EECS 545 – Machine Learning - Homework #2

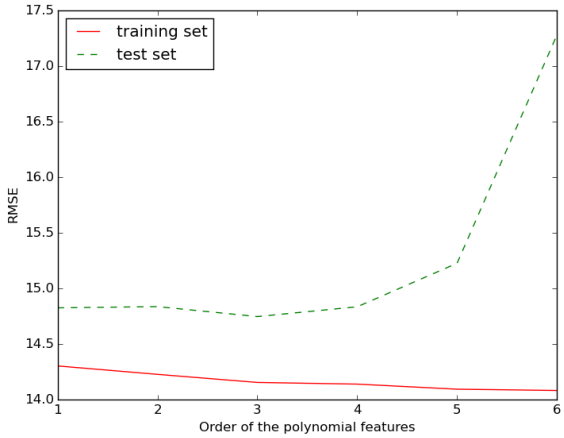
Mina Jafari
group member: Joshua Kammeraad

1) **Question 1.**

(a)

Including the id number in the feature set will cause a false dependence of controller output on an invalid parameter. Additionally, the id number is sorted while the data is not.

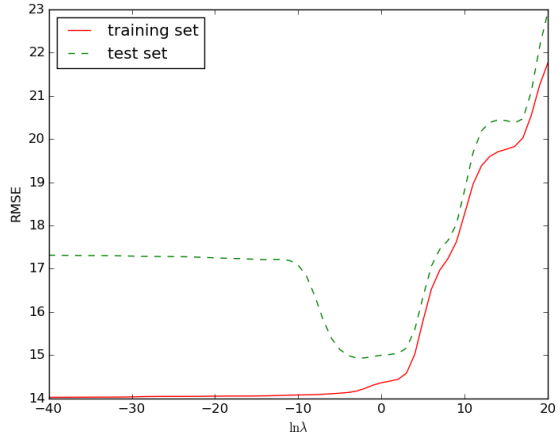
(i)




```
import numpy as np
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot as plt

# reading in the data file
A = np.loadtxt("train_graphs_f16_autopilot_cruise.csv", delimiter=",", skiprows=1)

numOfRows =  A.shape[0]
numOfCols = A.shape[1]

# deleting the first and last column
trainingData = np.delete(A, [0, numOfCols-1], 1)
# saving the last column as target vector
targetVector_train = A[:,-1]
# predict the values from test set
B = np.loadtxt("test_graphs_f16_autopilot_cruise.csv", delimiter=",", skiprows=1)
testData = np.delete(B, [0, B.shape[1]-1], 1)
targetVector_test = B[:,-1]

def computeFeatures(degree, inMatrix):
    outMatrix = inMatrix
    outMatrix = np.insert(outMatrix, 0, 1, axis=1)
    newColmn = np.empty([inMatrix.shape[0], 1])
    for power in range(2, degree+1):
        for j in range(0, inMatrix.shape[1]):
            for i in range(0, inMatrix.shape[0]):
                x = inMatrix[i][j]
                x_new = np.power(x, power)
                newColmn[i][0] = x_new
            outMatrix = np.insert(outMatrix, outMatrix.shape[1], newColmn.transpose(), axis=1)
    return outMatrix
```

```
rmseTraining = np.empty([6, 1])
rmseTest = np.empty([6, 1])
for i in range(1, 7):
    trainFeatures_i = computeFeatures(i, trainingData)
    testFeatures_i = computeFeatures(i, testData)
    # vector w
    coeffVector_i = np.dot(np.linalg.pinv(trainFeatures_i), targetVector_train)
    trainingTarget_i = np.dot(trainFeatures_i, coeffVector_i)
    testTarget_i = np.dot(testFeatures_i, coeffVector_i)
    # training error
    rmseTraining[i-1][0] = np.sqrt(mean_squared_error(targetVector_train, trainingTarget_i))
    # test error
    rmseTest[i-1][0] = np.sqrt(mean_squared_error(targetVector_test, testTarget_i))

x = [1,2,3,4,5,6]
plt.plot(x, rmseTraining, 'r', label="training set")
plt.plot(x, rmseTest, 'g--', label="test set")
plt.ylabel('RMSE')
plt.xlabel('Order of the polynomial features')
plt.legend(loc=2)
plt.show()
```

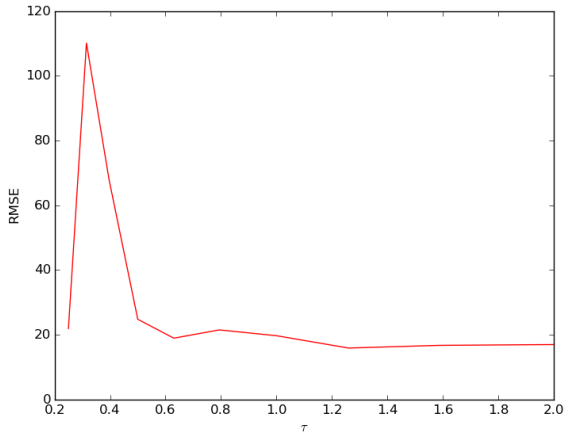
(ii)




```
import numpy as np
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot as plt
from numpy.linalg import inv, pinv

# reading in the data file
A = np.loadtxt("train_graphs_f16_autopilot_cruise.csv", delimiter=",", skiprows=1)

numOfRows =  A.shape[0]
numOfCols = A.shape[1]

# deleting the first and last column
trainingData = np.delete(A, [0, numOfCols-1], 1)
# saving the last column as target vector
targetVector_train = A[:,-1]

# predict the values from test set
B = np.loadtxt("test_graphs_f16_autopilot_cruise.csv", delimiter=",", skiprows=1)
testData = np.delete(B, [0, B.shape[1]-1], 1)
targetVector_test = B[:,-1]

def computeFeatures(degree, inMatrix):
    outMatrix = inMatrix
    outMatrix = np.insert(outMatrix, 0, 1, axis=1)
    newColmn = np.empty([inMatrix.shape[0], 1])
    for power in range(2, degree+1):
        for j in range(0, inMatrix.shape[1]):
            for i in range(0, inMatrix.shape[0]):
                x = inMatrix[i][j]
                x_new = np.power(x, power)
                newColmn[i][0] = x_new
```

```
            print "computing..."
            outMatrix = np.insert(outMatrix, outMatrix.shape[1], newColmn.transpose(), axis=1)
    return outMatrix

rmseTraining = np.empty([61, 1])
rmseTest = np.empty([61, 1])
for i in range(6, 7):
    trainFeatures_i = computeFeatures(i, trainingData)
    testFeatures_i = computeFeatures(i, testData)

    trainFeatures_i_T = trainFeatures_i.transpose()
    phi_T_phi = np.dot(trainFeatures_i_T, trainFeatures_i)
    ident = np.eye(phi_T_phi.shape[0], phi_T_phi.shape[1])
    j = 0
    for k in range(-40, 21):
        lamda = np.exp(k)
        lamdaEye = np.dot(lamda, ident)
        matrix_1 = np.add(lamdaEye, phi_T_phi)
        matrix_1_inv = inv(matrix_1)
        matrix_2 = np.dot(trainFeatures_i_T, targetVector_train)
        coeffVector_i_k = np.dot(matrix_1_inv, matrix_2)
        trainingTarget_i_k = np.dot(trainFeatures_i, coeffVector_i_k)
        testTarget_i_k = np.dot(testFeatures_i, coeffVector_i_k)
        # training error
        rmseTraining[j][0] = np.sqrt(mean_squared_error(targetVector_train, trainingTarget_i_k))
        # test error
        rmseTest[j][0] = np.sqrt(mean_squared_error(targetVector_test, testTarget_i_k))
        j = j + 1

x = range(-40, 21, 1)
plt.plot(x, rmseTraining, 'r', label="training set")
plt.plot(x, rmseTest, 'g--', label="test set")
plt.ylabel('RMSE')
plt.xlabel('$\ln \lambda$')
plt.legend(loc=2)
plt.show()
```

(b)




```
import numpy as np
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot as plt
from numpy.linalg import inv, norm, pinv

# reading in the data file
A = np.loadtxt("train_graphs_f16_autopilot_cruise.csv", delimiter=",", skiprows=1)

numOfRows =  A.shape[0]
numOfCols = A.shape[1]

# deleting the first and last column
trainingData = np.delete(A, [0, numOfCols-1], 1)
# saving the last column as target vector
targetVector_train = A[:,-1]
```

```
# predict the values from test set
B = np.loadtxt("test_locreg_f16_autopilot_cruise.csv", delimiter=",", skiprows=1)
testData = np.delete(B, [0, B.shape[1]-1], 1)
targetVector_test = B[:,-1]

a = np.zeros((3426, 3426), float)
phi_T = np.transpose(trainingData)
tau = np.logspace(-2, 1, num=10, base=2)
l = 0
rmseTest = np.empty([10])
for elem in tau:
    size = trainingData.shape[0]
    target = np.empty([testData.shape[0]])
    for j in range(0, testData.shape[0]):
        testPoint = testData[j]
        weightVector = np.empty([size])
        k = 0
        for i in range(0, trainingData.shape[0]):
            subt = np.subtract(trainingData[i], testPoint)
            normV = norm(subt)
            weightVector[k] = np.exp(-1 * np.power(normV, 2) / (2 * np.power(elem, 2)) )
            k += 1
        weightVector_2 = np.diag(weightVector)
        inverse = pinv(np.dot(np.dot(phi_T, weightVector_2), trainingData))
        rt = np.dot(weightVector_2, targetVector_train)
        vecW = np.dot(np.dot(inverse, phi_T), rt)
        target[j] = np.dot(vecW, testPoint)
    rmseTest[l] = np.sqrt(mean_squared_error(targetVector_test, target))
    l += 1

plt.plot(tau, rmseTest, 'r')
plt.ylabel('RMSE')
plt.xlabel('$\\tau$')
plt.show()
```

2) **Question 2.**

```
import numpy as np
from sklearn import datasets, linear_model
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

# reading in the data file
A = np.loadtxt("steel_composition_train_2.csv", delimiter=",", skiprows=1)
B = np.loadtxt("steel_composition_test.csv", delimiter=",", skiprows=1)

numOfRows =  A.shape[0]
numOfCols = A.shape[1]

# deleting the first and last column
trainingData = np.delete(A, [0, numOfCols-1], 1)
testData = np.delete(B, 0, 1)

# saving the last column as target vector
targetVector = A[:,-1]

# polynomial regression
poly2 = PolynomialFeatures(2)
poly2.fit_transform(trainingData, targetVector)

model = Pipeline([('poly2', PolynomialFeatures(degree=2)), \
        ('linear', linear_model.LinearRegression(fit_intercept=False))])
model = model.fit(trainingData, targetVector)
output = model.predict(testData)

f = open('test-degree2.csv', 'w')
f.write("id,Strength\n")
for i in range(1, output.shape[0]+1):
        f.write(str(i))
        f.write(",")
        f.write(str(output[i-1]))
        f.write("\n")
f.close()
```

3) **Question 3.**

(a)

$$
\begin{aligned}
E_D(w) &= \frac{1}{2}\sum_{i=1}^{N} r_i (w^\top x_i - t_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{N} \left[r_i (w^\top x_i)^2 - 2 r_i w^\top x_i t_i + r_i t_i^2 \right] \\
&= \frac{1}{2}\sum_{i=1}^{N} \left[w^\top x_i r_i w^\top x_i - 2 w^\top x_i r_i t_i + t_i r_i t_i \right] \\
&= w^\top X^\top R X w - 2 w^\top X^\top R t + t^\top R t \\
&= w^\top X^\top R X w - w^\top X^\top R t - w^\top X^\top R t + t^\top R t \\
w^\top X^\top R t &= (w^\top X^\top R t)^\top \text{ since this is a scalar} \\
&= w^\top X^\top R X w - w^\top X^\top R t - t^\top R X w + t^\top R t \\
&= w^\top X^\top R (X w - t) - t^\top R (X w - t) \\
&= (w^\top X^\top - t^\top)(R(X w - t)) \\
&= (X w)^\top - t^\top)(R(X w - t)) \\
&= (X w - t)^\top R (X w - t)
\end{aligned}
$$

$\mathbf{X}$: matrix, n $\times$ m (n=number of data points in training/test set, m=number of features), each x_i (data point) is one the rows of this matrix
$\mathbf{t}$: vector, n $\times$ 1 (n=number of data points in training/test set), each t_i (target value) is one the elements of this vector
$\mathbf{R}$: diagonal matrix, n $\times$ n (n=number of elements in t), each r_i (calculated weight) is one of the diagonal elements of this matrix

(b)

$$
\begin{aligned}
\nabla_w E_D(w) &= \nabla w \big((X w - t)^\top R (X w - t) \big) \\
&= \nabla w \big(w^\top X^\top R X w - 2 w^\top X^\top R t + t^\top R t \big) \\
&= X^\top R X w - X^\top R t \\
\nabla_w E_D(w^*) &= X^\top R X w^* - X^\top R t = 0 \\
&\implies w^* = (X^\top R X)^{-1} X^\top R t
\end{aligned}
$$

(c)

since the examples are independent, $f(t|x;w) = \prod_{i=1}^{N} p(t_i|x_i;w)$

$$
\begin{aligned}
MLE(w) &= \frac{\partial}{\partial w}\log\left(\prod_{i=1}^{N} p(t_i|x_i;w)\right) \\
&= \frac{\partial}{\partial w}\log\left(\prod_{i=1}^{N} \frac{1}{\sigma_i\sqrt{2\pi}}\exp\Big(-\frac{(t_i - w^T x_i)^2}{2(\sigma_i)^2}\Big)\right) \\
&= \frac{\partial}{\partial w}\sum_{i=1}^{N}\left(\log\frac{1}{\sigma_i\sqrt{2\pi}} - \frac{(t_i - w^T x_i)^2}{2(\sigma_i)^2}\right) \\
&= \frac{\partial}{\partial w}\sum_{i=1}^{N}\frac{-1}{2(\sigma_i)^2}(t_i - w^T x_i)^2
\end{aligned}
$$

$$
\begin{aligned}
MLE(w) &= \frac{1}{2}\frac{\partial}{\partial w}\sum_{i=1}^{N}\frac{-1}{(\sigma_i)^2}(t_i - w^T x_i)^2 \\
E_D(w) &= \frac{1}{2}\sum_{i=1}^{N} r_i(w^\top x_i - t_i)^2 \implies r_i = \frac{1}{\sigma_i^2}
\end{aligned}
$$

Maximizing the likelihood function $MLE(w)$ is equivalent to minimizing its negative $-MLE(w)$, so $\frac{-1}{\sigma_i^2}$ changes to $\frac{1}{\sigma_i^2}$.

4) **Question 4.**

(a)

```
import numpy as np
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot as plt
from numpy.linalg import inv, norm

# reading in the data file
A = np.loadtxt("spambase.train", delimiter=",")

numOfRows =  A.shape[0]
numOfCols = A.shape[1]

trainingData = np.delete(A, [numOfCols-1], 1)
targetVector_train = A[:,-1]

B = np.loadtxt("spambase.test", delimiter=",")
testData = np.delete(B, [B.shape[1]-1], 1)
targetVector_test = B[:,-1]

aggregMatrix = np.vstack((trainingData, testData))
targetVector_aggreg = np.hstack((targetVector_train, targetVector_test))

median = np.empty([aggregMatrix.shape[1]])
for i in range(0, aggregMatrix.shape[1]):
    temp = aggregMatrix[:,i]
    median[i] = np.median(temp)

trainingMapping = np.empty([trainingData.shape[0], trainingData.shape[1]])
for i in range(0, trainingData.shape[0]):
    for j in range(0, trainingData.shape[1]):
        if trainingData[i][j] < median[j]:
            trainingMapping[i][j] = 0
        else:
            trainingMapping[i][j] = 1

testMapping = np.empty([testData.shape[0], testData.shape[1]])
for i in range(0, testData.shape[0]):
    for j in range(0, testData.shape[1]):
        if testData[i][j] < median[j]:
            testMapping[i][j] = 0
        else:
            testMapping[i][j] = 1

sumSpam = 0.0
for elem in targetVector_train:
    if elem == 1:
        sumSpam += 1
sumHam = targetVector_train.shape[0] - sumSpam
probSpam = 0.0
probSpam = sumSpam / targetVector_train.shape[0]
probHam = 1 - probSpam
# finding theta
counterSpam = np.empty([trainingMapping.shape[1]])
counterHam = np.empty([trainingMapping.shape[1]])
for i in range(0, trainingMapping.shape[0]):
    if targetVector_train[i] == 1:
        for j in range(0, trainingMapping.shape[1]):
            if trainingMapping[i][j] == 1:
                counterSpam[j] += 1
    elif targetVector_train[i] == 0:
        for j in range(0, trainingMapping.shape[1]):
            if trainingMapping[i][j] == 1:
                counterHam[j] += 1

thetaSpam = np.empty([trainingData.shape[1]])
thetaHam = np.empty([trainingData.shape[1]])
for j in range(0, counterSpam.shape[0]):
    thetaSpam[j] = counterSpam[j] / sumSpam
    thetaHam[j] = counterHam[j] / sumHam

probabilitySpam = np.empty(testMapping.shape)
probabilityHam = np.empty(testMapping.shape)
for i in range(0, testMapping.shape[0]):
    for j in range(0, testMapping.shape[1]):
```

```
        if testMapping[i][j] == 0:
            probabilitySpam[i][j] = 1 - thetaSpam[j]
            probabilityHam[i][j] = 1 - thetaHam[j]
        elif testMapping[i][j] == 1:
            probabilitySpam[i][j] = thetaSpam[j]
            probabilityHam[i][j] = thetaHam[j]

misCalc = 0.0
sanityCalc = 0.0
probGivenSpam = np.ones(testData.shape[0])
probGivenHam = np.ones(testData.shape[0])
probIfSpam = np.ones(testData.shape[0])
probIfHam = np.ones(testData.shape[0])
calcTarget = np.empty([testData.shape[0]])
for i in range(0, probabilitySpam.shape[0]):
    for j in range(0, probabilitySpam.shape[1]):
        probGivenSpam[i] *= probabilitySpam[i][j]
        probGivenHam[i] *= probabilityHam[i][j]
    probIfSpam[i] = \
    (probGivenSpam[i] * probSpam) / (probGivenSpam[i] * probSpam + probGivenHam[i] * probHam)
    probIfHam[i] = \
    (probGivenHam[i] * probHam) / (probGivenSpam[i] * probSpam + probGivenHam[i] * probHam)
    if probIfSpam[i] > probIfHam[i]:
        calcTarget[i] = 1
    else:
        calcTarget[i] = 0
    if calcTarget[i] != targetVector_test[i]:
        misCalc += 1
    if targetVector_test[i] != 0:
        sanityCalc += 1

error = misCalc / targetVector_test.shape[0]
print error*100
print sanityCalc / targetVector_test.shape[0] * 100
```

(i)

All of them except nominal could be used for the preprocessing, meaning we can define a median value for data of that type.

The features are as follows:

The first 48 columns are percentage of words in the e-mail that match a specific WORD. The next 6 columns are percentage of characters in the e-mail that match a specific character, CHAR. The following single columns are the average length of uninterrupted sequences of capital letters, length of longest uninterrupted sequence of capital letters, and sum of length of uninterrupted sequences of capital letters, respectively. The last column is a nominal 0,1 class attribute which denotes whether the e-mail was considered spam (1) or not (0).

All of the feature columns can be considered ratio variables, because if their magnitude is zero, it means that word or character does not exist and the difference between two values is meaningful. However, one would argue that the difference between two values does not contain any useful information. For example, if we are trying to predict the amount of energy required to warm up a room when the outside temperature drops by 10 degrees, the magnitude of change in temperature is an important determining part of the equation. But, I am not sure if we can infer any significant information from small differences in length of capital letters or some of the other features.

reference: `https://archive.ics.uci.edu/ml/datasets/Spambase`

(ii)

Test error = 25.03%

If you classify the values less than or equal to the median to be non-spam, then the error would be 10.84%.

The majority class from the training data is non-spam with 59.5% probability. If we always predicted non-spam for the test test, the error would be 38.56%.

(b)

The extract_features.py script produces a sparse feature matrix with 5172 rows and 47922 columns. Given a row (data point = e-mail), value of each column is the number of times that a word appears in each e-mail. As an example, in e-mail one of the training set, the line is split after the first tab. It contains 5 words (Subject, christmas, tree, farm, pictures) and each of them occurred once. The lemmatize() function "groups together the different inflected forms of a word so they can be analyzed as a single item". The script also removes all the stopwords and punctuation marks.

```
import os
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
import cPickle
from sklearn.naive_bayes import MultinomialNB

def getFeature():
    with open(os.path.join('spam_filter_train.txt'), 'r') as f:
        trainData = f.readlines()
    with open(os.path.join('spam_filter_test.txt'), 'r') as f:
        testData = f.readlines()
    data = trainData + testData
    trainNum, testNum = len(trainData), len(testData)
    del trainData
    del testData

    targets = []
    for i in range(trainNum):
        targets.append(0) if data[i].split('\t')[0] == "ham" else targets.append(1)
    for i in range(len(data)):
        data[i] = data[i].replace('\n', '').split('\t')[1]
    # lemmatize
    lemmatized = []
    wnl = WordNetLemmatizer()
    for line in data:
        lemmatized.append([wnl.lemmatize(word) for word in line.split('␣')])
    # remove stopwords
    stopwordRemoved = []
    sw = set(stopwords.words('english'))
    for line in lemmatized:
        stopwordRemoved.append('␣'.join([x for x in line if x not in sw]))
    # tf feature
    vec = CountVectorizer()
    features = vec.fit_transform(stopwordRemoved)

#my addition
    multiNom = MultinomialNB(alpha=1.8, fit_prior=True)
    multiNom.fit(features[:trainNum], targets)
    testTargets =  multiNom.predict(features[trainNum:])
    f = open('testTargets.csv', 'w')
    f.write("id,output\n")
    for i in range(1, len(testTargets)+1):
        f.write(str(i))
        f.write(",")
        f.write(str(testTargets[i-1]))
        f.write("\n")
    f.close()
#my addition

'''
    with open('trainFeatures.pkl', 'wb') as f:
        cPickle.dump(features[:trainNum], f)
    with open('testFeatures.pkl', 'wb') as f:
        cPickle.dump(features[trainNum:], f)
'''

def main():
    getFeature()
'''
    with open('trainFeatures.pkl', 'rb') as f:
         trainFeatures = cPickle.load(f)
    with open('testFeatures.pkl', 'rb') as f:
         testFeatures = cPickle.load(f)
'''
```

```
if __name__ == '__main__':
    main()
```

5) **Question 5.**

(a)

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N}\prod_{k=0}^{K-1} p(C_k|\phi(\mathbf{x}_n))^{\mathbf{1}(t_n=k)}$$

$$\begin{aligned} E(\mathbf{w}) &= -\log p(\mathbf{t}|\mathbf{w}) \\ &= -\log\left(\prod_{n=1}^{N}\prod_{k=0}^{K-1} p(C_k|\phi(\mathbf{x}_n))^{\mathbf{1}(t_n=k)}\right) \\ &= -\log\left(\prod_{n=1}^{N}\prod_{k=0}^{K-1} \frac{\exp(\mathbf{w}_k^T\phi(\mathbf{x}_n))}{\sum_{k=0}^{K-1}\exp(\mathbf{w}_k^T\phi(\mathbf{x}_n))}\right)^{\mathbf{1}(t_n=k)} \\ &= -\sum_{n=1}^{N}\sum_{k=0}^{K-1}\mathbf{1}(t_n=k)\left[\mathbf{w}_k^T\phi(\mathbf{x}_n) - \log\left(\sum_{i=0}^{K-1}\exp(\mathbf{w}_i^T\phi(\mathbf{x}_n))\right)\right] \end{aligned}$$

The above expression is non-zero only when $t_n = k \implies$

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\mathbf{w}_{t_n}^T\phi(\mathbf{x}_n) + \sum_{n=1}^{N}\log\left(\sum_{i=0}^{K-1}\exp(\mathbf{w}_i^T\phi(\mathbf{x}_n))\right)$$

Gradient of the terms where $j \neq t_n$ is 0 with respect to $\mathbf{w}_j \implies$

$$\begin{aligned} \nabla_{\mathbf{w}_j}E(\mathbf{w}) &= -\sum_{n=1}^{N}\mathbf{1}(t_n=j)\phi(\mathbf{x}_n) + \sum_{n=1}^{N}\left(\frac{\phi(\mathbf{x}_n)\exp(\mathbf{w}_j^T\phi(\mathbf{x}_n)}{\sum_{i=0}^{K-1}\exp(\mathbf{w}_i^T\phi(\mathbf{x}_n))}\right) \\ &= \sum_{n=1}^{N}\left(\frac{\phi(\mathbf{x}_n)\exp(\mathbf{w}_j^T\phi(\mathbf{x}_n)}{\sum_{i=0}^{K-1}\exp(\mathbf{w}_i^T\phi(\mathbf{x}_n))} - \mathbf{1}(t_n=j)\phi(\mathbf{x}_n)\right) \end{aligned}$$

(b)

$$\begin{aligned} E^{\lambda}(\mathbf{w}) &= E(\mathbf{w}) + \frac{\lambda}{2}\sum_{k=0}^{K-1}\mathbf{w}_k^T\mathbf{w}_k \\ &= -\sum_{n=1}^{N}\sum_{k=0}^{K-1}\mathbf{1}(t_n=k)\left[\mathbf{w}_k^T\phi(\mathbf{x}_n) - \log\left(\sum_{i=0}^{K-1}\exp(\mathbf{w}_i^T\phi(\mathbf{x_n}))\right)\right] + \frac{\lambda}{2}\sum_{k=0}^{K-1}\mathbf{w}_k^T\mathbf{w}_k \end{aligned}$$

$$\begin{aligned} \nabla_{\mathbf{w}_j}E^{\lambda}(\mathbf{w}) &= \nabla_{\mathbf{w}_j}E(\mathbf{w}) + \nabla_{\mathbf{w}_j}\left(\frac{\lambda}{2}\sum_{k=0}^{K-1}\mathbf{w}_k^T\mathbf{w}_k\right) \\ &= \sum_{n=1}^{N}\left(\frac{\phi(\mathbf{x}_n)\exp(\mathbf{w}_j^T\phi(\mathbf{x}_n)}{\sum_{i=0}^{K-1}\exp(\mathbf{w}_i^T\phi(\mathbf{x}_n))} - \mathbf{1}(t_n=j)\phi(\mathbf{x}_n)\right) + \lambda\mathbf{w}_j \end{aligned}$$