Solutions to HW-2

- Arjun Parasuram Prasad (NetID: ap9334)

Support Vector Machines: SVMs with Pegasos

Ans 1:

Ans 1

$g \in \mathbb{R}^d$ is a subgradient of $f: \mathbb{R}^d \to \mathbb{R}$ if $\forall z$

$$\boxed{f(z) \geq f(x) + g^T(z-x)}$$

$\underbrace{f_1, f_2, f_3, \ldots, f_m}_{\text{convex functions}} : \mathbb{R}^d \to \mathbb{R}.$

$$f(x) = \max_{i=1,2,\ldots,m} f_i(x)$$

Let k be any index for which $f_k(x) = f(x)$ and choose $g \in \partial f_k(x)$

To prove: $g \in \partial f(x)$ also.

Proof:

given that: $g \in \partial(f_k(x))$

& say D_f is the domain of $f(x)$.

Let $D_{f_{k_i}}$ be the domain of f_{k_i} which has subgradient g_{k_i} &

$$f_{k_p}(x) = \max_{i=1\ldots m} f_i(x)$$

Case 1: $z \in D_{f_{k_0}}$ and $x \in D_{f_{k_0}}$

$\Longrightarrow$ for subgradient g_{k_0}, we know that

$$f_{k_0}(z) \geq f_{k_0}(x) + g_{k_0}^T(z-x)$$

as $f_{k_0}(t) = f(t) \quad \forall\, t \in D_{f_{k_0}}$

$\therefore$ we can write this expression as

$$f(z) \geq f(x) + g^T(z-x)$$

$\therefore$ when z and x belong to the same subdomain of f, $\exists$ subgradient $g_{k_0} \in \partial(f(x))$

- Case 2. z and x belong to different subdomains of $f(x)$

Let $z \in D_{f_{k_2}}$ and $x \in D_{f_{k_1}}$.

Let $g_{k_1} \in \partial(f_{k_1}(x))$

f Let $g_{k_2} \in \partial(f_{k_2}(x))$

Then for function f_{k_1} and subgrad. g_{k_1} we know that

$$f_{k_1}(z) \geq f_{k_1}(x) + g_{k_1}^T(z-x)$$

as $z \in D_{f_{k_2}} \Rightarrow f_{k_2}(z) \geq f_{k_1}(z)$

$\therefore$

$$f_{k_2}(z) \geq f_{k_1}(x) + g_{k_1}^T(z-x)$$

for $z \in D_{f_{k_2}}$, $f_{k_2}(z) = f(z)$

for $x \in D_{f_{k_1}}$, $f_{k_1}(x) = f(x)$

Thus, we can rewrite our eq^n as

$$f(z) \geq f(x) + g_{k_1}^T(z-x)$$

$\therefore$ even when z, x belong to 2 different subdomains of f,

$\exists\, g_{k_1}$ such that $g_{k_1} \in \partial(f(x))$

Thus we have that
$\forall\, x, z \in D_f$
if $g_k \in \partial(f(x))$

Hence Proved.

please turn over

Ans 2:

Ans 2 $J(w) = \max\left(0, 1 - y w^T x\right)$

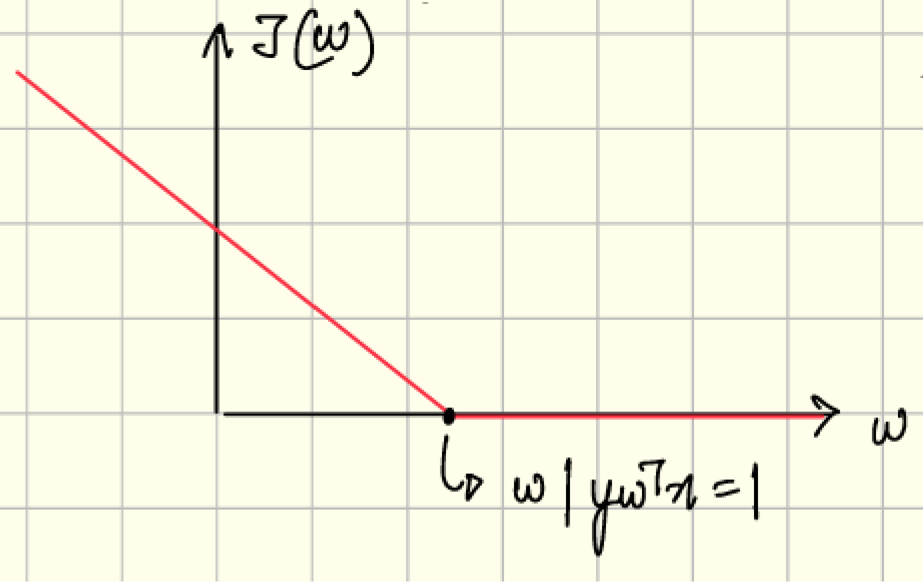
$\Rightarrow$ defining $g \in \partial(J(w))$

we can write

$$\boxed{g = \begin{cases} -yx, & y w^T x < 1 \\ 0, & y w^T x \geq 1 \end{cases}}$$

$[x \in \mathbb{R}^d, w \in \mathbb{R}^d, y \in \mathbb{R}]$

Proving that g is a subgradient for $J(w)$:-



$J(w)$ can be rewritten as

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

Take points w_1 and $w_2 = w_1 + d$. [w_1 & w_2 $\in \underline{\mathcal{D}}_{J(w)}$]
[→ domain

Case 1: $yw_1^T x \geq 1$ and $yw_2^T x \geq 1$

$\xrightarrow{\text{RHS}}$ $J(w_1) + g^T(w_2 - w_1)$

LHS

$= 0 + 0 \geq 0 = J(w_2)$

$\Rightarrow J(w_2) \geq J(w_1) + g^T(w_2 - w_1)$

$\therefore$ g satisfies subgradient conditions for case 1

Case 2: $yw_1^T x \geq 1$ and $yw_2^T x < 1$

RHS:

$J(w_1) + g^T(w_2 - w_1)$

$= 0 + 0 = 0$ [$g = 0$ @ w_1]

LHS:

$J(w_2) = 1 - yw_2^T x > 0$

$\therefore J(w_2) \geq J(w_1) + g(w_2 - w_1)$

Thus g satisfies subgradient cond.s of $J(w)$ for case 2.

Case 3:

$$y w_2^T x \geq 1 \text{ and } y w_1^T x < 1$$

RHS:

$\Rightarrow J(w_1) + g^T(w_2 - w_1)$

$= 1 - y w_1^T x + (-yx)^T (w_2 - w_1)$

$= 1 - \cancel{y w_1^T x} - y w_2^T x + \cancel{y w_1^T x}$ $[g = yx \text{ @ } w_1,$ $(-yx)^T w = -y w^T x]$

$= 1 - y w_2^T x < 0$

(since $y w_2^T x > 1 \Rightarrow 1 - y w_2^T x < 0$)

LHS:

$J(w_2) = 0 > 1 - y w_2^T x$

$\Rightarrow$

$$J(w_2) \geq J(w_1) + g^T(w_2 - w_1)$$

$\therefore$ g satisfies subgradient conditions for $J(w)$ for case 3.

Case 4: $y w_1^T x < 1$ and $y w_2^T x < 1$

$\Rightarrow$ RHS $= J(w_1) + g^T(w_2 - w_1)$

$$RHS = (1 - y w_1^T x) + (-yx)^T (w_2 - w_1)$$

$$[g = -yx \text{ @ } w_1]$$

$$[-y^T x^T w = -y^T w^T x]$$

$\Rightarrow$

$$RHS = 1 - \cancel{y w_1^T x} - y w_2^T x + \cancel{y w_1^T x}$$

$$= 1 - y w_2^T x = J(w_2) = LHS$$

$\therefore$

$$J(w_2) \geq J(w_1) + g^T (w_2 - w_1)$$

Hence g satisfies subgradient conditions for $J(w)$ for case 4.

Since g satisfies subgradient condition

$$J(z) = J(w) + g^T (z - x)$$

$\forall$ z, x in the domain of $J(w)$

$\therefore$ we have proved that

$$g \in \partial(J(w))$$

please turn over

Ans 3:

SVM with Pegasos algorithm.

Ans 3

$$J_i(w) = \frac{\lambda}{2}\|w\|^2 + \max\{0, 1 - y_i w^T x_i\}$$

The above expression is not defined for

$$(x_i, y_i) \in \{(x_i, y_i) \mid y_i w^T x_i = 1\}$$

$\forall\ w$, s.t. $y_i w^T x_i \neq 1$, we have

$$\nabla_w (J_i(w)) = \nabla_w\left(\frac{\lambda}{2} w^T w\right) + \nabla_w\left(\max\{0, 1 - y_i w^T x_i\}\right)$$

$$= \begin{cases} \lambda w + (-y_i x_i), & 0 > 1 - y_i w^T x_i \\ \lambda w + 0 & 0 < 1 - y_i w^T x_i \end{cases}$$

$$= \begin{cases} \lambda w - y_i x_i, & 0 > 1 - y_i w^T x_i \\ \lambda w & 0 < 1 - y_i w^T x_i \end{cases}$$

please turn over

Ans 4:

Ans 4

$$J(w) = \frac{\lambda}{2} w^T w + \max\{0, 1 - y_i w^T x_i\}$$

$$\partial(J(w)) = \underbrace{\lambda w}_{\partial(f_1(w))} + \underbrace{\partial\left[\max\{0, 1 - y_i w^T x_i\}\right]}_{\partial(f_2(w))}$$

we know that if

$$f = f_1 + f_2 + f_3 + \cdots f_n$$

then $\partial(f) = \partial(f_1) + \partial(f_2) + \cdots + \partial(f_n)$

$$\therefore \partial(J(w)) = \partial(f_1(w)) + \partial(f_2(w))$$

$$\nabla_w (f_1(w)) \in \partial(f_1(w))$$

[as $f_1(w)$ is convex and differentiable at all points]

$\Rightarrow$ a subgradient $g_1(w)$ of $f_1(w)$ is

$$\boxed{g_1(w) = \lambda w}$$

we also know that as subgradient of $f_2(w)$ is

$$g_2 = \begin{cases} -x_i y_i, & y_i w^T x < 1 \\ 0, & y_i w^T x \geq 1 \end{cases}$$

(refer proof in Ans 2 attached with this q. as well)

$$\therefore \quad \partial(J(w)) = \partial(f_1(w)) + \partial(f_2(w))$$

$\Rightarrow$ a subgradient of $J(w)$ is

$$g = \begin{cases} \lambda w - x_i y_i, & y_i w^T x < 1 \\ \lambda w, & y_i w^T x \end{cases}$$

// Hence proved.

please turn over

Ans 5:

```
def gen_sparse_bag_of_words(list_of_words):
    return Counter(list_of_words)
```

`please turn over`

Ans 6:

```
X = list(map(lambda review : gen_sparse_bag_of_words(review[0 : len(review)
- 1]), data))
y = list(map(lambda review_label : review_label[-1], data))

X_train = X[0:1500]
X_test = X[1500:]
y_train = y[0:1500]
y_test = y[1500:]
```

`please turn over`

Ans 7:

```
def pegasos_v1(X_train, y_train, lambda_reg, epochs, verbose = True,
tolerance = 0.001):
    W = {}
    n = len(X_train)
    err = np.inf

    for t in range(1, n*epochs+1):
        if(t%epochs == 0):
            # reshuffle the data
            data = list(zip(X_train, y_train))
            random.shuffle(data)
            X_train, y_train = zip(*data)

        eta = 1/(lambda_reg*t)
        Xj = X_train[(t-1)%n]
        yj = y_train[(t-1)%n]

        margin = yj * dotProduct(W, Xj)

        if margin >= 1:
            for i, v in Xj.items():
                W[i] = (1 - (eta*lambda_reg))*W.get(i, 0)
        else:
            for i, v in Xj.items():
                W[i] = (1 - (eta*lambda_reg))*W.get(i, 0) + v*eta*yj


        if (t%n == 0) and (verbose == True):
            clf_error = classification_error(X_train, y_train, W)
            print('------------------------------------')
            print(f'epoch: {t/n}')
            print(f'classification error is: {clf_error}')
            print(f'W.size is : {len(W)}')
            if abs(err - clf_error) <= tolerance:
                break
            err = clf_error

    return W
```

please turn over

Ans 8:

```
def pegasos_v2(X_train, y_train, lambda_reg, epochs, verbose = True,
tolerance = 0.001):
    W = {}
    s = 1
    n = len(X_train)
    err = np.inf

    for t in range(2, epochs*n + 1):
        if(t%epochs == 0):
            # reshuffle the data
            data = list(zip(X_train, y_train))
            random.shuffle(data)
            X_train, y_train = zip(*data)

        eta = 1 / (lambda_reg*t)
        Xj = X_train[(t-1)%n]
        yj = y_train[(t-1)%n]
        margin = yj * dotProduct(W, Xj) * s

        s = (1 - eta*lambda_reg)*s
        if margin < 1:
            increment(W, (1/s)*eta*yj, Xj)

        if(t%n == 0 and verbose == True):
            clf_error = classification_error(X_train, y_train, scale(W, s))
            print('-----------------------------------')
            print(f'epoch: {t/n}')
            print(f'classification error is: {clf_error}')
            print(f'W.size is : {len(W)}')
            if abs(err - clf_error) <= tolerance:
                break
            err = clf_error
    return scale(W, s)
```

please turn over

Ans 9:

```
print('epochs = 2')
begin = time.time()
pegasos_v1(X_train, y_train, lambda_reg= 0.01, epochs= 2, verbose= False,
tolerance= 0.001)
print(f'training time - pegasos_v1: {time.time() - begin}')
begin = time.time()
pegasos_v2(X_train, y_train, lambda_reg= 0.01, epochs= 2, verbose= False,
tolerance= 0.001)
print(f'training time - pegasos_v2: {time.time() - begin}')

print('epochs = 200. Tolerance has been disabled')
begin = time.time()
pegasos_v1(X_train, y_train, lambda_reg= 0.01, epochs= 200, verbose= False,
tolerance= -1)
print(f'training time - pegasos_v1: {time.time() - begin}')
begin = time.time()
pegasos_v2(X_train, y_train, lambda_reg= 0.01, epochs= 200, verbose= False,
tolerance= -1)
print(f'training time - pegasos_v2: {time.time() - begin}')
```

```
--------------------------------------------------------------------------------
A9
epochs = 2
training time - pegasos_v1: 1.4444091320037842
training time - pegasos_v2: 1.3131685256958008
epochs = 200. Tolerance has been disabled
training time - pegasos_v1: 29.29502511024475
training time - pegasos_v2: 15.590850353240967
```

please turn over

Ans 10:

```
def classification_error(X, y, W):
    y_pred = get_predictions(X, W)
    return sum(np.array(y) != np.array(y_pred))/(len(y_pred))
```

```
print(f"validation set classification error using W_pegasos_v2: \
    {classification_error(X_test, y_test, W_pegasos_v2)}")
```

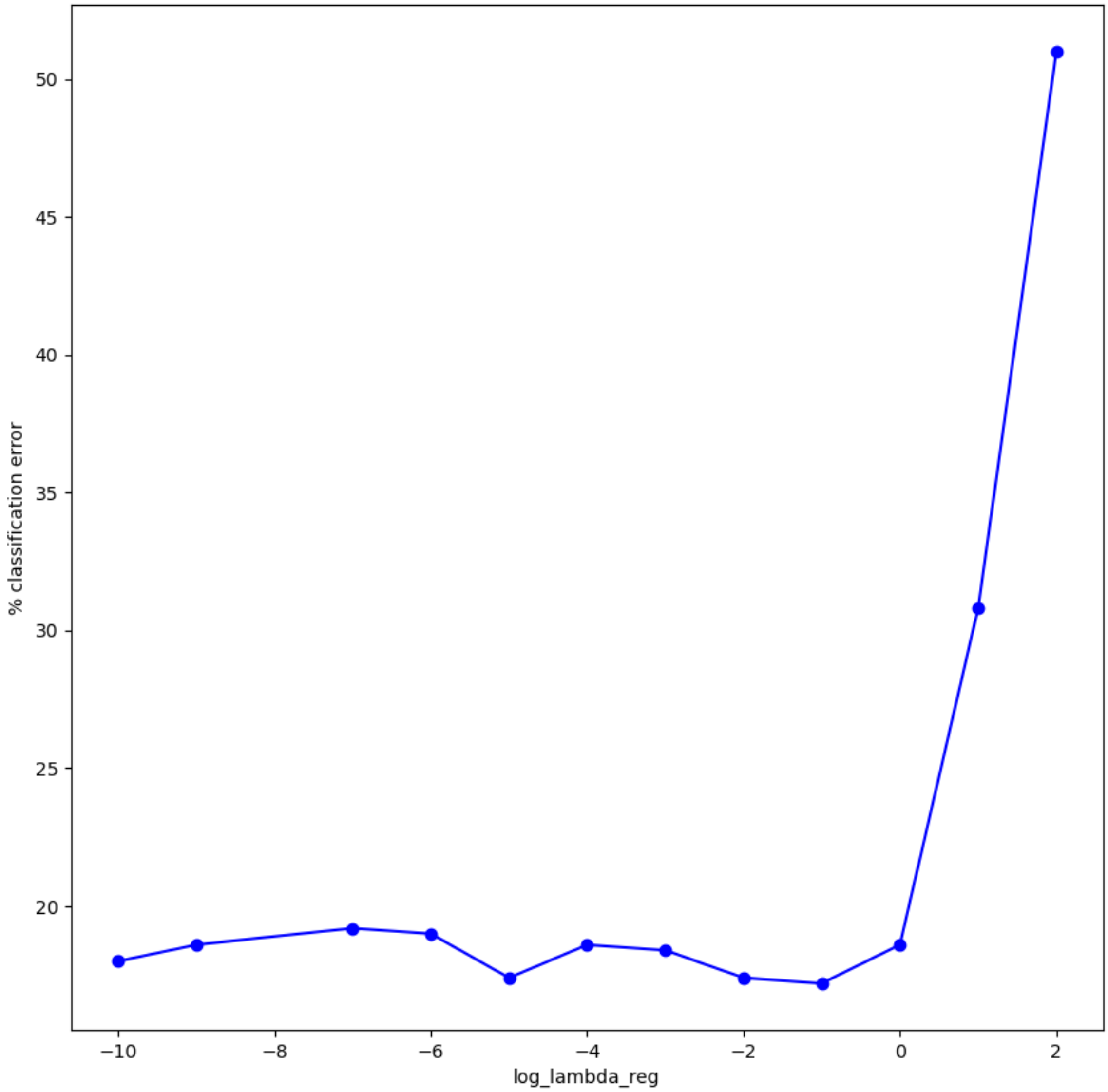
```
validation set classification error using W_pegasos_v2: 0.178
```

please turn over

Ans 11:

```
def plot_lambda_vs_error(X_train, y_train, X_test, y_test, lambda_reg_list
= [], epochs = 100):
    error_list = []
    for lambda_reg in lambda_reg_list:
        W = pegasos_v2(X_train, y_train, lambda_reg= lambda_reg, epochs=
epochs, tolerance = -1, verbose=False)
        error_list.append(classification_error(X_test, y_test, W)*100)
    plt.figure(figsize=(10, 10))
    plt.plot([np.log10(i) for i in lambda_reg_list], error_list, 'bo-',
label='Lambda vs %Classification Error')
    plt.xlabel("log_lambda_reg")
    plt.ylabel("% classification error")
    plt.show()
```

```
plot_lambda_vs_error(X_train, y_train, X_test, y_test, \
    [1e-10, 1e-9, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100],
epochs=100)
```

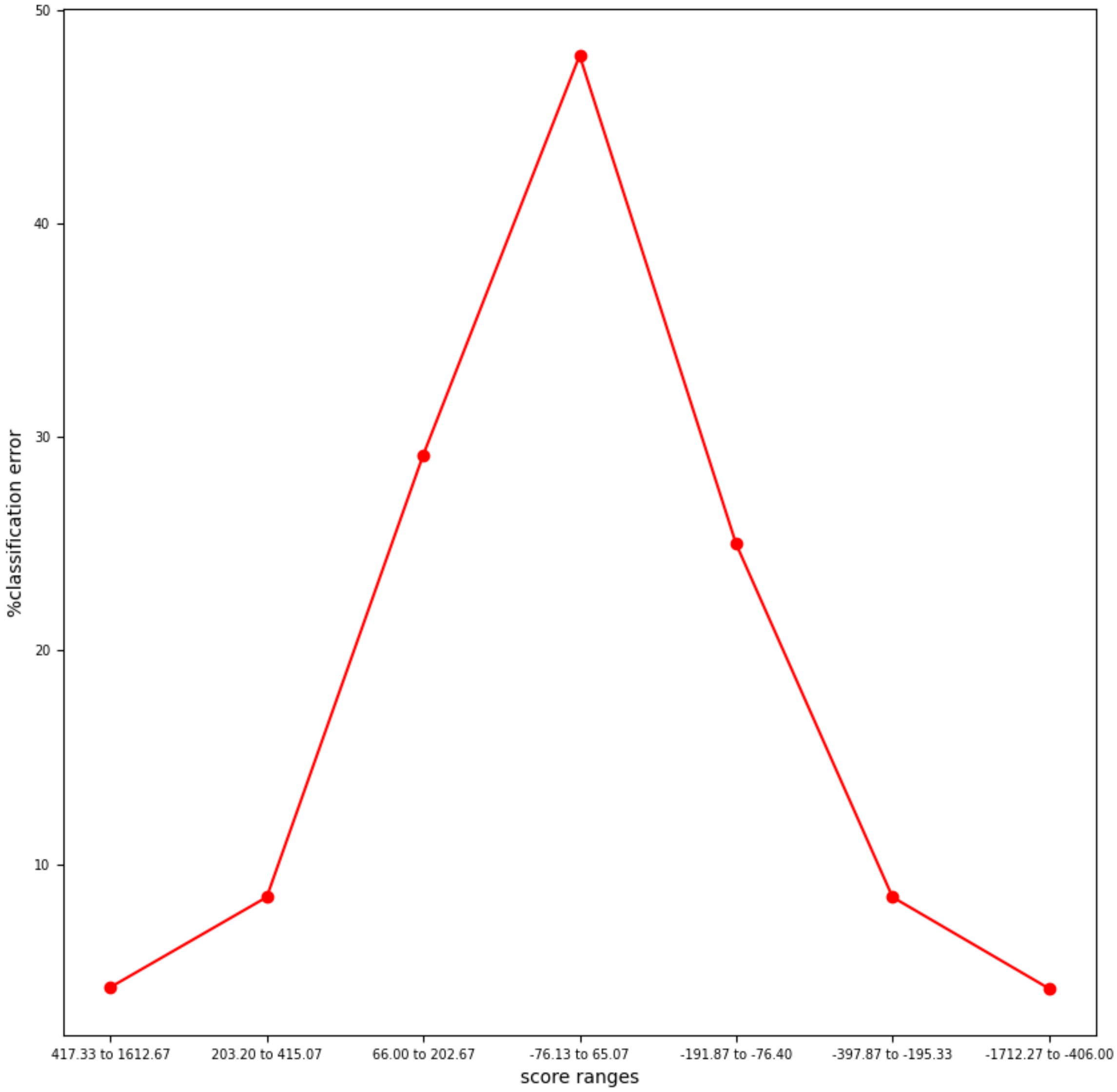


please turn over

Ans 12:

```
def plot_score_grps_vs_percentage_error(X, y, W, bins):
    scores = [dotProduct(i, W) for i in X]
    coupled_mat = np.column_stack((y, scores))
    coupled_mat = coupled_mat[coupled_mat[:, 1].argsort()[::-1]]
    range_start = 0
    range_end = 0
    score_ranges = []
    classification_errors = []

    for i in np.linspace(0, coupled_mat.shape[0], bins, dtype=int):
        if i == range_end:
            continue
        range_start = range_end
        range_end = i
        mat = coupled_mat[range_start:range_end, :]
        score_ranges.append(f"{np.min(mat[:, 1]):.2f} to {np.max(mat[:, 
1]):.2f}")
        classification_errors.append(100 * (np.sum(mat[:, 0] != 
np.sign(mat[:, 1]))) / mat.shape[0])

    plt.figure(figsize=(10, 10))
    plt.plot(score_ranges, classification_errors, 'ro-', label='score 
ranges vs classification erros')
    plt.xlabel("score ranges")
    plt.ylabel("%classification error")
    plt.xticks(fontsize=7)
    plt.yticks(fontsize=7)
    plt.show()
```

```
W_pegasos_v2 = pegasos_v2(X_train, y_train, lambda_reg= 1e-4, epochs= 50, 
verbose= False, tolerance= 1e-3)
plot_score_grps_vs_percentage_error(X_test, y_test, W_pegasos_v2, bins= 8)
```



Observations

- Classification error is relatively low for those score ranges which have high magnitude, i.e. abs(score) is high.
- Classification error is relatively high for those score ranges which have lower magnitude of scores, i.e. abs(score) is low

please turn over

Kernel Methods

Ans 13:

```
### 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
    """
    sq_euclidean_dist = scipy.spatial.distance.cdist(X1, X2, 'sqeuclidean')
    return np.exp((-1)*(1/(2*sigma*sigma))*sq_euclidean_dist)

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

please turn over

Ans 14:

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

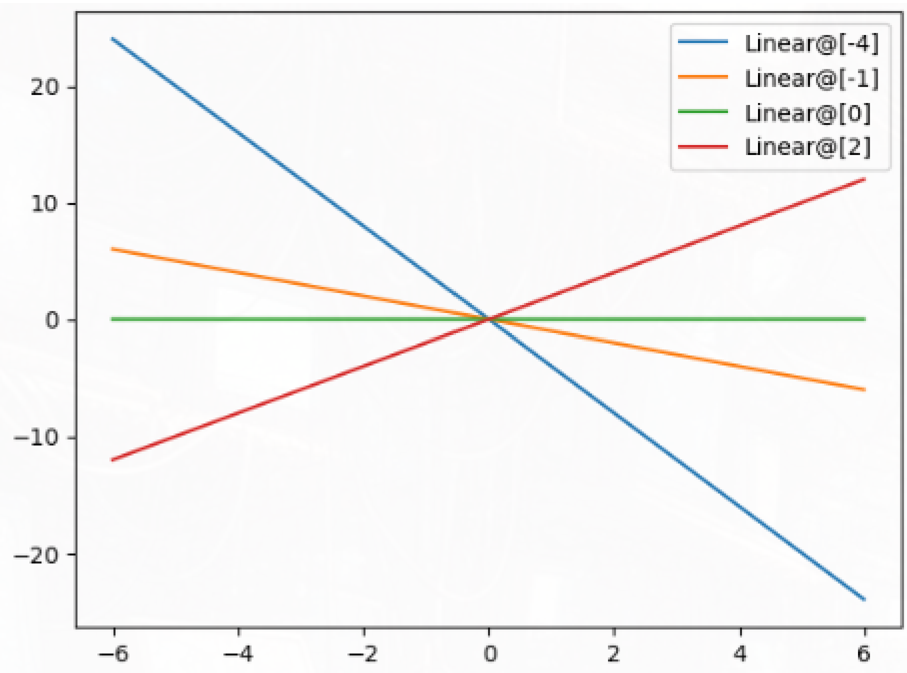
```
array([[16,  4,  0, -8],
       [ 4,  1,  0, -2],
       [ 0,  0,  0,  0],
       [-8, -2,  0,  4]])
```

`please turn over`

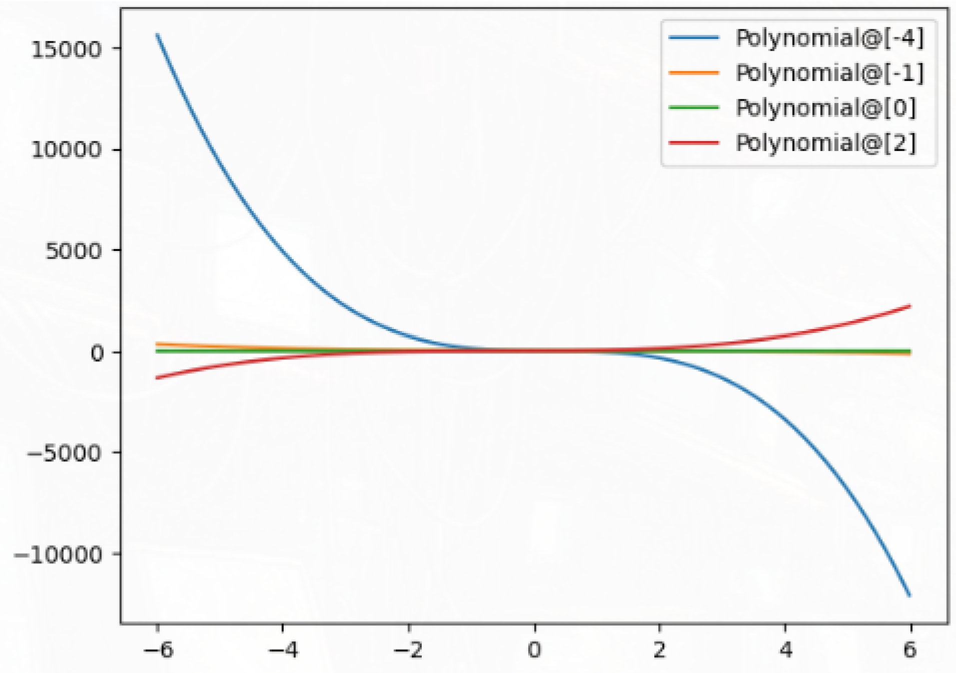
Ans 15:

```
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 kernel
y = polynomial_kernel(prototypes, xpts, offset=1, degree=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 kernel
y = RBF_kernel(prototypes, xpts, sigma= 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()
```

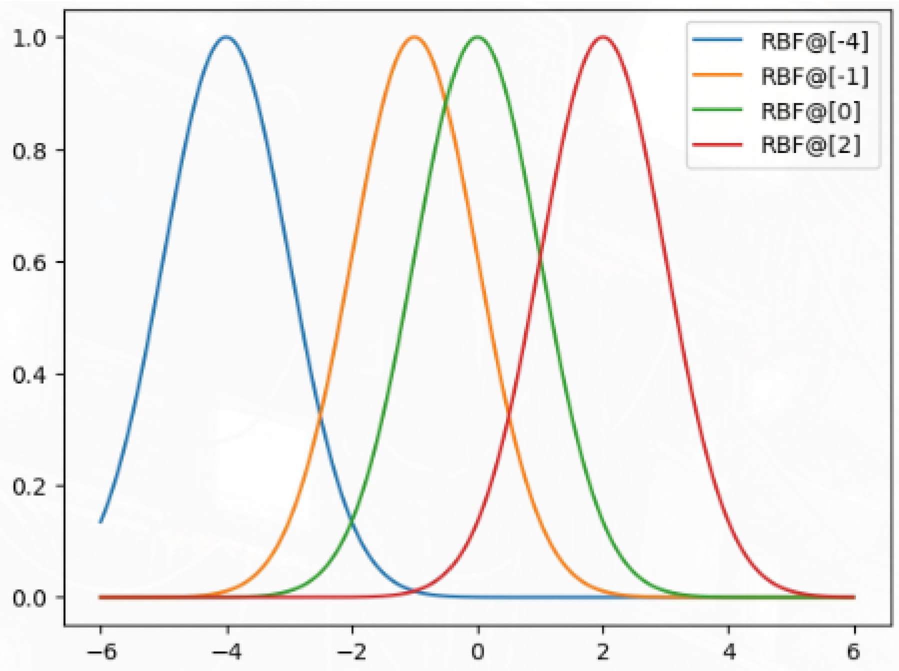
- set of functions x → klinear(x0,x) for x0 ∈ DX and for x ∈ [-6,6]



- set of functions x → kpoly(1,3)(x0,x) for x0 ∈ DX and for x ∈ [-6,6]



- set of functions x → kRBF(1)(x0,x) for x0 ∈ DX and for x ∈ [-6,6]

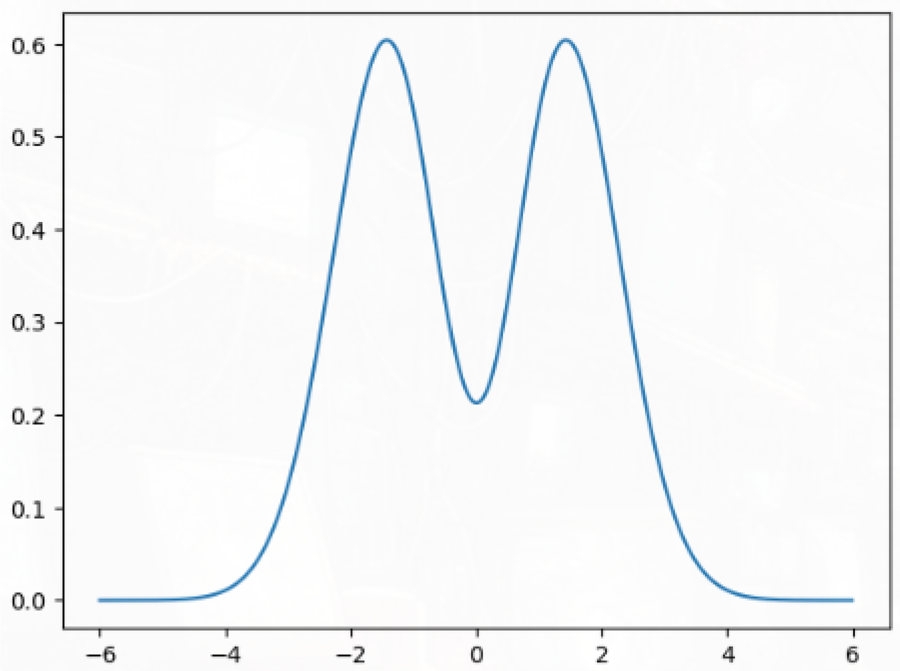


`please turn over`

Ans 16:

```
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)
        """
        K = self.kernel(X, self.training_points)
        return np.dot(K,self.weights)
```

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

Fx = Kernel_Machine(kernel= (lambda i, j : RBF_kernel(X1 = i, X2 = j,
sigma=1)), \
    training_points= prototypes, weights=weights)

plot_step = .01
xpts = np.arange(-6.0, 6, plot_step).reshape(-1,1)
plt.plot(xpts, Fx.predict(xpts))
plt.show()
```



`please turn over`

Logistic Regression

Ans 17:

Ans 17

To prove: Show that ERM with logistic loss and MLE with a Bernoulli response distribution and the logistic link function will produce the same solution for w.

ERM with logistic loss

$x \in \mathbb{R}^d, \quad w \in \mathbb{R}^d, \quad Y_{\pm} = \{+1, -1\}$

$D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \ldots\ldots (x^{(n)}, y^{(n)})\}$

$$\mathcal{F}_{score} = \{x \mapsto x^T w \mid w \in \mathbb{R}^d\}$$

$$\hat{y} = \text{sign}\{x^T w\}$$

logistic loss function

$$l_{logistic}(y, w) = \log(1 + \exp(-y w^T x))$$

$$\Rightarrow \hat{R}_n = \frac{1}{n} \sum_{i=1}^{n} l_{logistic}(f(x^{(i)}), y^{(i)})$$

$$= \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp(-y^{(i)} w^T x^{(i)})\right)$$

$$\Rightarrow \hat{R}_n = \frac{1}{2n} \sum_{i=1}^{n} \Big[(1+y^{(i)}) \log(1+\exp(-w^T x^{(i)})) + (1-y^{(i)}) \log(1+\exp(-w^T x^{(i)}))\Big]$$

minimizing $\hat{R}_n \Rightarrow$

$$\hat{f} \in \arg\min_{f} R_n(f)$$

$$\Rightarrow \hat{\omega} \in \arg\min_{\omega \in \mathbb{R}^d} R_n(\omega)$$

$$\Rightarrow \boxed{\hat{\omega} \in \arg\min_{\omega \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp\left(-\underbrace{y^{(i)} \omega^T x^{(i)}}_{\text{margin}}\right)\right)}$$

$\Updownarrow$ (proof done in assignment 1 as well.

$$\Rightarrow \hat{\omega} \in \arg\min_{\omega \in \mathbb{R}^d} \left(\frac{1}{n} \sum_{i=1}^{n} \left[\frac{1}{2}(1 - y^{(i)}) \log\left(1 + \exp\left(-\omega^T x^{(i)}\right)\right) + \frac{1}{2}(1 + y^{(i)}) \log\left(1 + \exp\left(\omega^T x^{(i)}\right)\right) \right] \right)$$

eqn 1

MLE with a Bernoulli response distribution and the logistic link function.

$$\left[\begin{array}{l} x \in \mathbb{R}^d,\ y_{\pm} \in \{-1, 1\},\ \omega \in \mathbb{R}^d \\ D = \{(x^{(1)}, y^{(1)})\ (x^{(2)}, y^{(2)}) \ldots\ldots (x^{(n)}, y^{(n)})\} \end{array}\right]$$

$$p(y=1 \mid x; \omega) = \frac{1}{1+\exp(-x^T\omega)}$$

$$p(y=-1 \mid x; \omega) = 1 - p(y=1 \mid x; \omega)$$

$$= 1 - \frac{1}{1+\exp(-x^T\omega)}$$

$$\Rightarrow p(y=-1 \mid x; \omega) = \frac{1}{1+\exp(x^T\omega)}$$

Likelihood function.

$$\Rightarrow L(\omega) = \prod_{i=1}^{n} p\left(y^{(i)} \mid x^{(i)}; \omega\right)$$

$$= \prod_{i=1}^{n}\left[\left(\frac{1}{1+\exp(-x^T\omega)}\right)^{\left(\frac{1+y^{(i)}}{2}\right)}\left(\frac{1}{1+\exp(x^T\omega)}\right)^{\left(\frac{1-y^{(i)}}{2}\right)}\right]$$

$\Rightarrow$ the log likelihood function is:

$$\log L(\omega) = \frac{1}{2}\sum_{i=1}^{n}\Big[(1+y^{(i)})\log\left(\left(1+\exp(-x^T\omega)\right)^{-1}\right)$$

$$+$$

$$(1-y^{(i)})\log\left(\left(1+\exp(x^T\omega)\right)^{-1}\right)\Big]$$

[NOTE:

$$x^T w = w^T x \quad \text{as } x \in \mathbb{R}^d, w \in \mathbb{R}$$

]

The negative log likelihood can be written as:

$$NLL(w) = -\log L(w)$$

$$= \frac{1}{2} \sum_{i=1}^{n} \left[(1 - y^{(i)}) \log(1 + \exp(-w^T x)) + (1 + y^{(i)}) \log(1 + \exp(w^T x)) \right]$$

$\therefore$ minimizing the $NLL(w)$

$$\Rightarrow \hat{w} \in \arg\min_{w \in \mathbb{R}^d} NLL(w)$$

$$\Rightarrow \hat{w} \in \arg\min_{w \in \mathbb{R}^d} \left(\frac{1}{n}\right) NLL(w)$$

$$\boxed{\Longrightarrow \hat{w} \in \underset{w \in \mathbb{R}^d}{\arg\min} \left(\frac{1}{n} \left(\sum_{i=1}^{n} \left[(1-y^{(i)}) \log\left(1+\exp(-w^T x)\right) + (1+y^{(i)}) \log\left(1+\exp(w^T x)\right) \right] \right) \right)}$$

eqn (2)

This is the same as eqn (1)

$\therefore$ since the optimization problem is same in both eqn.s (1) and (2)

$\therefore$ $\hat{w}$ s produced as a solution of eqn (1) and (2) will also be the same.

Hence proved.

please turn over

Ans 18:

Ans 18

Decision boundary of logistic regression

$$= \log \frac{p(y=1 \mid x; w)}{p(y=-1 \mid x; w)}$$

$$= \log \left(\frac{\left(1 / 1+\exp(-x^T w) \right)}{\left(1 / 1+\exp(x^T w) \right)} \right)$$

$$= \log \left(\frac{1+\exp(x^T w)}{1+\exp(-x^T w)} \right)$$

$$= \log \left(\frac{1+\exp(x^T w)}{\left(1+\exp(x^T w)\right) / \exp(x^T w)} \right)$$

$$= \underline{\underline{x^T w}}$$

we want margin > 0 for correct classification

$\therefore$ if $y = 1$

then $y \cdot (x^T w) > 0$

$\Rightarrow x^T w > 0$

if $y = -1$

then

$y \cdot (x^T w) > 0$

$\Rightarrow x^T w < 0$

from these 2 cond-s we can see that

$\boxed{x^T w = 0}$

is the decision boundary of log. reg.

This eq^n is scale invariant

$$x^T (cw) = c (x^T w) = 0$$

$$\Updownarrow$$

$$x^T w = 0 \quad \forall c \in \mathbb{R}.$$

please turn over

Ans 19:

Ans 19

$$\log L(c\hat{\omega}) = -\sum_{i=1}^{n}\left[\left(\frac{1+y^{(i)}}{2}\right)\log\left(1+\exp\left(-x^{(i)T}(c\omega)\right)\right) + \left(\frac{1-y^{(i)}}{2}\right)\log\left(1+\exp\left(x^{(i)T}(c\hat{\omega})\right)\right)\right]$$

$$= -\sum_{i=1}^{n}\log\left(1+\exp\left(-y^{(i)}x^{(i)T}(c\hat{\omega})\right)\right)$$

$$\frac{\partial}{\partial c}\log L(c\hat{\omega}) = -\sum_{i=1}^{n}\frac{\partial}{\partial c}\left[\log\left(1+\exp\left(-y^{(i)}x^{(i)T}(c\hat{\omega})\right)\right)\right]$$

$$= -\sum_{i=1}^{n}\frac{-y^{(i)}x^{(i)T}\hat{\omega}}{1+\exp\left(y^{(i)}x^{(i)T}(c\hat{\omega})\right)}$$

$$\boxed{\frac{\partial}{\partial c}\log\left(L(c\hat{\omega})\right) = \sum_{i=1}^{n}\frac{y^{(i)}x^{(i)T}\hat{\omega}}{1+\exp\left(y^{(i)}x^{(i)T}(c\hat{\omega})\right)}}$$

for linearly seperable data.

$$y^{(i)} x^{(i)T} \hat{\omega} > 0$$

$$\therefore \frac{\partial}{\partial c} \log(L(c\hat{\omega})) > 0$$

$\therefore$ The log likelihood (and therefore the likelihood) always increases as c increases

Therefore MLE is not well defined in this case as $\hat{\omega}$ can be scaled to make the likelihood arbitrarily large

please turn over

Ans 20:

Regularized Logistic Regression

Ans 20

$$J_{logistic}(w) = \frac{1}{n}\sum_{i=1}^{n} \overbrace{\log\left(1+\exp\left(-y^{(i)} w^T x^{(i)}\right)\right)}^{J_1} + \overbrace{\lambda \|w\|_2^2}^{J_2}$$

$$J_{logistic} : \mathbb{R}^n \longrightarrow \mathbb{R}.$$

dom $J_{logistic} \equiv \mathbb{R}^n$ and is a convex set.

checking convexity of $J_1 \Longrightarrow$

$$J_1(w) = \log(1+\exp(-y w^T x))$$
$$= \log(1+\exp(-y x^T w))$$

Let $-yx^T w = t$, $(t \in \mathbb{R})$

$$\rightarrow J(t) = \log(1+\exp(t))$$

$J(t) = \log(1 + \exp(t))$

→ $\exp(t)$ is a convex function
→ 1 is a convex function.

$J(t) = \omega_1 (1) + \omega_2 (\exp(t))$

$[\omega_1 = 1, \omega_2 = 1]$

(from 3.2.1 || Rosenberg's notes on optimization)

$\therefore$ $J(t)$ is a convex function.

⇒ $\boxed{J_1(\omega) = \log(1 + \exp(-y\omega^T x)) \text{ is convex.}}$

$\boxed{J_2(\omega) = \lambda \|\omega\|_2^2 \text{ is convex } [\lambda \geq 0]}$

(Every norm on $\mathbb{R}^n$ is convex)

($J_2(\omega) = \omega_1 \|\omega\|_2^2$ $[\omega_1 = \lambda, \omega_1 \geq 0]$)

$$J(\omega) = \frac{1}{n}\left(\sum_{i=1}^{n} (1)(J_1^{(i)}(\omega)) + (1)(J_2^{(i)}(\omega))\right)$$

$$= \left(\frac{1}{n}\right) J_1^1(\omega) + \left(\frac{1}{n}\right) J_1^2(\omega) \cdots + \left(\frac{1}{n}\right) J_1^n(\omega)$$

$$+ \left(\frac{1}{n}\right) J_2^1(\omega) + \left(\frac{1}{n}\right) J_2^2(\omega) - - + \left(\frac{1}{n}\right) J_2^n(\omega)$$

[where $J_1^{(i)}(\omega)$ and $J_2^{(i)}(\omega)$ have $x = x^{(i)}$ and $y = y^{(i)}$]

$1/n \geq 0$ as $n \geq 0$

Therefore $J(\omega)$ is a convex function.

(from 3.2.1 || Rosenberg's notes on optimization)

please turn over

Ans 21:

```
def f_objective(theta, X, Y, l2_param=1):
    '''
    Args:
        theta: 1D numpy array of size num_features
        X: 2D numpy array of size (num_instances, num_features)
        y: 1D numpy array of size num_instances
        l2_param: regularization parameter

    Returns:
        objective: scalar value of objective function
    '''

    n = X.shape[0]

    z = X @ theta

    logistic_loss = (1/n) * np.sum(np.logaddexp(0, -Y * z))

    regularization_loss = l2_param * (theta.T @ theta)

    return logistic_loss + regularization_loss
```

please turn over

Ans 22:

```
def fit_logistic_reg(X, y, objective_function, l2_param=1):
    '''
    Args:
        X: 2D numpy array of size (num_instances, num_features)
        y: 1D numpy array of size num_instances
        objective_function: function returning the value of the objective
        l2_param: regularization parameter

    Returns:
        optimal_theta: 1D numpy array of size num_features
    '''
    theta = np.zeros(X.shape[1])
    f = lambda i : objective_function(i, X, y, l2_param)


    optimizer_obj = minimize(f, theta)
    return optimizer_obj.x
```

```
X_train, y_train, X_val, y_val = load_data()

X_train, X_val = std_scaler(X_train, X_val)

X_train = np.hstack((X_train, np.ones((X_train.shape[0], 1))))  # Add bias term
X_val = np.hstack((X_val, np.ones((X_val.shape[0], 1))))

y_train = np.where(y_train == 0, -1, y_train)
y_val = np.where(y_val == 0, -1, y_val)

print(f'X_train.shape: {X_train.shape}')
print(f'y_train.shape: {y_train.shape}')
print(f'X_val.shape: {X_val.shape}')
print(f'y_val.shape: {y_val.shape}')

theta = fit_logistic_reg(X_train, y_train, f_objective, 0.01)

print(f'theta.shape: {theta.shape}')
print(f'classification error - training set: {classification_error(X_train, y_train, theta)}')
print(f'classification error - validation set: {classification_error(X_val, y_val, theta)}')
```

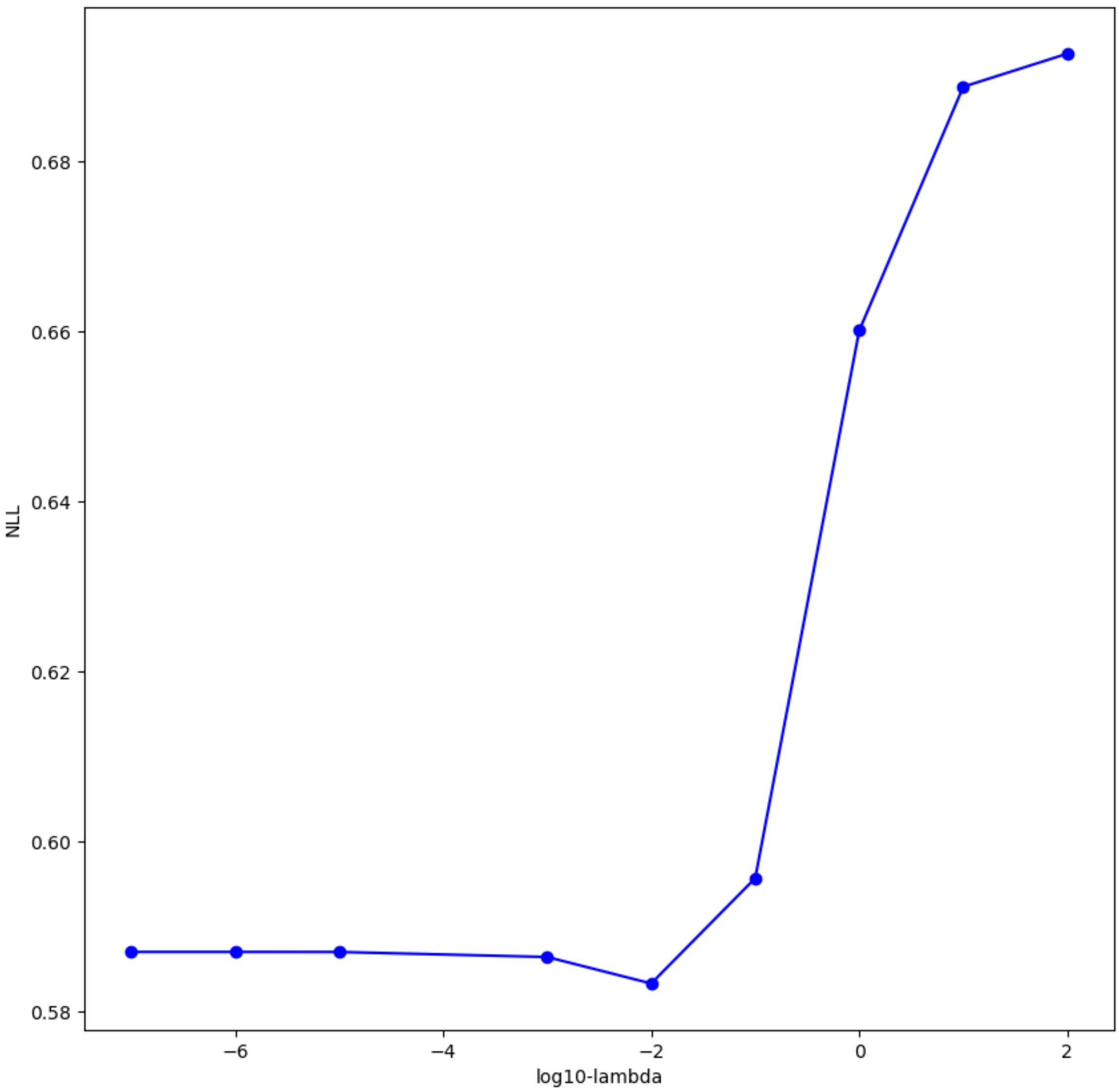
```
X_train.shape: (1600, 21)
y_train.shape: (1600,)
X_val.shape: (400, 21)
y_val.shape: (400,)
theta.shape: (21,)
classification error - training set: 0.255
classification error - validation set: 0.26
```

please turn over

Ans 23:

```
def plot_lambda_reg_vs_log_likelihood(X_train, y_train, X_test, y_test, \
    lambda_reg = [1e-7, 1e-6, 1e-5, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2]):
    log_likelihoods = []
    for l in lambda_reg:
        theta = fit_logistic_reg(X_train, y_train, f_objective, l)
        log_likelihoods.append(log_likelihood(theta, X_test, y_test))
    plt.figure(figsize=(10, 10))
    plt.plot([np.log10(i) for i in lambda_reg], log_likelihoods, 'bo-', \
        label = 'log-lambda values vs log likelihoods')
    plt.xlabel('log10-lambda')
    plt.ylabel('NLL')
    plt.show()
```

```
plot_lambda_reg_vs_log_likelihood(X_train, y_train, X_val, y_val)
```

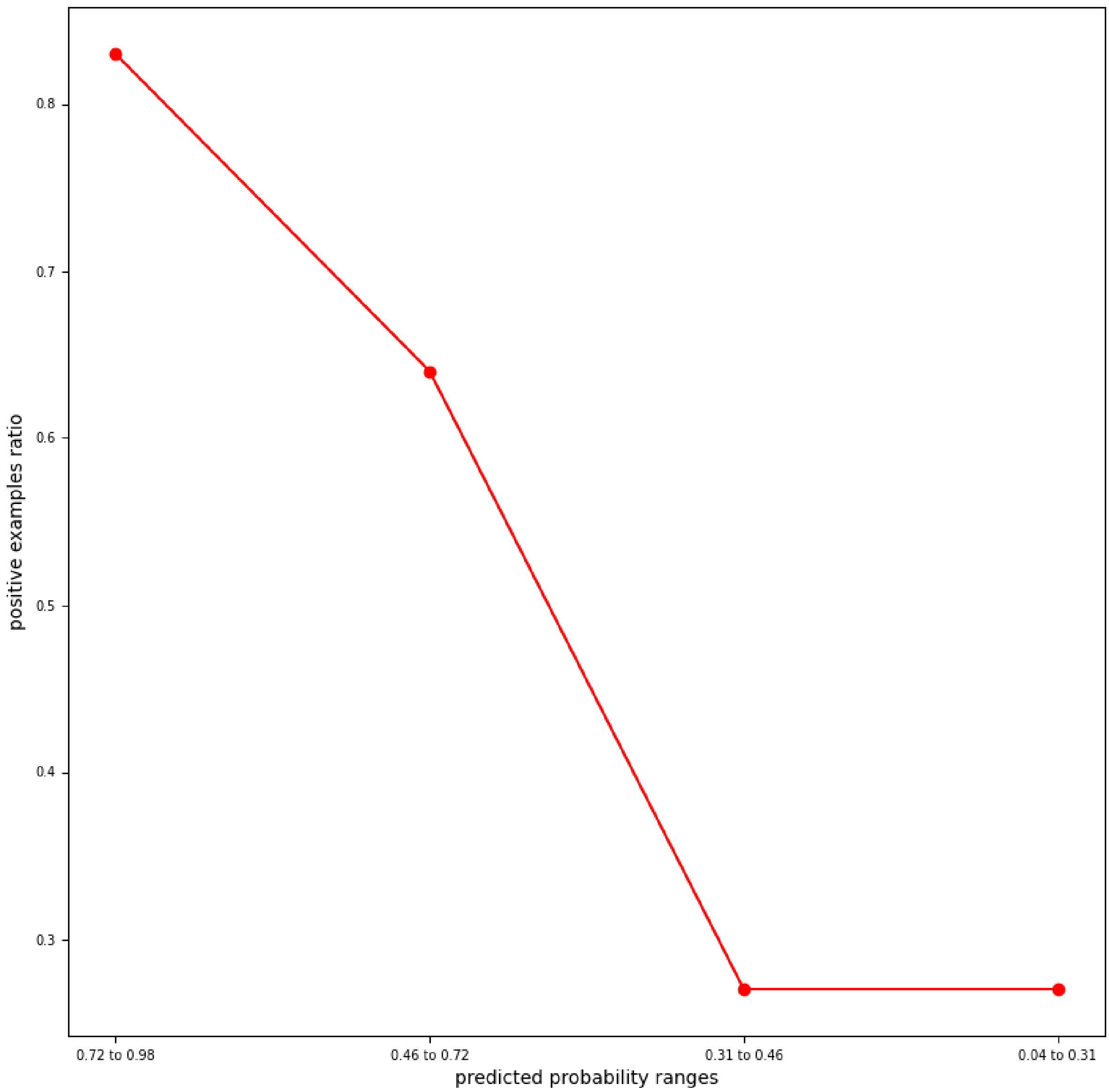


please turn over

Ans 24:

```
def plot_predicted_prob_calibration(theta, X, y, bins = 5):
    y_pred_prob = sigmoid(X @ theta)
    coupled_mat = np.column_stack((y, y_pred_prob))
    coupled_mat = coupled_mat[coupled_mat[:, 1].argsort()[::-1]]
    range_start = 0
    range_end = 0
    probability_ranges = []
    percentage_positive_samples = []
    for i in np.linspace(0, coupled_mat.shape[0], bins, dtype=int):
        if i == range_end:
            continue
        range_start = range_end
        range_end = i
        mat = coupled_mat[range_start:range_end, :]
        probability_ranges.append(f"{np.min(mat[:, 1]):.2f} to
{np.max(mat[:, 1]):.2f}")
        percentage_positive_samples.append((np.sum(mat[:, 0] == 1)) /
mat.shape[0])

    plt.figure(figsize=(10, 10))
    plt.plot(probability_ranges, percentage_positive_samples, 'ro-',
label='score ranges vs classification erros')
    plt.xlabel("predicted probability ranges")
    plt.ylabel("positive examples ratio")
    plt.xticks(fontsize=7)
    plt.yticks(fontsize=7)
    plt.show()
```

```
plot_predicted_prob_calibration(X= X_val, y= y_val, theta= theta, bins= 5)
```



Observations

- The % of positive examples in each prediction probability range changes almost linearly with the mean of the predicted probability ranges having value similar to the % of positive labelled samples in that class
- sigmoid((W.T)(X)) ~ (number of positive samples in class / total number of samples in class)