HW3 Siyong Liu

1

$$max_i f_i(z) \geq f_k(z) \geq f_k(x) + g^T(z-x) = f(x) + g^T(z-x)$$
$$g \in \partial f(x)$$

2

$$\partial J(w) = \begin{cases} 0 & 1 - yw^T x < 0 \\ -yx^T & otherwise \end{cases}$$

4

$$\begin{aligned} \nabla_w J(w) &= \nabla_w\left(\frac{1}{n}\sum \ell(y_i w^\intercal x_i) + \lambda||w||^2\right) \\ &= \frac{1}{n}\sum \nabla_w \ell(y_i w^\intercal x_i) + 2\lambda w) \\ &= \begin{cases} \frac{1}{n}\sum -y_i x_i + 2\lambda w & y_i w^\intercal x_i < 1 \\ undefine & otherwise \end{cases} \end{aligned}$$

5

$$w_{t+1} = w_t - \eta(\lambda w - y_i x)$$
$$w_{t+1} = \begin{cases} (1-\eta_t\lambda)w_t + \eta_t y_i x_i & y_i w^\intercal x_i < 1 \\ (1-\eta_t\lambda)w_t & otherwise \end{cases}$$

6

```
# 6
def to_sparse(l):
  return Counter(l)
```

7

```
# 7
def load_data():
  reviews = load_and_shuffle_data()
  train = reviews[:1500]
  test = reviews[1500:]
  X_train = [x[:-1] for x in train]
  y_train = [x[-1] for x in train]
  X_test = [x[:-1] for x in test]
  y_test = [x[-1] for x in test]
  return X_train, y_train, X_test, y_test

X_train, y_train, X_test, y_test = load_data()
```

8

```
# 8
def pegasos1(X, y, lambda_reg=0.25, max_epoch=30):
    epoch = 0
    w = {}
    t = 0
    order_list = range(len(X))
    while epoch < max_epoch:
        random.shuffle()
        for i in order_list:
            t += 1
            eta = 1 / (t * lambda_reg)
            if y[i] * dotProduct(X[i], w) < 1:
                increment(w, - eta * lambda_reg, w)
                increment(w, eta * y[i], X[i])
            else:
                increment(w, - eta * lambda_reg, w)
        epoch += 1
    return w
```

9

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

```
# 9
```

```
def pegasos2(X, y, lambda_reg=0.1, max_epoch=30, tolerance=1e-2,
useConverge=True):
    epoch = 0
    w = {}
    t = 1
    scale = 1
    order_list = range(len(X))
    while epoch < max_epoch:
        epoch += 1
        prev_sum = sum(w[weight]**2 for weight in w)
        for i in order_list:
            t += 1
            eta = 1 / (t * lambda_reg)
            scale = (1 - eta * lambda_reg) * scale
            if y[i] * scale * dotProduct(w, X[i]) < 1:
                increment(w, eta * y[i] / scale, X[i])
        cur_sum = sum(w[weight]**2 for weight in w)
        if useConverge and np.abs(scale**2 * (prev_sum - cur_sum)) < tolerance:
            break
    for k, v in w.items():
        w[k] = v * scale
    return w
```

10

```
%%time
w1 = pegasos1(X_train, y_train)
>>>
Wall time: 6min 43s
```

```
%%time
w2 = pegasos2(X_train, y_train, useConverge=False)
>>>
Wall time: 5.41 s
```

```
print("w1['friends']: ", w1['friends'])
print("w2['friends']: ", w2['friends'])
>>>
w1['friends']:  0.020533333333333324
w2['friends']:  0.017555165440767868
```

11

```
#11 classification error
def classification_error(w, X, y):
  cnt = 0
  for i in range(len(X)):
    if np.sign(dotProduct(X[i],w)) != y[i]:
      cnt += 1
  return cnt/len(X)

w1_err = classification_error(w1, X_test, y_test)
w2_err = classification_error(w2, X_test, y_test)
print('w1_err: ', w1_err)
```

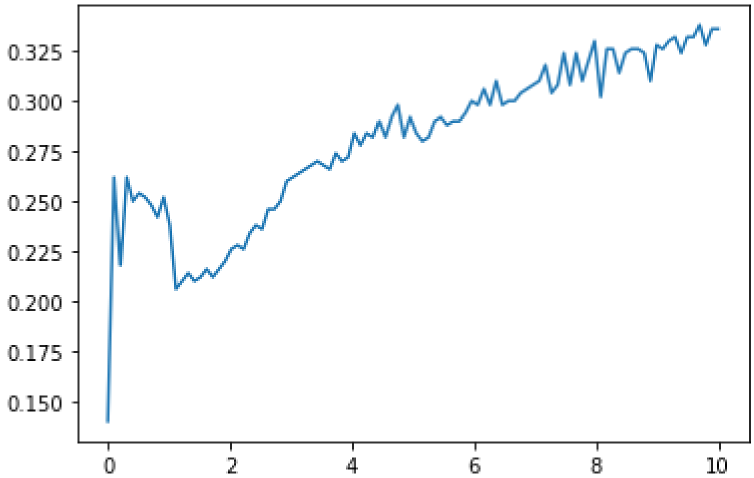
```
print('w2_err: ', w2_err)
>>>
w1_err:  0.246
w2_err:  0.276
```

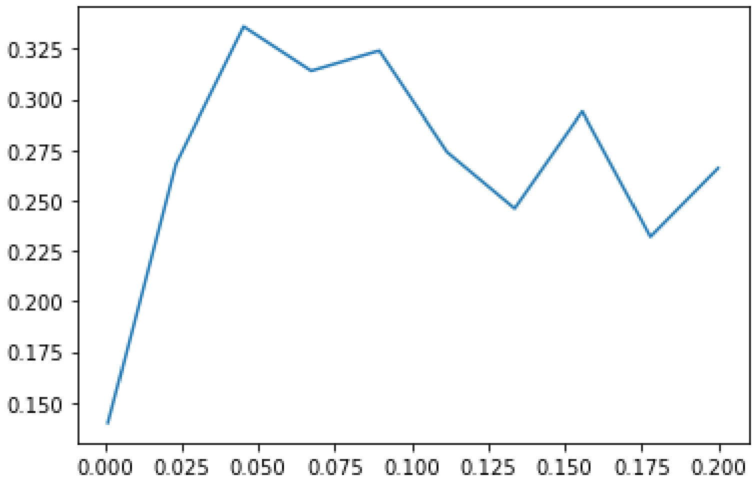
12

```
def test_lambda(lambda_list, X_train, y_train, X_test, y_test):
    err_list = []
    for lambda_reg in lambda_list:
        w = pegasos2(X_train, y_train, lambda_reg)
        err_list.append(classification_error(w, X_test, y_test))
    return err_list
```

```
lambda_list = np.linspace(0.001, 10, 100)
err_list = test_lambda(lambda_list, X_train, y_train, X_test, y_test)
plt.plot(lambda_list, err_list)
```

```
lambda_list = np.linspace(0.001, 0.2, 10)
err_list = test_lambda(lambda_list, X_train, y_train, X_test, y_test)
plt.plot(lambda_list, err_list)
```





best lambda is around [0, 0.025]

15

$$\begin{aligned}
\partial_w J(w) = 2X^T(Xw-y) + 2\lambda w &= 0 \\
X^T X w + \lambda I w &= X^T y \\
(X^T X + \lambda I) w &= X^T y \\
w &= (X^T X + \lambda I)^{-1} X^T y
\end{aligned}$$

16

$$\begin{aligned}
X^T X w + \lambda I w &= X^T y \\
\lambda w &= X^T y - X^T X w \\
w &= \frac{1}{\lambda}(X^T y - X^T X w)
\end{aligned}$$

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

17

$$w = X^T \alpha = \sum_{i=1}^{m} \alpha_i x_i$$

18

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

19

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

20

$$f(x) = x^T w^* = x^T X^T \alpha = k_x^T \alpha$$

21

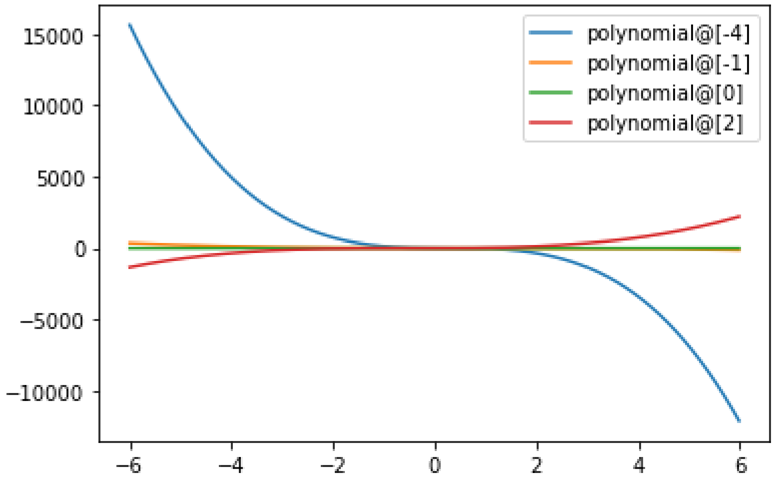
```
def linear_kernel(X1, X2):
    return np.dot(X1,np.transpose(X2))

def RBF_kernel(X1,X2,sigma):
    distance = scipy.spatial.distance.cdist(X1,X2,'sqeuclidean')
    return np.exp(- 0.5 * distance / pow(sigma, 2))

def polynomial_kernel(X1, X2, offset, degree):
    return pow((offset + np.inner(X1, X2)), degree)
```

22

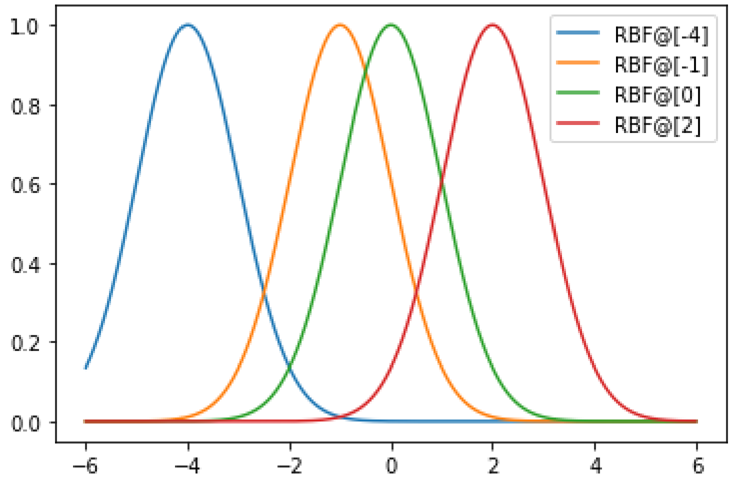
```
array = np.array([[-4, -1, 0, 2]]).T
linear_kernel(array, array)
>>>
array([[16,  4,  0, -8],
       [ 4,  1,  0, -2],
       [ 0,  0,  0,  0],
       [-8, -2,  0,  4]])
```

23

```
# 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()
```



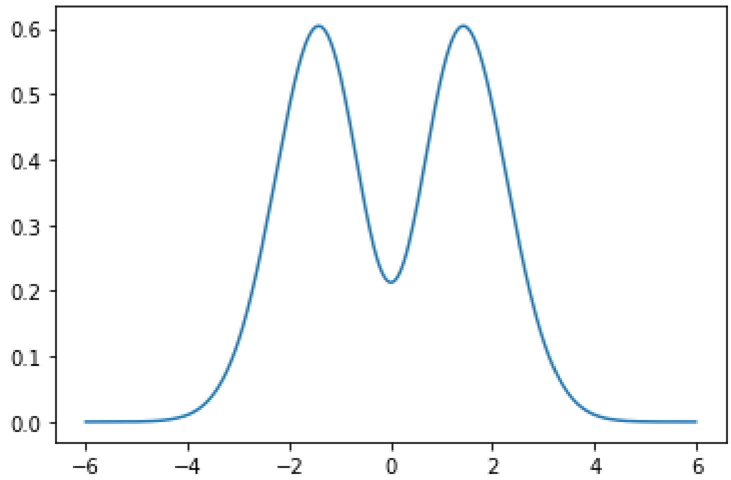

```
y = RBF_kernel(prototypes, xpts, 1)
for i in range(len(prototypes)):
    label = "RBF@"+str(prototypes[i,:])
    plt.plot(xpts, y[i,:], label=label)
plt.legend(loc = 'best')
plt.show()
```



24

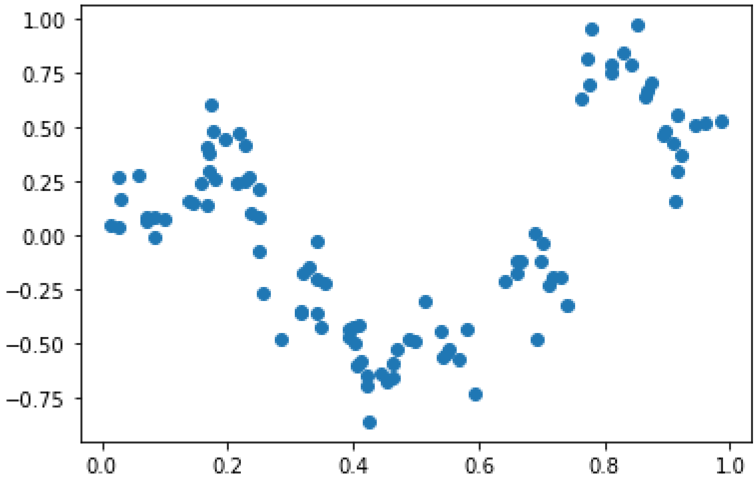
```
def predict(self, X):
    return self.kernel(X, self.training_points) @ self.weights
```

```
rbf_kernel = functools.partial(RBF_kernel, sigma=1)
xpts = np.arange(-6.0, 6, plot_step).reshape(-1,1)
k_machine = Kernel_Machine(rbf_kernel,
    np.array([-1, 0, 1]).reshape(-1, 1),
    np.array([1, -1, 1]).reshape(-1, 1))
plt.plot(xpts, k_machine.predict(xpts))
plt.show()
```



25

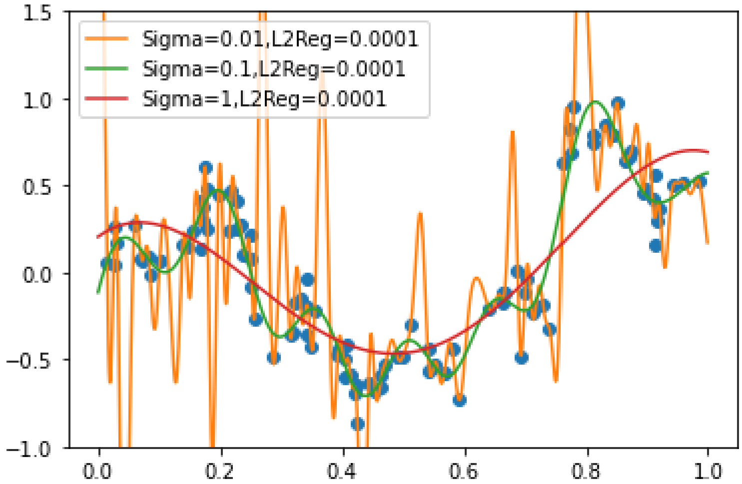
```
plt.scatter(x_train, y_train)
plt.plot()
```



26

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

27

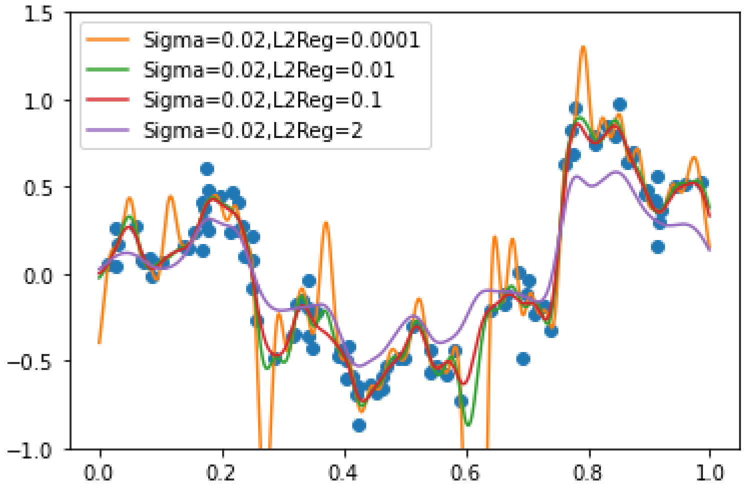


overfit: 0.01

underfit: 1

best: 0.1

28



when $\lambda \rightarrow \infty$, model become underfit