Q1.

For any $z \in R^d$.

$$f(z) \geq f_k(z) \quad \text{since } f(z) = \max_{i=1\cdots m} f_i(z)$$

$$\geq f_k(x) + g^T(z-x) \quad \text{since } g \in \partial f_k(x)$$

$$= f(x) + g^T(z-x) \quad \text{since } f(x) = f_k(x)$$

Hence, it's proved that:

for all $z \in R^d$. $f(z) \geq f(x) + g^T(z-x)$, $g \in \partial f(x)$

Q2.

$$\partial J(w) = \begin{cases} 0 & \text{for } yw^Tx > 1 \\ -yx & \text{for } yw^Tx \leq 1 \end{cases}$$

Q3

$$\nabla J_i(w) = \begin{cases} \lambda w - y_i x_i & \text{for } y_i w^T x_i < 1 \\ \lambda w & \text{for } y_i w^T x_i > 1 \\ \text{undefined} & \text{Otherwise.} \end{cases}$$

Q 4

let $f_i(w_i) = \max\{0, 1 - y_i w_i^T x_i\}$; $a \in \{0, 1\}$.

$$
\begin{aligned}
& a f_i(w_1) + (1-a) f_i(w_2) \\
& \geq a \max\{0, 1 - y_i w_1^T x_i\} + (1-a) \max\{0, 1 - y_i w_2^T x_i\} \\
& \geq \max\{0, a(1 - y_i w_1^T x_i) + (1-a)(1 - y_i w_2^T x_i)\} \\
& \geq \max\{0, 1 - y_i (a w_1^T + (1-a) w_2^T) x_i\} \\
& \geq f_i(a w_1 + (1-a) w_2)
\end{aligned}
$$

Since $a f_i(w_1) + (1-a) f_i(w_2) \geq f_i(a w_1 + (1-a) w_2)$
it's proved that $f_i(w_i)$ is a convex function.
Obviously. $\frac{\lambda}{2} \| w \|^2$ is also a convex function.
Then. $J_i(w) = \frac{\lambda}{2} \| w \|^2 + f_i(w)$ is proved a convex function

$$\partial J_i(w) = \partial \left(\frac{\lambda}{2} \| w \|^2\right) + \partial (f_i(w))$$

Consider that:

$$
\begin{aligned}
\partial f_i(w) &= \partial (\max\{0, 1 - y_i w_i^T x_i\}) \\
&= \begin{cases} -y_i x_i & \text{for } y_i w^T x_i < 1 \\ 0 & \text{for } y_i w^T x_i \geq 1 \end{cases}
\end{aligned}
$$

Hence:

$$
g_W = \partial J_i(w) = \begin{cases} \lambda w - y_i x_i & \text{for } y_i w^T x_i < 1 \\ \lambda w & \text{for } y_i w^T x_i \geq 1 \end{cases}
$$

When step size is $\eta_t = \frac{1}{\lambda t}$

Then each time we update w with subgradient direction from the previous problem.

$$w_{t+1} = \begin{cases} w_t - \eta_t(\lambda w_t - y_j x_j) & \text{for } y_j w^T x_j < 1 \\ w_t - \eta_t \lambda w_t & \text{for } y_j w^T x_j \geq 1 \end{cases}$$

That is:

$$w_{t+1} = \begin{cases} (1-\eta_t\lambda) w_t + \eta_t y_j x_j & \text{for } y_j w^T x_j < 1 \\ (1-\eta_t\lambda) w_t & \text{for } y_j w^T x_j \geq 1 \end{cases}$$

which shows the same result as given in the pseudocode

Q5

- Write a function that converts an example (a list of words) into a sparse bag-of-words representation. You may find Python's Counter3 class to be useful here. Note that a Counter is itself a dictionary.

```
def construct_word_dict(lst_of_words):
    """
    convert the given list of words into a sparse bag-of-words representation
    """
    dict_of_words = Counter(lst_of_words)

    return dict_of_words
```

```
# construct_word_dict(data[0][:-1])
```

Q6

- Load all the data and split it into 1500 training examples and 500 validation examples. Format the training data as a list X train of dictionaries and y train as the list of corre- sponding 1 or -1 labels. Format the test set similarly.

```
class review:
    def __init__(self, lst_of_words_):
        self.lst_of_words = lst_of_words_[:-1]
        self.dict_of_words = Counter(self.lst_of_words)
        self.label = lst_of_words_[-1]
```

```
X_train_lst_of_reviews = list(map(review, data[:1500]))
X_val_lst_of_reviews = list(map(review, data[1500:2000]))
```

```
X_train = [review.dict_of_words for review in X_train_lst_of_reviews]
X_val = [review.dict_of_words for review in X_val_lst_of_reviews]
```

```
y_train = [review.label for review in X_train_lst_of_reviews]
y_val = [review.label for review in X_val_lst_of_reviews]
```

```
len(X_train)
len(X_val)
```

```
1500
```

```
500
```

Q7

```
def pegasos_algorithm(lst_of_reviews, epoch, lamb):
    # initialization
    np.random.seed(9)

    weight = Counter()
    iter_num = 0
    epoch_ = 1

    while epoch_ <= epoch:
        shuffled_lst_of_reviews = np.random.choice(list(range(len(lst_of_reviews))), len(lst_of_reviews), replace=False
        for j in shuffled_lst_of_reviews:
            iter_num += 1
            eta = 1.0 / (iter_num * lamb)
            increment(weight, - eta * lamb, weight)
            condition = lst_of_reviews[j].label * dotProduct(lst_of_reviews[j].dict_of_words, weight)
            if condition < 1:
                increment(weight, eta * lst_of_reviews[j].label, lst_of_reviews[j].dict_of_words)
        epoch_ += 1
    return weight
```

Q 8

Given that:

$$\begin{cases} W = sW \\ s_{t+1} = (1 - \eta_t \lambda) s_t \\ W_{t+1} = W_t + \frac{1}{s_{t+1}} \eta_t y_j x_j \end{cases}$$

We can deduce that:

$$\begin{aligned} W_{t+1} &= s_{t+1} W_{t+1} \\ &= (1 - \eta_t \lambda) s_t \cdot \left(W_t + \frac{1}{s_{t+1}} \eta_t y_j x_j \right) \\ &= (1 - \eta_t \lambda) \left(s_t W_t + \frac{s_t}{s_{t+1}} \eta_t y_j x_j \right) \\ &= (1 - \eta_t \lambda) \left(w_t + \frac{1}{1 - \eta_t \lambda} \eta_t y_j x_j \right) \\ &= (1 - \eta_t \lambda) W_t + \eta_t y_j x_j \end{aligned}$$

Hence it's verified that:

Pegasos Algorithm with the (s, W) representation works the same way as the Pegasos update step.

Q8

```
def pegasos_algorithm_fast(lst_of_reviews, epoch, lamb):
    # initialization
    np.random.seed(9)

    weight = Counter()
    s = 1
    iter_num = 1
    epoch_ = 1

    while epoch_ <= epoch:
        shuffled_lst_of_reviews = np.random.choice(list(range(len(lst_of_reviews))), len(lst_of_reviews), replace=False
        for i in shuffled_lst_of_reviews:
            iter_num += 1
            eta = 1.0 / (iter_num * lamb)
            condition = lst_of_reviews[i].label * dotProduct(lst_of_reviews[i].dict_of_words, weight) * s
            s = (1 - eta * lamb) * s
            if  condition < 1:
                increment(weight, eta * lst_of_reviews[i].label / s, lst_of_reviews[i].dict_of_words)
        epoch_ += 1

    epoch_weight = Counter()
    increment(epoch_weight, s, weight)

    return epoch_weight
```

Q9

```
lamb_ = 0.01
epoch = 5

start = time.time()
weight_ = pegasos_algorithm(X_train_lst_of_reviews, epoch, lamb_)
end = time.time()
print('epoch:' + str(epoch) + 'pegasos_algorithm run time:' + str(end-start))

start = time.time()
weight_fast = pegasos_algorithm_fast(X_train_lst_of_reviews, epoch, lamb_)
end = time.time()
print('epoch:' + str(epoch) + 'pegasos_algorithm_fast run time:' + str(end-start))

increment(weight_, -1, weight_fast)
sqr_diff = dotProduct(weight_, weight_)
print("The square difference between the weight of pegasos_algorithm and the weight of pegasos_algorithm_fast is: "+ st
```

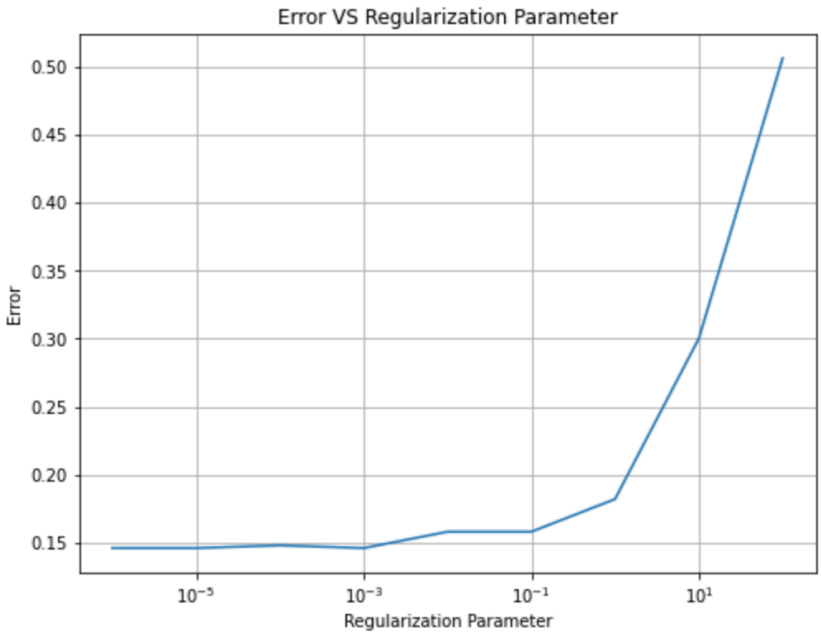
```
epoch:5pegasos_algorithm run time:70.50635290145874
epoch:5pegasos_algorithm_fast run time:0.7125890254974365
The square difference between the weight of pegasos_algorithm and the weight of pegasos_algorithm_fast is: 1.11399253
53372174e-05
```

Q10

```
def sign(review_dict_of_words, weight):
    if dotProduct(review_dict_of_words, weight) > 0:
        return 1
    else:
        return -1

def classification_error(weight, lst_of_reviews):
    predict_true = [sign(review.dict_of_words, weight) != review.label for review in lst_of_reviews]

    return sum(predict_true)/len(predict_true)
```

Q11

```
param_lst = [10 ** n for n in range(-6, 3)]
weight_list = []
percent_err_list = []
for lamb_ in param_lst:
    weight_ = pegasos_algorithm_fast(lst_of_reviews=X_train_lst_of_reviews, epoch=100, lamb=lamb_)
    weight_list.append(weight_)
    per_err = classification_error(weight=weight_, lst_of_reviews=X_val_lst_of_reviews)
    percent_err_list.append(per_err)
fig, ax = plt.subplots(figsize = (8,6))
ax.grid()
ax.set_xlabel("Regularization Parameter")
ax.set_ylabel("Error")
plt.semilogx(param_lst, percent_err_list)
plt.show();
```



Q12

```
# calculate the mean score for each group
weight_ = pegasos_algorithm_fast(lst_of_reviews=X_train_lst_of_reviews, epoch=100, lamb=0.001)
y_predict = []
for reivew in X_val_lst_of_reviews:
    y_predict.append(dotProduct(reivew.dict_of_words, weight_))
```

```
bin_size = 5
group_num = int(len(y_predict)/bin_size)
y_predict_group = np.array_split(y_predict, bin_size)
group_mean_score = []
group_per_error = []
group_index = 0
for group in y_predict_group:
    group_mean_score.append(np.mean(group))
    group_per_error.append(classification_error(weight=weight_, \
                                                lst_of_reviews=X_val_lst_of_reviews[group_num*group_index:\
                                                                                   group_num*(group_index+1)]))
    group_index += 1
```

```
rank_index_lst = list(np.array(group_mean_score).argsort()) # ascending
group_per_error_rank = [group_per_error[index] for index in rank_index_lst]
group_mean_score_rank = np.sort(group_mean_score)
```

```
plt.plot(group_mean_score_rank, group_per_error_rank)
plt.title(f'Percentage Error VS Average Score For the {bin_size} groups')
plt.xlabel('Average Score')
plt.ylabel('Percentage Error')
plt.show();
```

Q13:

The model might be incorrect in following two situations:

1) There are negative words in a positive comment; or, there are positive words in a negative comments. It would confuse the model.

2) The model might overweight the stop words such as "and". However, "and" doesn't convey any sentiment in human sense. This leads to a wrong classfication result.

To fix these problems, we can add these features:

1) N-grams. We can concatenate 2 words together as a feature or n words together as a feature. This helps the machine to understand the context better. For example, the positive words in a negative comment might be understood as contextually negative if we look into the neighboring words.

2) Tf-idf. Essentially, tf-idf weights the frequency of a term by its inverse frequency of appearing in a document. The stop words such as "and" will appear in many documents. Thus, the tf-idf value of "and" tends to be weighted down and the algorithm tends to rely less on these stop words to make a decision.

Q 14.

$$
\begin{aligned}
J(w) &= \|Xw-y\|^2 + \lambda\|w\|^2 \\
&= \langle Xw-y,\ Xw-y\rangle + \lambda\langle w, w\rangle \\
&= (Xw-y)^T(Xw-y) + \lambda w^T w \\
&= (w^T X^T - y^T)(Xw-y) + \lambda w^T w \\
&= w^T X^T X w - y^T X w - w^T X^T y + y^T y + \lambda w^T w
\end{aligned}
$$

$$
\begin{aligned}
\nabla J(w) &= (X^T X + X^T X)w - X^T y - X^T y + 2\lambda I w \\
&= 2X^T X w - 2X^T y + 2\lambda I w
\end{aligned}
$$

For w to be a minimizer of $J(w)$,
we must have $\nabla J(w) = 2X^T X w - 2X^T y + 2\lambda I w = 0$.

In other words, we must have: $X^T X w + \lambda I w = X^T y$

To prove $(X^T X + \lambda I)$ is invertible:

let $M \in R^d$

then $M^T(X^T X + \lambda I)M = M^T X^T X M + \lambda M^T M$

$= \langle XM, XM\rangle + \lambda\langle M, M\rangle \geq 0$

Hence it's proved that $X^T X + \lambda I$ is psd.

Then $X^T X + \lambda I$ is proved invertible

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

Q15

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

Q16

$$w = X^T a$$
$$= \begin{pmatrix} | & & | \\ x_1 & \cdots & x_n \\ | & & | \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$$
$$= a_1 x_1 + \cdots + a_n x_n$$

Since w is the linear combination of X, it's " in the span of the data"

Q17

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

식 18

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

식 19

$$f(x) = x^Tw^*$$
$$= x^TX^T\alpha$$
$$= x^T\begin{pmatrix} -x_1- \\ \vdots \\ -x_n- \end{pmatrix}\alpha$$
$$= \begin{pmatrix} x^Tx_1 \\ \vdots \\ x^Tx_n \end{pmatrix}\alpha$$
$$= k_x\alpha$$

Q20

```
### 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
    """
    #TODO
    return np.exp(-scipy.spatial.distance.cdist(X1, X2, 'sqeuclidean') / (2*sigma**2))

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

Q21

```
# Plot kernel machine functions
plot_step = .01
xpts = np.arange(-5.0, 6, plot_step).reshape(-1, 1)
prototypes = np.array([-4, -1, 0, 2]).reshape(-1, 1)

# Compute the kernel matrix
proto_kernel_mat = linear_kernel(prototypes, prototypes)
proto_kernel_mat
```

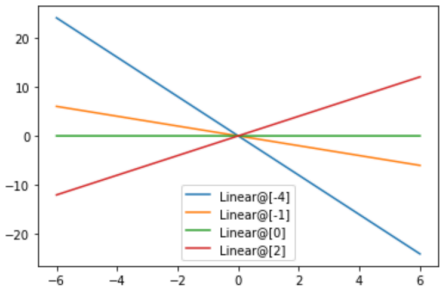
```
array([[16,  4,  0, -8],
       [ 4,  1,  0, -2],
       [ 0,  0,  0,  0],
       [-8, -2,  0,  4]])
```

Q22

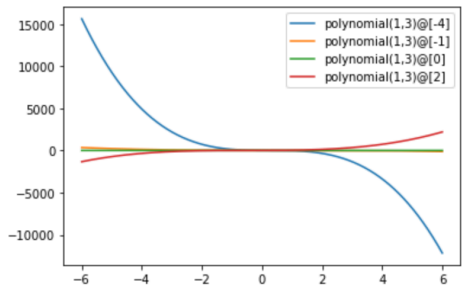
```
plot_step = .01
xpts = np.arange(-6.0, 6.01, plot_step).reshape(-1, 1)
prototypes = np.array([-4,-1,0,2]).reshape(-1, 1)
```

```
### Plot kernel machine
```

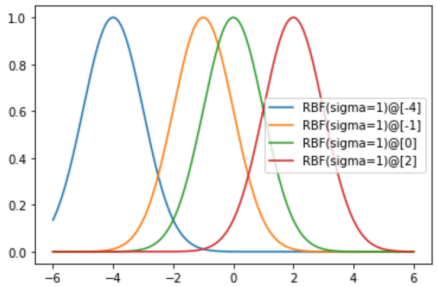
```
# Linear kernel
y = linear_kernel(prototypes, xpts)
for i in range(len(prototypes)):
    label = "Linear@"+str(prototypes[i,:])
    plt.plot(xpts, y[i,:], label=label)
plt.legend(loc = 'best');
plt.show();
```




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



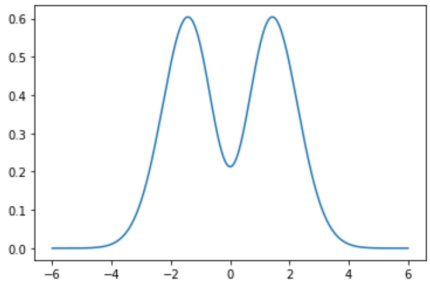

```
# RBF kernel
y = RBF_kernel(prototypes, xpts, sigma=1)
for i in range(len(prototypes)):
    label = "RBF(sigma=1)@"+str(prototypes[i,:])
    plt.plot(xpts, y[i,:], label=label)
plt.legend(loc = 'best')
plt.show();
```



Q23

```
class Kernel_Machine(object):
    def __init__(self, kernel, training_points, weights):
        """
        Args:
            kernel(X1,X2) - a function return the cross-kernel matrix between rows of X1 and rows of X2 for kernel k
            training_points - an nxd matrix with rows x_1,..., x_n
            weights - a vector of length n with entries alpha_1,...,alpha_n
        """

        self.kernel = kernel
        self.training_points = training_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 vector is
                Sum_{i=1}^R alpha_i k(x_j, mu_i)
        """
        # TODO
        return self.kernel(X, self.training_points) @ self.weights
```

```
proto_points = np.array((-1,0,1)).reshape(-1,1)
weights = np.array((1,-1,1)).reshape(-1,1)

RBF_partial = functools.partial(RBF_kernel, sigma=1)
RBF_func = Kernel_Machine(RBF_partial, proto_points, weights)

plot_step = .01
xpts = np.arange(-6.0, 6.01, plot_step).reshape(-1, 1)
plt.plot(xpts, RBF_func.predict(xpts))
plt.show();
```

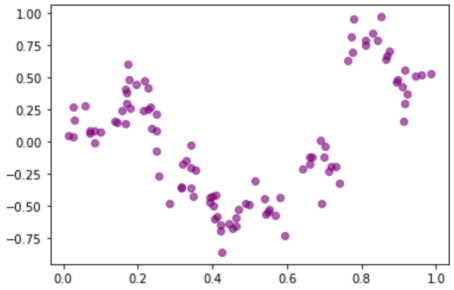


Q24

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

```
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', color='purple', alpha=0.6);
```

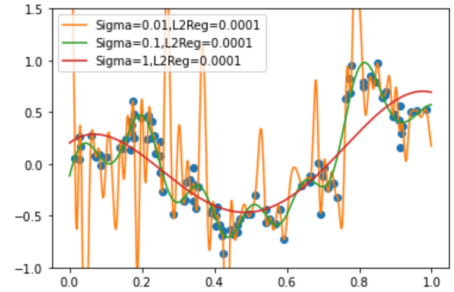


Q25

```
def train_kernel_ridge_regression(X, y, kernel, l2reg):
    # TODO
    alpha = np.linalg.inv(l2reg * np.identity(len(X)) + kernel(X, X)) @ y

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

Q26

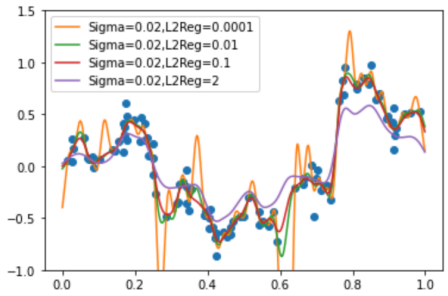
```
plot_step = .001
xpts = np.arange(0 , 1, plot_step).reshape(-1,1)
plt.plot(x_train,y_train,'o')

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();
```



- When Sigma = 0.1, the plot fits the training data the best;
- When Sigma = 0.01, the plot overfits the training data;
- When Sigma = 1, the plot underfits the training data;

Q27

```
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();
```



- As λ -> ∞, the regularizer tends to penalize the weights of the prediction function less heavily. In other words, the prediction function would behave underfitting on the training set.
- It's shown in this plot that, with a low L2Reg(i.e. L2Reg=0.0001), the prediction function overfits the data; as L2Reg becomes larger(i.e. L2Reg=2), that underfits the data.

Q28

```
df_RBF = df_toshow[df_toshow.param_kernel == 'RBF']

mean_test_score_RBF_lst = df_RBF.mean_test_score.tolist()
param_l2reg_lst = df_RBF.param_l2reg.tolist()
param_sigma = df_RBF.param_sigma.tolist()

best_param_l2reg, best_param_sigma = df_RBF.iloc[np.argmin(mean_test_score_RBF_lst), [2, 4]]
print(f'The best param_l2reg is {best_param_l2reg}')
print(f'The best param_sigma is {best_param_sigma}')

the_param_l2reg_lst = [0.0625, 0.125] # param_sigma = 0.1
the_param_sigma_lst = [0.1, 1] # param_l2reg = 0.0625

def_RBF_param = pd.DataFrame()
def_RBF_param = def_RBF_param.append(df_RBF.loc[(df_RBF['param_l2reg'].isin(the_param_l2reg_lst))&(df_RBF['param_sigma'
def_RBF_param = def_RBF_param.append(df_RBF.loc[(df_RBF['param_sigma'].isin(the_param_sigma_lst))&(df_RBF['param_l2reg'
def_RBF_param
```

```
The best param_l2reg is 0.0625
The best param_sigma is 0.1
```

	param_degree	param_kernel	param_l2reg	param_offset	param_sigma	mean_test_score	mean_train_score
24	-	RBF	0.1250	-	0.1	0.022885	0.024608
27	-	RBF	0.0625	-	0.1	0.021270	0.023245
27	-	RBF	0.0625	-	0.1	0.021270	0.023245
28	-	RBF	0.0625	-	1	0.063632	0.098843

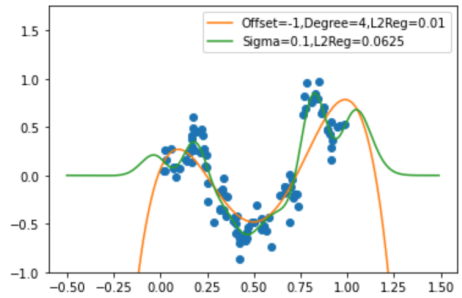
```
df_poly = df_toshow[df_toshow.param_kernel == 'polynomial']

mean_test_score_poly_lst = df_poly.mean_test_score.tolist()
param_degree_lst = df_RBF.param_degree.tolist()
param_offset = df_RBF.param_offset.tolist()

best_param_degree, best_param_l2reg, best_param_offset = df_poly.iloc[np.argmin(mean_test_score_poly_lst), [0, 2, 3]]
print(f'The best param_degree is {best_param_degree}')
print(f'The best param_l2reg is {best_param_l2reg}')
print(f'The best param_offset is {best_param_offset}')

the_param_degree_lst = [3, 4] # param_l2reg = 0.01; param_offset = -1
the_param_l2reg_lst = [0.01, 0.1] # param_degree = 4; param_offset = -1
the_param_offset_lst = [-1, 0] # param_degree = 4; param_l2reg = 0.01

def_poly_param = pd.DataFrame()
def_poly_param = def_poly_param.append(df_poly.loc[(df_poly['param_degree'].isin(the_param_degree_lst))&(df_poly['param
def_poly_param = def_poly_param.append(df_poly.loc[(df_poly['param_l2reg'].isin(the_param_l2reg_lst))&(df_poly['param_c
def_poly_param = def_poly_param.append(df_poly.loc[(df_poly['param_offset'].isin(the_param_offset_lst))&(df_poly['param
def_poly_param
```

```
The best param_degree is 4
The best param_l2reg is 0.01
The best param_offset is -1
```

	param_degree	param_kernel	param_l2reg	param_offset	param_sigma	mean_test_score	mean_train_score
45	3	polynomial	0.01	-1	-	0.068156	0.097027
54	4	polynomial	0.01	-1	-	0.043454	0.060135
51	4	polynomial	0.10	-1	-	0.188090	0.148359
54	4	polynomial	0.01	-1	-	0.043454	0.060135
54	4	polynomial	0.01	-1	-	0.043454	0.060135
55	4	polynomial	0.01	0	-	0.130907	0.168813

Q29

```
## 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= -1
degree = 4
l2reg = 0.01
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 = 0.1
l2reg= 0.0625
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();
```



Q 30

Bayes Decision Function

$$E(y|x) = E((f(x)+\varepsilon)|x) = E(f(x)|x) + E(\varepsilon|x) = f(x)$$

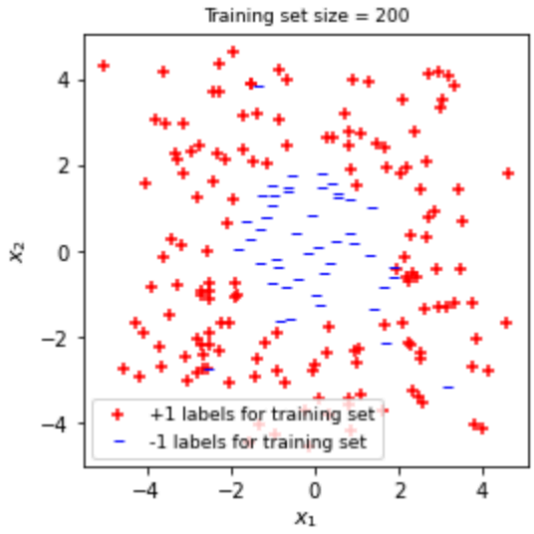
Bayes Risk:

$$var(y|x) = var((f(x)+\varepsilon|x) = var(f(x)|x) + var(\varepsilon|x)$$
$$= var(\varepsilon) = 0.1^2 = 0.01$$

Kernel SVM optional problem

Q31

```
# 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 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();
```



- The data is not linearly separable.
- The data is quadratically separable.
- We can use some RBF kernel to obtain appropriate decision boundary to separate this data.