | & & | \\ x_1 & \cdots & x_n \\ | & & | \end{pmatrix}\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} = a_1x_1 + \cdots a_nx_n$.

$\therefore$ we can say w is in the span of data.

5.4 & 5.5 & 5.6

④ $\alpha = \frac{1}{\lambda}(y - Xw)$

$\lambda I \alpha = y - Xw$

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

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

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

⑤ $Xw = XX^T \alpha$

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

⑥ $w^* = X^T \alpha^*$

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

since $k_x = \begin{pmatrix} x^T x_1 \\ \vdots \\ x^T x_n \end{pmatrix}$. $\therefore f(x) = k_x^T (\lambda I + XX^T)^{-1} y$

Q6 – Optional

7.1 & 7.2

7. ① we know

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

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

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

$$\therefore y_j \langle w^{(t)}, x_j \rangle = y_j k_j \alpha^{(t)}$$

② in the case of no margin violation

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

$$= (1 - \eta^{(t)} \lambda) \left(\sum_{i=1}^{n} \alpha_i^{(t)} x_i \right)$$

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

$$= \sum_{i=1}^{n} \underbrace{\alpha_i^{(t+1)}} x_i$$

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

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

7.3

1 ② $w^{(t+1)} = (1-y^{(t)}\lambda)\, w^{(t)} + \eta_t y_j x_j$

$$= (1-y^t\lambda)\left(\sum_{i=1}^{n} \alpha_i^{(t)} x_i\right) + \eta_t\, y_j\, x_j$$

if $i=j$

$$= (1-y^t\lambda)\left(\sum_{i=1}^{n} \alpha_i^{(t)} x_i\right) + \eta_t y_i x_i$$

$$= \left((1-y^t\lambda)\sum_{i=1}^{n} \alpha_i^{(t)} + \eta_t y_i\right) x_i$$

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

$$\therefore \alpha^{(t+1)} = (1-y^{(t)}\lambda)\alpha^{(t)} + \eta_t y_i$$

else:

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

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

Since $i \neq j$

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

The algorithm:

$\alpha^{(1)} = (0, 0, \cdots 0) \in R^n$

$t = 0$

repeat

$t = t+1$

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

randomly choose j in $1 \cdots n$

if $y_j x_j \alpha^{(t)} < 1$

if $i \neq j$

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

else

$\alpha^{(t+1)} = (1-y^{(t)}\lambda)\alpha^{(t)} + \eta_t y_i$

else

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

until bored

return

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

7.4 – Optional

8.2.1

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import scipy.spatial
import functools

%matplotlib inline
```

```
### Kernel function generators
def linear_kernel(W, X):
    """
    Computes the linear kernel between two sets of vectors.
    Args:
        W, X - two matrices of dimensions n1xd and n2xd
    Returns:
        matrix of size n1xn2, with w_i^T x_j in position i,j
    """
    return np.dot(W,np.transpose(X))

def RBF_kernel(W,X,sigma):
    """
    Computes the RBF kernel between two sets of vectors
    Args:
        W, X - two matrices of dimensions n1xd and n2xd
        sigma - the bandwidth (i.e. standard deviation) for the RBF/Gaussian kernel
    Returns:
        matrix of size n1xn2, with exp(-||w_i-x_j||^2/(2 sigma^2)) in position i,j
    """
    kernel = scipy.spatial.distance.cdist(W,X,'sqeuclidean')
    kernel = (-kernel)/(2*(sigma**2))
    return np.exp(kernel)



def polynomial_kernel(W, X, offset, degree):
    """
    Computes the inhomogeneous polynomial kernel between two sets of vectors
    Args:
        W, X - two matrices of dimensions n1xd and n2xd
        offset, degree - two parameters for the kernel
    Returns:
        matrix of size n1xn2, with (offset + <w_i,x_j>)^degree in position i,j
    """
    kernel = np.dot(W,np.transpose(X))
    kernel = offset + kernel

    return (kernel)**degree
```

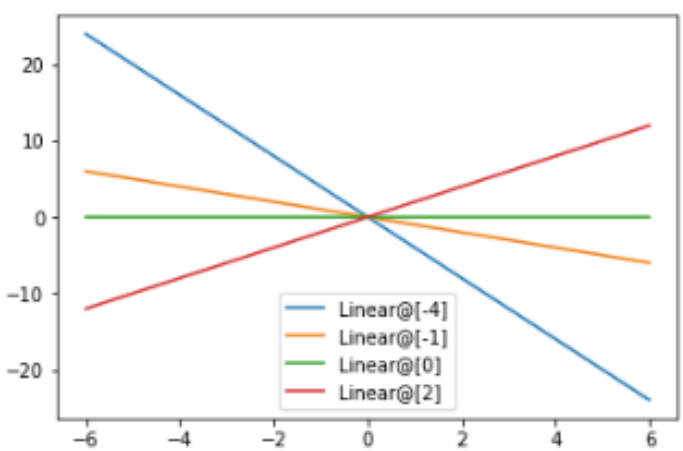
8.2.2

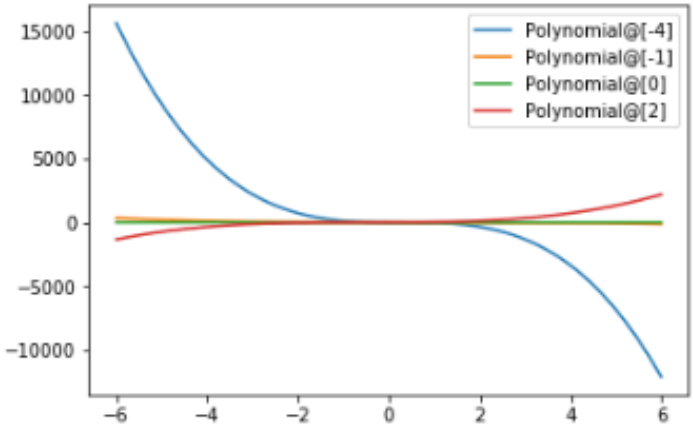
```
x0 = np.array([[-4],[-1],[0],[2]])
linear_kernel(x0,x0)
```

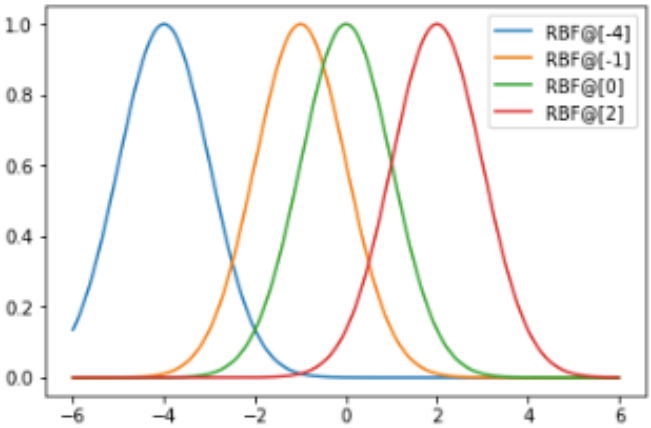
```
array([[16,  4,  0, -8],
       [ 4,  1,  0, -2],
       [ 0,  0,  0,  0],
       [-8, -2,  0,  4]])
```

8.2.3

```
# PLot kernel machine functions

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()

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

#RBF
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()
```




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

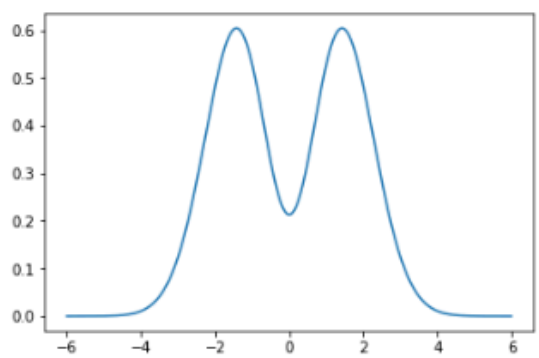

RBF@[-4]
RBF@[-1]
RBF@[0]
RBF@[2]
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6

8.2.4

```
class Kernel_Machine(object):
    def __init__(self, kernel, prototype_points, weights):
        """
        Args:
            kernel(W,X) - a function return the cross-kernel matrix between rows of W and rows of X for kernel 
            prototype_points - an Rxd matrix with rows mu_1,...,mu_R
            weights - a vector of length R
        """

        self.kernel = kernel
        self.prototype_points = prototype_points
        self.weights = weights

    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 vecto
                Sum_{i=1}^R w_i k(x_j, mu_i)
        """

        self.kernlized_points = self.kernel(self.prototype_points,X)
        return  np.dot(self.kernlized_points.T,self.weights)
```

```
prototypes = np.array([[-1],[0],[1]])
weight = np.array([[1],[-1],[1]])

k = functools.partial(RBF_kernel, sigma = 1)
y = Kernel_Machine(k, prototypes, weight).predict(xpts)
plt.plot(xpts, y)
```

```
[<matplotlib.lines.Line2D at 0x11688d198>]
```

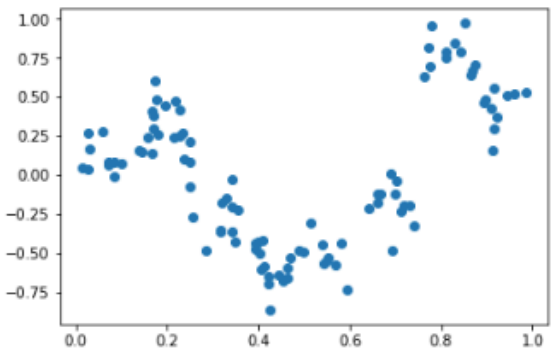


8.3.1

```
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")
```

```
[<matplotlib.lines.Line2D at 0x116977dd8>]
```

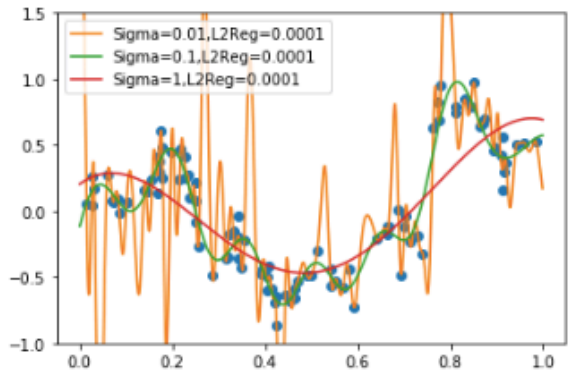


8.3.2

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

    return Kernel_Machine(kernel, X, alpha)
```

8.3.3

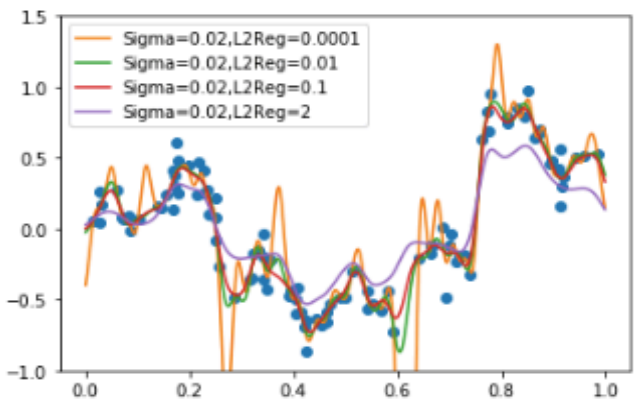
```
plot_step = .001
xpts = np.arange(0 , 1, plot_step).reshape(-1,1)
plt.plot(x_train,y_train,'o')
l2reg = 0.0001
for sigma in [.01,.1,1]:
    k = functools.partial(RBF_kernel, sigma=sigma)
    f = train_kernel_ridge_regression(x_train, y_train, k, l2reg=l2reg)
    label = "Sigma="+str(sigma)+",L2Reg="+str(l2reg)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



It seems that when sigma equals to 1 is much more generalized. When sigma is smaller, it looks much more likely to over fit

8.3.4

```
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 is 0.0001, the line looks very close to data points. I think when lambda is getting close to infinity, it will be an almost straight line that is over-generalized.

8.3.5

```
from sklearn.base import BaseEstimator, RegressorMixin, ClassifierMixin

class KernelRidgeRegression(BaseEstimator, RegressorMixin):
    """sklearn wrapper for our kernel ridge regression"""

    def __init__(self, kernel="RBF", sigma=1, degree=2, offset=1, l2reg=1):
        self.kernel = kernel
        self.sigma = sigma
        self.degree = degree
        self.offset = offset
        self.l2reg = l2reg

    def fit(self, X, y=None):
        """
        This should fit classifier. All the "work" should be done here.
        """
        if (self.kernel == "linear"):
            self.k = linear_kernel
        elif (self.kernel == "RBF"):
            self.k = functools.partial(RBF_kernel, sigma=self.sigma)
        elif (self.kernel == "polynomial"):
            self.k = functools.partial(polynomial_kernel, offset=self.offset, degree=self.degree)
        else:
            raise ValueError('Unrecognized kernel type requested.')

        self.kernel_machine_ = train_kernel_ridge_regression(X, y, self.k, self.l2reg)

        return self

    def predict(self, X, y=None):
        try:
            getattr(self, "kernel_machine_")
        except AttributeError:
            raise RuntimeError("You must train classifer before predicting data!")

        return(self.kernel_machine_.predict(X))

    def score(self, X, y=None):
        # get the average square error
        return((self.predict(X)-y).mean())
```

```
from sklearn.model_selection import GridSearchCV,PredefinedSplit
from sklearn.model_selection import ParameterGrid
from sklearn.metrics import mean_squared_error,make_scorer
import pandas as pd

test_fold = [-1]*len(x_train) + [0]*len(x_test)   #0 corresponds to test, -1 to train
predefined_split = PredefinedSplit(test_fold=test_fold)
```

Broad search:

```
param_grid = [{'kernel': ['RBF'],'sigma':np.logspace(-10, 2, 100), 'l2reg': np.logspace(-5, 5, 100)},
              {'kernel':['polynomial'],'offset':np.linspace(-5, 5, 100), 'degree':np.arange(2,15,1),'l2reg':np.logspa
ce(-10, 2, 100) },
              {'kernel':['linear'],'l2reg': np.logspace(-5, 5, 100)}]
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)
                  # n_jobs = -1  #should allow parallelism, but crashes Python on my machine
                   )
grid.fit(np.vstack((x_train,x_test)),np.vstack((y_train,y_test)))



pd.set_option('display.max_rows', 20)
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"])


#find the best
best_linear = df_toshow.ix[df_toshow[df_toshow.param_kernel == "linear"]["mean_test_score"].argmin()]
best_rbf = df_toshow.ix[df_toshow[df_toshow.param_kernel == "RBF"]["mean_test_score"].argmin()]
best_polynomial = df_toshow.ix[df_toshow[df_toshow.param_kernel == "polynomial"]["mean_test_score"].argmin()]

pd.concat([best_linear, best_rbf, best_polynomial], axis=1).T
```

	param_degree	param_kernel	param_l2reg	param_offset	param_sigma	mean_test_score	mean_train_score
140055	-	linear	3.59381	-	-	0.16451	0.206552
4472	-	RBF	0.278256	-	0.053367	0.0138428	0.0141175
120043	13	polynomial	1e-10	-0.656566	-	0.0216871	0.0254216

Narrowing Down:

```
param_grid = [{'kernel': ['RBF'],'sigma':np.logspace(-2, -1, 100), 'l2reg': np.logspace(-1,0, 100)},
              {'kernel':['polynomial'],'offset':np.linspace(-0.8, -.5, 100), 'degree':np.arange(12,14,1),'l2reg':np.l
ogspace(-11, -9, 100) },
              {'kernel':['linear'],'l2reg': np.logspace(0, 1, 100)}]


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)
                  # n_jobs = -1  #should allow parallelism, but crashes Python on my machine
                   )
grid.fit(np.vstack((x_train,x_test)),np.vstack((y_train,y_test)))


pd.set_option('display.max_rows', 20)
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"])



best_linear = df_toshow.ix[df_toshow[df_toshow.param_kernel == "linear"]["mean_test_score"].argmin()]
best_rbf = df_toshow.ix[df_toshow[df_toshow.param_kernel == "RBF"]["mean_test_score"].argmin()]
best_polynomial = df_toshow.ix[df_toshow[df_toshow.param_kernel == "polynomial"]["mean_test_score"].argmin()]

pd.concat([best_linear, best_rbf, best_polynomial], axis=1).T
```

	param_degree	param_kernel	param_l2reg	param_offset	param_sigma	mean_test_score	mean_train_score
30059	-	linear	3.94421	-	-	0.16451	0.206561
2477	-	RBF	0.174753	-	0.0599484	0.0138052	0.014419
25450	13	polynomial	1.23285e-10	-0.648485	-	0.0198574	0.0317414

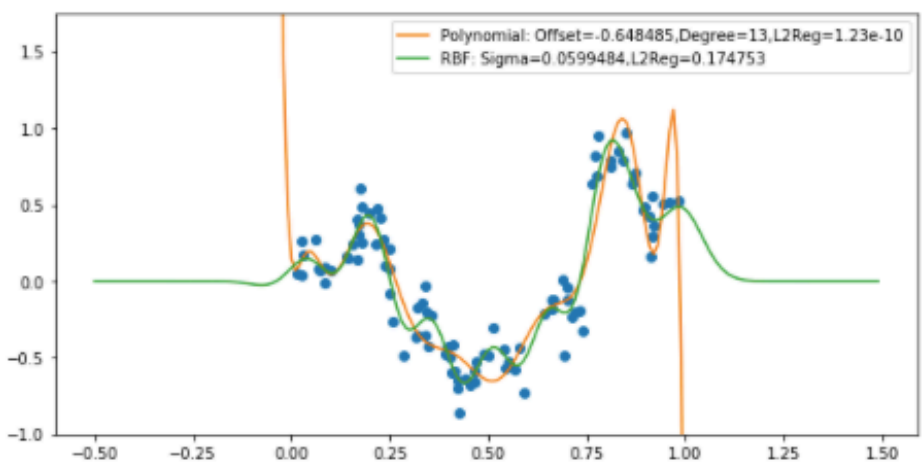
Validating the global Minimal

```
param_grid = [{'kernel': ['RBF'],'sigma':[0.059,0.0599484,0.06], 'l2reg': [0.17,0.174753,0.18]},
              {'kernel':['polynomial'],'offset':[-0.64,-0.65,-0.648485], 'degree':[12,13,14],'l2reg':[1e-10,1.23285e-
10,2e-10] },
              {'kernel':['linear'],'l2reg': [3.9,3.94421,4]}]
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)
                  # n_jobs = -1  #should allow parallelism, but crashes Python on my machine
                   )
grid.fit(np.vstack((x_train,x_test)),np.vstack((y_train,y_test)))


pd.set_option('display.max_rows', 39)
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_degree	param_kernel	param_l2reg	param_offset	param_sigma	mean_test_score	mean_train_sc
5	-	RBF	1.747530e-01	-	0.06	0.013805	0.014428
4	-	RBF	1.747530e-01	-	0.0599484	0.013805	0.014419
7	-	RBF	1.800000e-01	-	0.0599484	0.013805	0.014467
8	-	RBF	1.800000e-01	-	0.06	0.013806	0.014476
2	-	RBF	1.700000e-01	-	0.06	0.013806	0.014384
1	-	RBF	1.700000e-01	-	0.0599484	0.013806	0.014375
6	-	RBF	1.800000e-01	-	0.059	0.013810	0.014310
3	-	RBF	1.747530e-01	-	0.059	0.013815	0.014264
0	-	RBF	1.700000e-01	-	0.059	0.013820	0.014222
23	13	polynomial	1.232850e-10	-0.648485	-	0.019857	0.031742
22	13	polynomial	1.232850e-10	-0.65	-	0.019882	0.032243
21	13	polynomial	1.232850e-10	-0.64	-	0.020243	0.028963
19	13	polynomial	1.000000e-10	-0.65	-	0.021748	0.025479
20	13	polynomial	1.000000e-10	-0.648485	-	0.021805	0.025455
18	13	polynomial	1.000000e-10	-0.64	-	0.022347	0.025154
9	12	polynomial	1.000000e-10	-0.64	-	0.030011	0.023359
11	12	polynomial	1.000000e-10	-0.648485	-	0.030100	0.023362
10	12	polynomial	1.000000e-10	-0.65	-	0.030113	0.023362
12	12	polynomial	1.232850e-10	-0.64	-	0.030186	0.023378
14	12	polynomial	1.232850e-10	-0.648485	-	0.030302	0.023383
13	12	polynomial	1.232850e-10	-0.65	-	0.030319	0.023383
15	12	polynomial	2.000000e-10	-0.64	-	0.030861	0.023482
27	14	polynomial	1.000000e-10	-0.64	-	0.031020	0.023640
17	12	polynomial	2.000000e-10	-0.648485	-	0.031090	0.023499
16	12	polynomial	2.000000e-10	-0.65	-	0.031125	0.023501
29	14	polynomial	1.000000e-10	-0.648485	-	0.031449	0.023771
28	14	polynomial	1.000000e-10	-0.65	-	0.031544	0.023798
30	14	polynomial	1.232850e-10	-0.64	-	0.031987	0.023780
32	14	polynomial	1.232850e-10	-0.648485	-	0.032265	0.023869
31	14	polynomial	1.232850e-10	-0.65	-	0.032335	0.023890
35	14	polynomial	2.000000e-10	-0.648485	-	0.034565	0.024214
34	14	polynomial	2.000000e-10	-0.65	-	0.034577	0.024221
33	14	polynomial	2.000000e-10	-0.64	-	0.034606	0.024209
25	13	polynomial	2.000000e-10	-0.65	-	0.067943	0.040315
26	13	polynomial	2.000000e-10	-0.648485	-	0.072594	0.043424
24	13	polynomial	2.000000e-10	-0.64	-	0.149284	0.104629
36	-	linear	3.900000e+00	-	-	0.164510	0.206560
37	-	linear	3.944210e+00	-	-	0.164510	0.206561
38	-	linear	4.000000e+00	-	-	0.164510	0.206563

8.3.6

```
## Plot the best polynomial and RBF fits you found
plt.figure(figsize=(10,5))
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= -0.648485
degree = 13
l2reg = 1.23E-10
k = functools.partial(polynomial_kernel, offset=offset, degree=degree)
f = train_kernel_ridge_regression(x_train, y_train, k, l2reg=l2reg)
label = "Polynomial: Offset="+str(offset)+",Degree="+str(degree)+",L2Reg="+str(l2reg)
plt.plot(xpts, f.predict(xpts), label=label)
#Plot best RBF fit
sigma = 0.0599484
l2reg= 0.174753
k = functools.partial(RBF_kernel, sigma=sigma)
f = train_kernel_ridge_regression(x_train, y_train, k, l2reg=l2reg)
label = "RBF: 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()
```



It seems that RBF fits better than Polynomial especially for the little peak around 0.5 and the peak around 0.3. Also, RBF has a smaller test error.

8.3.7

8(7). $\delta = E(l(\hat{y}, y))$

$$= E(\hat{y} - y)^2 = E([f(x) + \varepsilon - f(x)]^2)$$

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

$$= Var(\varepsilon) + E^2(\varepsilon)$$

$$= (0.1)^2 + 0$$

$$= 0.01$$

Optional Questions