In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import scipy.spatial
import functools
from scipy.spatial import distance

%matplotlib inline
```

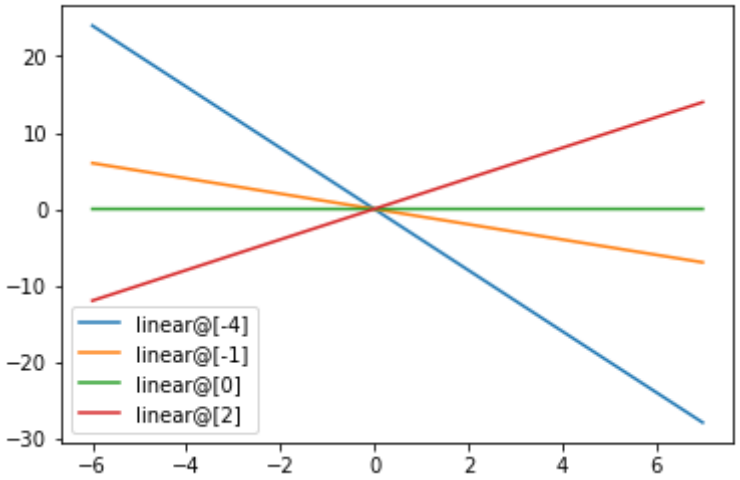
In [2]:
```
### Kernel function generators
def linear_kernel(W, X):
    """
    Computes the linear kernel between two sets of vectors.
    Args:
        W, X - two matrices of dimensions n1xd and n2xd
    Returns:
        matrix of size n1xn2, with w_i^T x_j in position i,j
    """
    return np.dot(W,np.transpose(X))

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 k
ernel
    Returns:
        matrix of size n1xn2, with exp(-||w_i-x_j||^2/(2 sigma^2)) in position
 i,j
    """
    #TODO
    dist = distance.cdist(W,X,'sqeuclidean')
    return np.exp(-(dist)/(2*sigma**2))

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
    """
    #TODO
    return (offset+linear_kernel(W,X))**degree
```

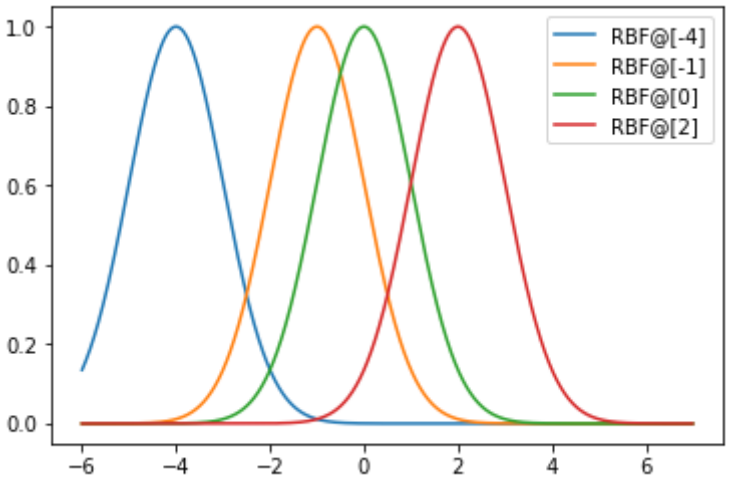
In [3]:
```
# 8.2.2 Kernel Matrix on given set of points
X = [[-4], [-1], [0], [2]]
k = linear_kernel(X,X)
print(k)
```

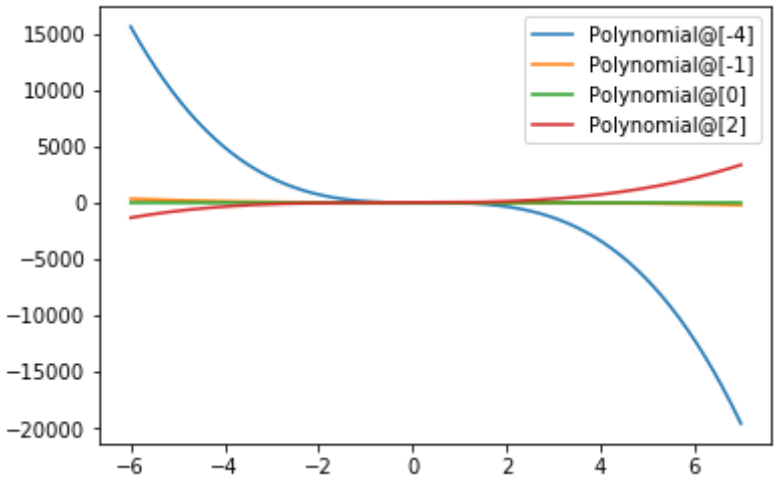
```
[[16  4  0 -8]
 [ 4  1  0 -2]
 [ 0  0  0  0]
 [-8 -2  0  4]]
```

In [4]:
```
# 8.2.3
# PLot kernel machine functions

plot_step = .01
xpts = np.arange(-6.0, 7, plot_step).reshape(-1,1)
prototypes = np.array([-4,-1,0,2]).reshape(-1,1)
print(xpts)
print(prototypes)
# Linear kernel
y_lin = linear_kernel(prototypes, xpts)

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

# RBF Kernel
y_rbf = RBF_kernel(prototypes, xpts,1)
y_poly = polynomial_kernel(prototypes, xpts,1,3)
plot_kmf(y_lin,'linear')
plot_kmf(y_rbf,'RBF')
plot_kmf(y_poly,'Polynomial')
```

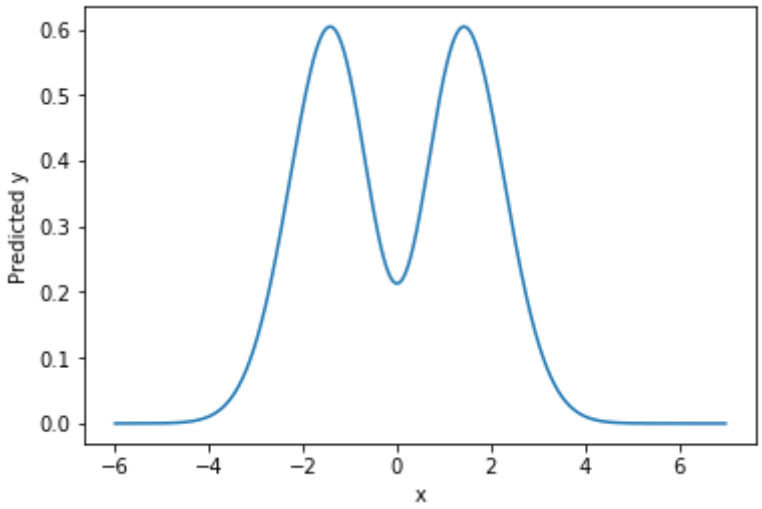
```
[[-6.  ]
 [-5.99]
 [-5.98]
 ...,
 [ 6.97]
 [ 6.98]
 [ 6.99]]
[[-4]
 [-1]
 [ 0]
 [ 2]]
```





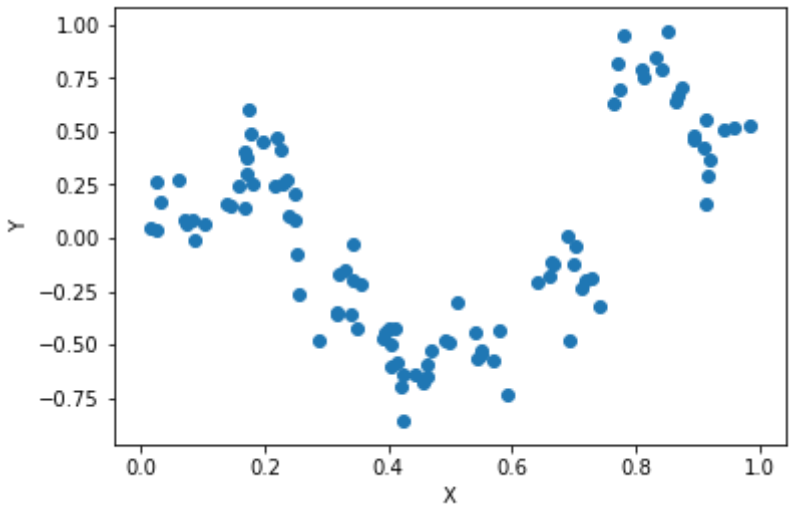

Polynomial@[-4]
Polynomial@[-1]
Polynomial@[0]
Polynomial@[2]
15000
10000
5000
0
−5000
−10000
−15000
−20000
−6
−4
−2
0
2
4
6

In [5]:
```
class Kernel_Machine(object):
    def __init__(self, kernel, prototype_points, weights):
        """
        Args:
            kernel(W,X) - a function return the cross-kernel matrix between ro
ws of W and rows of X for kernel k
            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.  Specif
ically, jth entry of return vector is
                Sum_{i=1}^R w_i k(x_j, mu_i)
        """
        # TODO
        k = self.kernel(self.prototype_points, X)
        return np.dot(self.weights, k)
from functools import partial
kernel = partial(RBF_kernel,sigma=1)
#kernel machine object
kernel_machine = Kernel_Machine(kernel,[[-1],[0],[1]],[1,-1,1])

# Plot the resulting functions
#plot_step = .001
#xpts = np.arange(-3 , 3, plot_step).reshape(-1,1)
plt.plot(xpts, kernel_machine.predict(xpts), label="Prediction Funciton")
plt.xlabel('x')
plt.ylabel('Predicted y')
plt.show()
##prototype_points = np.array([-1,0,1]).reshape(-1,1)
#weights = np.array([1,-1,1]).reshape(-1,1)
#print(prototype_points)
#print(weights)
```



Load train & test data; Convert to column vectors so it generalizes well to data in higher dimensions.

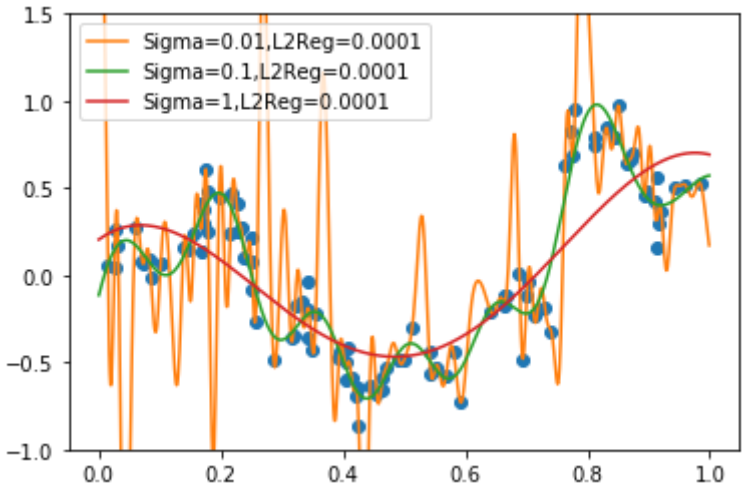
In [6]:
```
data_train,data_test = np.loadtxt("krr-train.txt"),np.loadtxt("krr-test.txt")
x_train, y_train = data_train[:,0].reshape(-1,1),data_train[:,1].reshape(-1,1)
x_test, y_test = data_test[:,0].reshape(-1,1),data_test[:,1].reshape(-1,1)

#8.3.1
# Training Data plot
plt.scatter(x_train,y_train)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



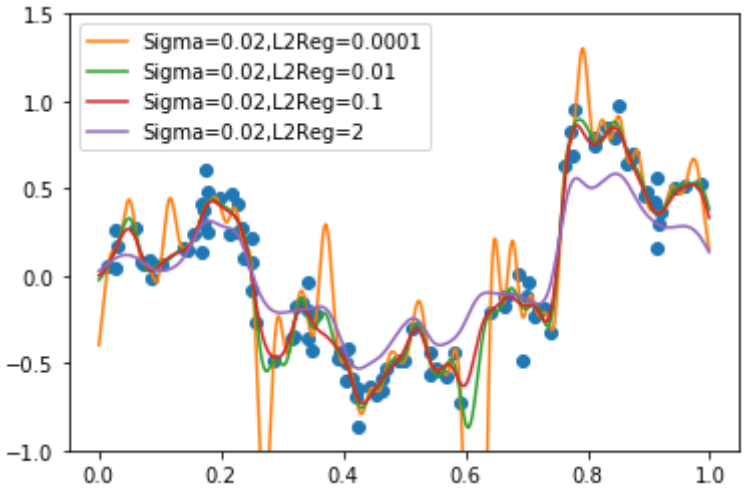
In [7]:
```
def train_kernel_ridge_regression(X, y, kernel, l2reg):
    # TODO
    K = kernel(X,X)
    alpha =
np.dot(np.linalg.inv((l2reg*np.identity(K.shape[0])+K)),y).reshape(1,-1)[0]
    return Kernel_Machine(kernel, X, alpha)
```

In [8]:
```
plot_step = .001
xpts = np.arange(0 , 1, plot_step).reshape(-1,1)
plt.plot(x_train,y_train,'o')
l2reg = 0.0001
print(x_train.shape)
print(y_train.shape)
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()
```

```
(100, 1)
(100, 1)
```



In [9]:
```
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()
```



In [10]:
```
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, sel
f.l2reg)

        return self

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

        return(self.kernel_machine_.predict(X))

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

In [11]:
```
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 t
o train
predefined_split = PredefinedSplit(test_fold=test_fold)
```

In [12]:
```
param_grid = [{'kernel': ['RBF'],'sigma':np.arange(0.01,0.1,0.01), 'l2reg':
np.exp2(-np.arange(-5,5,1))},
              {'kernel':['polynomial'],'offset':[-2,-1,0,1,2,3,4],
'degree':np.arange(4,7,1),'l2reg':[0.01,0.1,10] },
              {'kernel':['linear'],'l2reg': np.arange(100000,1000000,100000)}]
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)))
```

Out[12]:
```
GridSearchCV(cv=PredefinedSplit(test_fold=array([-1, -1, ...,  0,  0])),
       error_score='raise',
       estimator=KernelRidgeRegression(degree=2, kernel='RBF', l2reg=1, offse
t=1, sigma=1),
       fit_params={}, iid=True, n_jobs=1,
       param_grid=[{'kernel': ['RBF'], 'sigma': array([ 0.01,  0.02,  0.03,
 0.04,  0.05,  0.06,  0.07,  0.08,  0.09]), 'l2reg': array([ 32.    ,  16.
,   8.    ,   4.    ,   2.    ,   1.    ,
         0.5   ,   0.25  ,   0.125 ,   0.0625])}, {'kernel': ['polynomial'],
 'offset': [-2, -1, 0, 1, 2, 3, 4], 'degree': array([4, 5, 6]), 'l2reg': [0.0
1, 0.1, 10]}, {'kernel': ['linear'], 'l2reg': array([100000, 200000, 300000,
 400000, 500000, 600000, 700000, 800000,
       900000])}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=make_scorer(mean_squared_error, greater_is_better=False),
       verbose=0)
```

In [13]:
```
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"])
```

Out[13]:

	param_degree	param_kernel	param_l2reg	param_offset	param_sigma	mean_test
87	-	RBF	0.0625	-	0.07	0.013847
77	-	RBF	0.1250	-	0.06	0.013894
68	-	RBF	0.2500	-	0.06	0.013922
58	-	RBF	0.5000	-	0.05	0.013975
67	-	RBF	0.2500	-	0.05	0.014053
57	-	RBF	0.5000	-	0.04	0.014147
78	-	RBF	0.1250	-	0.07	0.014286
86	-	RBF	0.0625	-	0.06	0.014465
56	-	RBF	0.5000	-	0.03	0.014747
66	-	RBF	0.2500	-	0.04	0.014773
...	...	...	...	...	...	...
154	-	linear	200000.0000	-	-	0.168039
155	-	linear	300000.0000	-	-	0.168040
156	-	linear	400000.0000	-	-	0.168040
157	-	linear	500000.0000	-	-	0.168040
158	-	linear	600000.0000	-	-	0.168040
159	-	linear	700000.0000	-	-	0.168040
160	-	linear	800000.0000	-	-	0.168040
161	-	linear	900000.0000	-	-	0.168040
98	4	polynomial	0.1000	-1	-	0.188090
126	5	polynomial	10.0000	-1	-	0.232929

162 rows × 7 columns

In [14]:
```
# results for linear Kernel
linear_report = df_toshow[df_toshow["param_kernel"]=='linear']
linear_report.sort_values(by=['mean_test_score'])    #### Tuned to a certain e
xtent
#plt.scatter(linear_report['param_l2reg'],linear_report['mean_test_score'])
#plt.show()
```

Out[14]:

	param_degree	param_kernel	param_l2reg	param_offset	param_sigma	mean_test
153	-	linear	100000.0	-	-	0.168038
154	-	linear	200000.0	-	-	0.168039
155	-	linear	300000.0	-	-	0.168040
156	-	linear	400000.0	-	-	0.168040
157	-	linear	500000.0	-	-	0.168040
158	-	linear	600000.0	-	-	0.168040
159	-	linear	700000.0	-	-	0.168040
160	-	linear	800000.0	-	-	0.168040
161	-	linear	900000.0	-	-	0.168040

In [15]:
```
# results for RBF Kernel
rbf_report = df_toshow[df_toshow['param_kernel']=='RBF']
rbf_report.sort_values(by=['mean_test_score'])

# observations
# 1. mean_test_score decreases with lambda value so does the train score
# 2. for lambda =0.03125, minimum occurs between 0.01 and 0.1 or 0.01 and 0.00
1(values tried [.0001,.001,.01,.1,1,10])
# 3. min(=0.018699) at 0.009(values tried(0.001,0.1))
# 4. min(=0.018575) at 0.0095(values tried (0.009,0.01))
# 5. min(=      0.014036) at 0.07(values tried (0.01,0.1))
# 6. for the given lambda_range, minm(= 0.013847) occurs at 0.0625 and 0.07 si
gma
```

Out[15]:

	param_degree	param_kernel	param_l2reg	param_offset	param_sigma	mean_test_s
87	-	RBF	0.0625	-	0.07	0.013847
77	-	RBF	0.1250	-	0.06	0.013894
68	-	RBF	0.2500	-	0.06	0.013922
58	-	RBF	0.5000	-	0.05	0.013975
67	-	RBF	0.2500	-	0.05	0.014053
57	-	RBF	0.5000	-	0.04	0.014147
78	-	RBF	0.1250	-	0.07	0.014286
86	-	RBF	0.0625	-	0.06	0.014465
56	-	RBF	0.5000	-	0.03	0.014747
66	-	RBF	0.2500	-	0.04	0.014773
...	...	...	...	...	...	...
6	-	RBF	32.0000	-	0.07	0.089544
5	-	RBF	32.0000	-	0.06	0.093893
4	-	RBF	32.0000	-	0.05	0.099444
10	-	RBF	16.0000	-	0.02	0.104364
18	-	RBF	8.0000	-	0.01	0.105149
3	-	RBF	32.0000	-	0.04	0.106620
2	-	RBF	32.0000	-	0.03	0.116212
1	-	RBF	32.0000	-	0.02	0.129290
9	-	RBF	16.0000	-	0.01	0.129333
0	-	RBF	32.0000	-	0.01	0.146154

90 rows × 7 columns

In [16]:
```
# results for polynomial kernel
poly_report = df_toshow[df_toshow['param_kernel']=='polynomial']
poly_report.sort_values(by=['mean_test_score'])
# observations
# For offset=0,degree=2, minm(=0.151723) occurs between (0.1,0.01)
# minm (=0.151724) at 0.09(0.01,.1)
# minm(=0.151723) t 0.1

# now keeping lambda = 0.1
# min=(0.126171) at 6

# now keeping degree=6 and lambda=0.1
# min(=0.032528) at offset=3
```

Out[16]:

	param_degree	param_kernel	param_l2reg	param_offset	param_sigma	mean_test
144	6	polynomial	0.10	3	-	0.032528
135	6	polynomial	0.01	1	-	0.032700
116	5	polynomial	0.01	3	-	0.032724
117	5	polynomial	0.01	4	-	0.033097
145	6	polynomial	0.10	4	-	0.033520
115	5	polynomial	0.01	2	-	0.033573
143	6	polynomial	0.10	2	-	0.035520
136	6	polynomial	0.01	2	-	0.036289
137	6	polynomial	0.01	3	-	0.038178
138	6	polynomial	0.01	4	-	0.038890
...	...	...	...	...	...	...
113	5	polynomial	0.01	0	-	0.127214
99	4	polynomial	0.10	0	-	0.130804
92	4	polynomial	0.01	0	-	0.130907
127	5	polynomial	10.00	0	-	0.137694
106	4	polynomial	10.00	0	-	0.138611
105	4	polynomial	10.00	-1	-	0.139007
148	6	polynomial	10.00	0	-	0.139045
147	6	polynomial	10.00	-1	-	0.159014
98	4	polynomial	0.10	-1	-	0.188090
126	5	polynomial	10.00	-1	-	0.232929

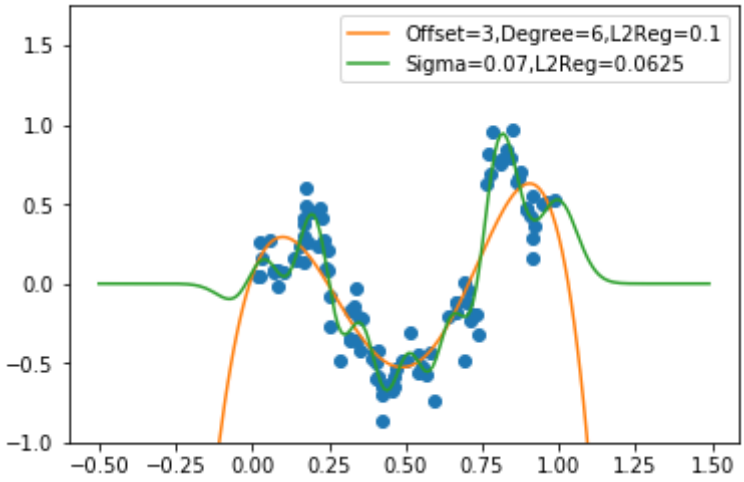
63 rows × 7 columns

In [17]:
```
# The best combination
rbf_report=rbf_report.sort_values(by=['mean_test_score'])
poly_report = poly_report.sort_values(by=['mean_test_score'])
best_rbf_params = rbf_report.iloc[0]
best_poly_params = poly_report.iloc[0]
print("Best RBF Hyperparameters\n", best_rbf_params)
print("Best Polynomial parameters\n", best_poly_params)
```

```
Best RBF Hyperparameters
 param_degree                -
param_kernel              RBF
param_l2reg            0.0625
param_offset                -
param_sigma              0.07
mean_test_score     0.0138468
mean_train_score    0.0145748
Name: 87, dtype: object
Best Polynomial parameters
 param_degree                 6
param_kernel        polynomial
param_l2reg                0.1
param_offset                 3
param_sigma                  -
mean_test_score      0.0325283
mean_train_score      0.048157
Name: 144, dtype: object
```

In [18]:
```
# Can be a more convenient way to look at the table
#import qgrid
#qgrid.nbinstall(overwrite=True)  # copies javascript dependencies to your /nb extensions folderd_toshow = df[show].fillna('-')
#qgrid.show_grid(df_toshow)
```

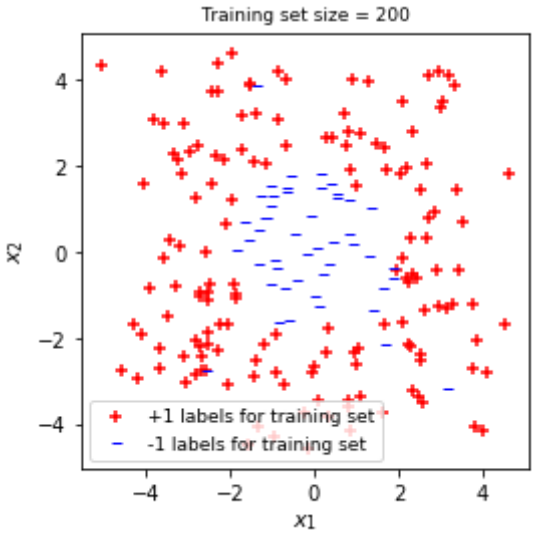
In [19]:
```
## Plot the best polynomial and RBF fits you found
plot_step = .01
xpts = np.arange(-.5 , 1.5, plot_step).reshape(-1,1)
plt.plot(x_train,y_train,'o')
#Plot best polynomial fit
offset= best_poly_params['param_offset']     #### Inseted Code here
degree = best_poly_params['param_degree']    #### Inseted Code here
l2reg = best_poly_params['param_l2reg']      #### Inseted Code here
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 = best_rbf_params['param_sigma']  #### Inseted Code here
l2reg= best_rbf_params['param_l2reg']   ### Inserted Code here
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()
```



Comments:
RBF kernel fits the data really well. Polynomial kernel is restricted from overfitting the data using the regularization term.

In [ ]:
```
# Bayes Decision Function and Risk
```

In [39]:
```
# 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 la
bels for training set')
minuses = ax.scatter (xminus[:,0], xminus[:,1], marker=r'$-$', c='b', label =
'-1 labels for training set')

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()
```



Clearly the data is not linearly separable.
But the data is quadratically separable. We can obtain a reasonalbe classifier using RBF kernel, which could not be obtained otherwise as the data is not linearly separable and RBF provides a richer set of features to work with.

In [40]:
```
np.zeros(5)
```

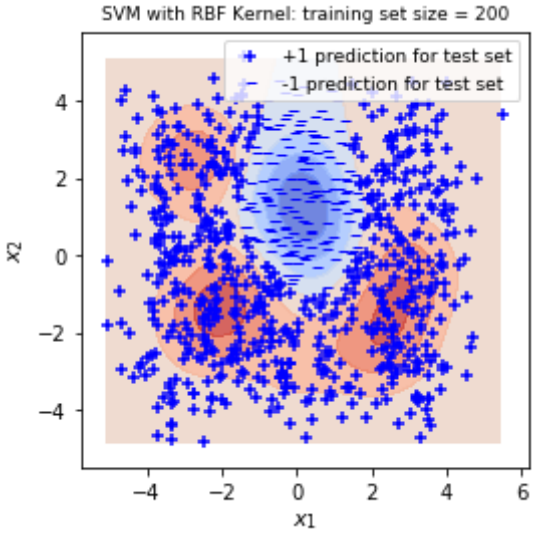
Out[40]:
```
array([ 0.,  0.,  0.,  0.,  0.])
```

In [44]:
```
from random import randint
# kernelized Pegasos(train_soft_svm)
max_iters=10
def train_soft_svm(X,y,kernel,Lambda):
    k=kernel(X,X)
    num = y.shape[0]
    d=X.shape[0]
    iters=0
    t=0
    alpha = np.zeros(d)
    while(iters<max_iters):
        iters = iters+1
        t=t+1
        eta = 1/(t*Lambda)
        j = randint(1,num)
        if ((y[j]*k[j]).dot(alpha.T))<1:
            alpha = (1-1/t)*alpha
            alpha[j]=alpha[j]+(eta*y[j])
        else:
            alpha = (1-1/t)*alpha

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

In [ ]:
```
# Hyperparameter Tuning
from sklearn.base import BaseEstimator, RegressorMixin, ClassifierMixin

class KernelSVM(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, sel
f.l2reg)

        return self

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

        return(self.kernel_machine_.predict(X))

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

In [46]:
```
#Hyperparameter Tuning Contd
sigma=1
k = functools.partial(RBF_kernel, sigma=sigma)
import pdb
#pdb.set_trace()
f = train_soft_svm(x_train, y_train, k, Lambda=1)
```

In [47]:
```
# Code to help plot the decision regions
# (Note: This ode isn't necessarily entirely appropriate for the questions ask
ed. So think about what you are doing.)



#determine the decision regions for the predictions
x1_min = min(x_test[:,0])
x1_max= max(x_test[:,0])
x2_min = min(x_test[:,1])
x2_max= max(x_test[:,1])
h=0.1
xx, yy = np.meshgrid(np.arange(x1_min, x1_max, h),
                     np.arange(x2_min, x2_max, h))

Z = f.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

#determine the predictions for the test set
y_bar = f.predict (x_test)
yplus = np.ma.masked_where(y_bar<=0, y_bar)
xplus = x_test[~np.array(yplus.mask)]
yminus = np.ma.masked_where(y_bar>0, y_bar)
xminus = x_test[~np.array(yminus.mask)]

#plot the learned boundary and the predictions for the test set
figsize = plt.figaspect(1)
f, (ax) = plt.subplots(1, 1, figsize=figsize)
decision =ax.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
pluses = ax.scatter (xplus[:,0], xplus[:,1], marker='+', c='b', label = '+1 pr
ediction for test set')
minuses = ax.scatter (xminus[:,0], xminus[:,1], marker=r'$-$', c='b', label =
'-1 prediction for test set')
ax.set_ylabel(r"$x_2$", fontsize=11)
ax.set_xlabel(r"$x_1$", fontsize=11)
ax.set_title('SVM with RBF Kernel: training set size = %s'% len(data_train), f
ontsize=9)
ax.axis('tight')
ax.legend(handles=[pluses, minuses], fontsize=9)
plt.show()
```


SVM with RBF Kernel: training set size = 200
+1 prediction for test set
-1 prediction for test set
x2
x1
4
2
0
−2
−4
−4
−2
0
2
4
6

In [ ]:

9.1)

9.1) $\|m_0\| = \|x\|$

$\Rightarrow \|m_0\|^2 = \|x\|^2$

$\Rightarrow \|x - m_0\|^2 = 0$ (By Pythagorean Theorem)

$\Rightarrow \langle (x - m_0), (x - m_0) \rangle = 0$

$\Rightarrow x - m_0 = 0$

$\Rightarrow \boxed{x = m_0}$

9.3)

9.3 Let $A = \begin{bmatrix} — \Psi(x_1) — \\ \vdots \\ — \Psi(x_n) — \end{bmatrix}$

$\Rightarrow w \longrightarrow L(\langle w, \Psi(x_1) \rangle \cdots \langle w, \Psi(x_n) \rangle) = w \longrightarrow L(Aw)$

$\Rightarrow w \longrightarrow L(Aw)$ is convex since L is convex & composition of convex & Affine is con

Also, $\|w\|$ is convex & $w \to R(\|w\|)$ is convex $\rightarrow$ (2)
(∵ all norms on R^d is convex).

(1) + (2) $\Rightarrow \boxed{J(w) \text{ is convex}}$

6.1) $J_i(w) = \frac{\lambda}{2} \|w\|_2^2 + l_i(w)$.

$\Rightarrow \partial(J_i(w)) = \partial\left(\frac{\lambda}{2} \|w\|_2^2\right) + \partial(l_i(w))$.

$\Rightarrow g_i(w) = \lambda \|w\| + \partial_i(w)$

$\Rightarrow \boxed{g_i(w) = \partial_i(w) + \lambda \|w\|}$

6.2)

10.1)

1) Let $\gamma = \Omega(f_*)$

if $\exists$ some f' with $\Omega(f') \leq \gamma$ and

$$\phi(f') < \phi(f_*)$$

$$\Rightarrow \quad \phi(g) + \Omega(g) < \phi(f_*) + \lambda \Omega(f_*) .$$

$\Rightarrow$ f_* is not a Tikhonov regularization Solution

A. Contradiction

$\Rightarrow$

$\Rightarrow$ f_* is also an Ivanov solution.

9.2)

9.2) Proof by contradiction

Suppose Not

i.e the minimizer w^* has a component orthogonal to the span of Data M.

- Let w = component of w^* in the span of data

$w^{\perp} = (w^* - w)$ = " " w^* orthogonal to M.

e have, $\|w\| \leq \|w^*\|$

① $\|w\| = \|w^*\| \Rightarrow w^* = w \Rightarrow$ **w^* is in the span of Data**

② $\|w\| < \|w^*\| \Rightarrow R(\|w\|) < R(\|w^*\|)$ [$\because$ R is strictly increasing] — ①

$L(\langle w^*, \psi(x_1)\rangle \ldots \langle w^*, \psi(x_n)\rangle)$ — ②

$= L(\langle w, \psi(x_1)\rangle \ldots \langle w, \psi(x_n)\rangle)$

[$\because$ inner product with $w^{\perp} = 0$]

i.e $\langle w + w^{\perp}, \psi(x_1)\rangle = \langle w, \psi(x_1)\rangle$

① + ② $\Rightarrow$ $J(w) < J(w^*)$ → contradiction

$\Rightarrow$ w^* must be in the span of Data.

10.2.1) $L(w, \lambda) = \phi(w) + \lambda[\Omega(w) - \gamma]$

10.2.2) $\max_{\lambda \geq 0} \left(\min_{w} \left(\phi(w) + \lambda[\Omega(w) - \gamma] \right) \right)$

10.2.3) Given,

$$\phi(w^*) = g(\lambda^*)$$

$$= \min_{w} \left(\phi(w) + \lambda^* [\Omega(w) - \gamma] \right)$$

$$\leq \phi(w^*) + \lambda^* [\Omega(w^*) - \gamma]$$

$\lambda^* [\Omega(w^*) - \gamma] \leq 0$

Minimum must be smaller than a specific value

$$\leq \phi(w^*)$$

$$\Rightarrow w^* = \arg\min_{w^*} \left(\phi(w) + \lambda^* [\Omega(w) - \gamma] \right)$$

$$= \arg\min_{w} \left(\phi(w) + \lambda^* \Omega(w) \right)$$

$$\Rightarrow \boxed{w^* = \arg\min_{w} \phi(w) + \lambda^* \Omega(w)}$$

10.2.4) The dual Function,

$$g(\lambda) = \min_{w} \left(\phi(w) + \lambda [\Omega(w) + r] \right)$$

we have,

$$g(0) = \min_{w} (\phi(w)) < \inf_{\substack{w \in R^d \\ \Omega(w) \leq r}} \phi(w) = g(\lambda^*).$$

$\Rightarrow \lambda^* \neq 0$ — (1) *; Also we know that

$\lambda \geq 0$ — (2)

(1) + (2) $\Rightarrow \lambda > 0$

10.3) Ridge Regression in Ivanov form:

$$\text{minimize} \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

$$\text{s.t} \quad w^T w \leq \gamma$$

~~Clearly~~

① To show that it is a convex optimization problem we need to show that both objective function & ~~convex~~ constraints are convex.

objective function: is convex since it is composition of a convex function (square function) & an affine function

constraint $w^T w \leq \gamma$ is convex clearly.

Also, it has a strictly feasible point, $w = 0$ [$\gamma > 0$]

$\Rightarrow$ Slater's conditions are satisfied

$\Rightarrow$ Strong Duality

1.1) Novelty Detection algorith can be described as the following objective funtion

$$\min_{c \in F, r \in R} r^2 \quad \rightarrow \text{objective Function}$$

$$\text{s.t} \quad \| \psi(x_i) - c \|^2 \leq r^2 \quad \forall i$$

1.2) Lagrangian:

$$g(\lambda) = \left(r^2 + \sum_{\text{all } i} \lambda_i \left(\| \psi(x_i) - c \|^2 - r^2 \right) \right)$$

where $\lambda_i \rightarrow$ Lagrangian multiplier

"inf sup" version →

$$\inf_{c \in F, r \in R} \sup_{\lambda \geq 0} g(\lambda)$$

(5.) $J(w) = \|Xw - y\|^2 + \lambda \|w\|^2$

For w to be a minimizer,

$$\frac{d\,J(w)}{dw} = 0$$

$$\Rightarrow \frac{d}{dw}\left(X^TXw^2 + y^2 - 2X^Tyw + \lambda \|w\|^2 \right) = 0$$

$$\Rightarrow 2X^TXw + 0 - 2X^Ty + 2\lambda w = 0$$

$$\Rightarrow 2X^TXw + 2\lambda w - 2X^Ty = 0$$

$$\Rightarrow 2(X^TXw + \lambda w - X^Ty) = 0$$

$$\Rightarrow (X^TX + \lambda I)w = X^Ty$$

$$\Rightarrow w = (X^TX + \lambda I)^{-1} X^Ty .$$

Also $\frac{d^2J}{dw^2} = 2X^TX + 2\lambda > 0$

$\Rightarrow$ w is a minimizer

$\Rightarrow (X^TX + \lambda I)^{-1} X^Ty$ is a minimizer

X^TX is PSD and adding λI makes it positive definite. $\Rightarrow (X^TX + \lambda I)$ is invertible for $\lambda > 0$.

5.2) We have,

$$(X^T X w + \lambda I w) = X^T y$$

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

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

$$= X^T \alpha, \quad \boxed{\text{where } \alpha = \frac{y - Xw}{\lambda}}$$

5.3) From Above Expressions, we can see that w is a linear combination of input vectors i.e $X \Rightarrow$ w is in the dimension of the input vectors $\Rightarrow$ w is in the span of data.

5.4) $$\alpha = \left(\frac{y - Xw}{\lambda}\right)$$

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

$$\Rightarrow \lambda\alpha = (y - XX^T\alpha)$$

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

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

5.5)

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

$$= K(\lambda I + K)^{-1} y.$$

5.6) $f(x) = x^T w^*$, w^* is the minimizer.

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

$$= x^T X^T \alpha^*$$

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

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

9#

7.1) $y_j \langle w^{(t)}, x_j \rangle$

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

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

where K_j = j^{th} row of Kernel matrix

$\alpha^{(t)} = (\alpha_1^{(t)} \cdots \alpha_n^{(t)})^T$

7.2 No margin violation $\Rightarrow$

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

$$= (1 - 1/t)\, w^{(t)}$$

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

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

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

where $\boxed{\alpha_i^{(t+1)} = (1 - \eta^{(t)}\lambda)\, \alpha_i^{(t)}}$

7.

MARGIN VIOLATION $\Rightarrow$

7.3) $w^{(t+1)} = (1 - \eta^{(t)}\lambda) w^{(t)} + \eta_t y_j x_j$

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

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

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

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

Where,

$$\alpha_i^{(t+1)} = (1 - \eta^{(t)}\lambda)\alpha_i^{(t)}, \quad i \neq j$$

$$= (1 - \eta^{(t)}\lambda)\alpha_i^{(t)} + \eta_t y_j, \quad i = j \; //$$

Pseudo-Code

input : Kernel Matrix K, labels $y_1, y_2 \cdots y_n \in \{-1, 1\}$

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

$t = 0$ # step number

repeat

$t = t + 1$

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

randomly choose j in $1, \cdots n$

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

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

$$\alpha^{(t+1)}[j] = \alpha^{[t+1)}[j] + \eta_t y_j$$

else

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

until bored

return $\alpha_*^{(t)}$

2.1 For a 2×2 matrix $\begin{bmatrix} p & t \\ q & u \end{bmatrix}$

orthogonality $\Rightarrow$ $p^2 + t^2 = 1$

$q^2 + u^2 = 1$

$pt + qu = 0$

$\therefore \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ is orthogonal but not symmetric

2.2) By Spectral Theorem,

$\Sigma = Q^T M Q$ is a diagonal Matrix of eigen values of M

Also, M is psd $\Leftrightarrow$ for any $A \in R^{n \times d}$

$\text{diag}(A^T M A) \geq 0$

$\Rightarrow \text{diag}(Q^T M Q) \geq 0$

$\Rightarrow \text{diag}(\Sigma) \geq 0$

$\Rightarrow$ Eigen values of $M \geq 0$

2.3) Part 1 : M is psd $\Rightarrow M = BB^T$

M is psd $\Rightarrow M = Q\Sigma Q^T$ (by spectral Theorem)

$\Rightarrow M = Q\Sigma^{1/2}(\Sigma^{1/2})^T Q^T$ (Σ = diag (all eigenvalues))

$= BB^T$

$\because$ all eigenvalues ≥ 0

$\Rightarrow$ we can have

$\Sigma^{1/2}$ = diag (square root of eigenvalues)

part 2 : $M = BB^T \Rightarrow$ M is a psd.

$M = BB^T \Rightarrow x^T M x$

$= x^T BB^T x$

$= (B^T x)^T B^T x$

$= \|B^T x\|^2 \geq 0$

$\Rightarrow$ M is psd.

3.1) By spectral Theorem,

$\Sigma = Q^T M Q$ is a diagonal matrix of eigen values.

Also, M is positive definite $\Leftrightarrow$ for any $A \in R^d$

$$\text{diag}(A^T M A) > 0$$

$$\Rightarrow \text{diag}(\Sigma) > 0$$

$$\Rightarrow \overset{\text{All}}{\text{Eigen values of}}\ M > 0$$

3.2) $M = Q \Sigma Q^T$

Let $M' = Q \Sigma^{-1} Q^T$

$$M'M = (Q \Sigma^{-1} Q^T)(Q \Sigma Q^T)$$

$$= Q \Sigma^{-1} (Q^T Q) \Sigma Q^T \quad [\because Q \text{ is orthogonal} \Rightarrow Q^T Q = I]$$

$$= Q (\Sigma^{-1} \Sigma) Q^T \quad [\Sigma^{-1}\Sigma = \text{diag}(\sigma_1 \sigma_1^{-1} \cdots \sigma_n^{-1} \sigma_n) = \text{diag}(1 \ldots 1) = I]$$

$$= Q Q^T$$

$$= I \quad [\because Q \text{ is orthogonal}]$$

$\Rightarrow$ $M' = Q \Sigma^{-1} Q^T$ is inverse of M

3.3) $M+\lambda I$ is spd

iff $x^T(M+\lambda I)x > 0$ for any $x \in R^n$

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

~~$= x^T M x$~~

$x^T M x \geq 0$ $\because$ M is psd

$x^T \lambda I x > 0$ $\because \lambda > 0 \; \& \; x \neq 0$

$\Rightarrow x^T M x + x^T \lambda I x > 0$

$\Rightarrow M+\lambda I$ is spd.

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

$= (Q(\Sigma + \lambda I)Q^T)^{-1}$

$= Q(\Sigma+\lambda I)^{-1} Q^T$

3.9) $M+N$ is spd

iff $x^T(M+N)x > 0$

iff, $x^T Mx + x^T Nx > 0$

$x^T Mx \geq 0$ and $x^T Nx > 0$

$\because M$ is psd $\quad \because N$ is ~~semi~~-SPD

$\Rightarrow x^T Mx + x^T Nx > 0$

$\Rightarrow M+N$ is spd

(4.) distance b/w 2 points x_1, x_2

$= d(x_1, x_2) = \|x_1 - x_2\|$

$= \|x_1\|^2 + \|x_2\|^2 - 2\langle x_1, x_2 \rangle$

$K_{11} = \langle x_1, x_1 \rangle = \|x_1\|^2$

$K_{22} = \langle x_2, x_2 \rangle = \|x_2\|^2$

$K_{12} = \langle x_1, x_2 \rangle$

$\Rightarrow$ distance b/w 2 points can be computed using Kernel Matrix Entries.

8.3.7 Bayes decision function for square loss for given $X \sim Y$ is $E[y|x]$

$$E[y|x] = E[f(x) + \epsilon | x] = f(x)$$

Associated Bayes Risk

$$\text{var}(y|x)$$

$\because f(x)$ is independent of ϵ

$$\text{Var}(y|x) = \text{var}(f(x) + \epsilon | x)$$

$$= \text{Var}(f(x|x) + \text{var}(\epsilon | x)$$

$$= \text{var}(\epsilon) = 0.1$$