Assignment 3: Classification & Model Selection

CS 589 - ML

Shubham Shetty (shubhamshett@umass.edu)
Brinda Murulidhara (bmurulidhara@umass.edu)
Adarsh Kolya (akolya@umass.edu)

March 2021

Preface

Before running our codes, the following packages are imported and test & training data are assigned to variables. Also, some reusable functions are defined -

```
# Import Statements
import numpy as np
import sklearn as sk
from sklearn.tree import *
from sklearn.linear_model import *
from sklearn.svm import *
from sklearn.metrics import *
from sklearn.neighbors import *
from sklearn.model_selection import KFold
from datetime import datetime
import math
import csv
import pandas as pd
import sys
import matplotlib.pyplot as plt
import multiprocessing

# Load Data
data = np.load("data.npz")

X_trn = data["X_trn"]
y_trn = data["y_trn"]
X_tst = data["X_tst"]


# Common Functions

# Show image from data
def show(x):
    img = x.reshape((3,32,32)).transpose(1,2,0)
    plt.imshow(img)
    plt.axis('off')
    plt.draw()
    plt.pause(0.01)

# Write data to CSV file on local system
def write_csv(y_pred, filename):
    """Write a 1d numpy array to a Kaggle-compatible .csv file"""
    with open(filename, 'w+') as csv_file:
        csv_writer = csv.writer(csv_file)
        csv_writer.writerow(['Id', 'Category'])
        for idx, y in enumerate(y_pred):
            csv_writer.writerow([idx, y])

# Helper funtion to print table
# Input variable table is a numpy array and headers is an array of
    column headers
def prettyPrintTable(table, headers) :
```

```
    for i, d in enumerate(table):
        if i == 0 :
            line = '|'.join(str(x).ljust(30) for x in headers)
            print(line)
            print('-' * len(line))

        line = '|'.join(str(x).ljust(30) for x in d)
        print(line)
    print()
```

Answer 1

Given -

$V = \{0, 1\}$

Probabilities for training outputs are-

$P(0) = 1/4$

$P(1) = 3/4$

Let D be cross entropy and I represent information gain. Then, cross entropy before the split is (assuming log base e)-

$$D_0 = -[\tfrac{1}{4}log(\tfrac{1}{4}) + \tfrac{3}{4}log(\tfrac{3}{4})] = 0.562$$

1. Split at x = 0.5

   $D_{m<0.5} = 0$

   $D_{m\geq0.5} = -[\frac{1}{4}log(\frac{1}{4}) + \frac{3}{4}log(\frac{3}{4})] = 0.562$

   Therefore,

   $\boxed{I = 0.562 - 0.562 = 0}$

2. Split at x = 1.5

   $D_{m<1.5} = -log(1) = 0$

   $D_{m\geq1.5} = -[\frac{2}{3}log(\frac{2}{3}) + \frac{1}{3}log(\frac{1}{3})] = 0.636$

   Therefore,

   $\boxed{I = 0.562 - \frac{3}{4} * 0.634 = 0.0846}$

3. Split at x = 2.5

   $D_{m<2.5} = -[\frac{1}{2}log(\frac{1}{2}) + \frac{1}{2}log(\frac{1}{2})] = 0.693$

   $D_{m\geq2.5} = -log(1) = 0$

   Therefore,

   $\boxed{I = 0.562 - \frac{1}{2} * 0.693 = 0.2155}$

4. Split at x = 3.5

   $D_{m<3.5} = -[\frac{2}{3}log(\frac{2}{3}) + \frac{1}{3}log(\frac{1}{3})] = 0.636$

   $D_{m\geq3.5} = -log(1) = 0$

   Therefore,

   $\boxed{I = 0.562 - \frac{3}{4} * 0.634 = 0.0846}$

5. Split at x = 4.5

   $D_{m<4.5} = -[\frac{1}{4}log(\frac{1}{4}) + \frac{3}{4}log(\frac{3}{4})] = 0.562$

   $D_{m\geq4.5} = 0$

   Therefore,

   $\boxed{I = 0.562 - 0.562 = 0}$

Answer 2

Time complexity to evaluate that classification tree on a single new input-

$$\mathcal{O}(M)$$

In the worst case, the longest path from root to leaf of the tree may be traversed to evaluate the classification tree on a single new input. The longest root to leaf path is equal to the depth (M) of the tree.

Answer 3

Time complexity to train a classification stump-

$$\mathcal{O}(DNlogN)$$

There are D dimensions, and it takes N log N time for each sorting operation. Checking all the split points for a given dimension takes N time once the data is sorted.

Without any optimisation, the complexity is $\mathcal{O}(DN^2)$. There are D dimensions and checking all the split points for a given dimension takes N^2 time once the data is sorted.

Answer 4

Following method trains a classification tree to predict outputs for each of the following possible depths: 1,3,6,9,12,14. Then using 5-fold cross validation out of sample classification error is determined.
These methods use `DecisionTreeClassifier` and `KFold` modules from `scikit-learn`.

```
import numpy as np
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeClassifier

def train_classifier(X_trn, y_trn, depths):
    errors = np.zeros((6, 2))
    n_splits = 5
    kf = KFold(n_splits=n_splits, random_state=None, shuffle=True)
    for ind, depth in enumerate(depths):
        clf = DecisionTreeClassifier(random_state=None, max_depth=depth)
        error = 0
        for train_index, test_index in kf.split(X_trn):
            # Split train-test
            X_train, X_test = X_trn[train_index], X_trn[test_index]
            y_train, y_test = y_trn[train_index], y_trn[test_index]

            # Train the model
            model = clf.fit(X_train, y_train)
            predictions = model.predict(X_test)

            error += 1-sum([1 for actual, predicted in zip(y_test, predictions) if actual==predicted])/len(X_test)
        errors[ind] = depth, error/n_splits

    print(errors)


data = np.load("data.npz")
X_trn = data["X_trn"]
y_trn = data["y_trn"]
X_tst = data["X_tst"]
train_classifier(X_trn, y_trn, {1, 3, 6, 9, 12, 14})
```

The below table lists the out of sample classification error for different depths.

Depth	**Mean classification error**
1	0.6365
3	0.54316667
6	0.498
9	0.50266667
12	0.51783333
14	0.50866667

Table 1: Mean out of sample classification error for Classification Tree

Answer 5

1. We chose depth of tree to be **6**.
2. Our estimated Generalized error using 5-fold validation for this depth was **0.498**
3. On the public part of the leader-board the accuracy observed was **0.49722**

Answer 6

The time complexity of KNN classifier that uses a brute force approach to pick K nearest points from N points is :

$$\mathcal{O}(ND + NK)$$

In a KNN classifier, the steps involved are as below :

1. Calculate distance of N points from the given point - $\mathcal{O}(ND)$
2. Pick K smallest distances from N distances - $\mathcal{O}(NK)$
3. Find label with max frequency in K outputs - $\mathcal{O}(K)$

The over all time complexity - $\mathcal{O}(ND + NK)$

Answer 7

Following methods do nearest-neighbor prediction for each of the following possible values of K: 1, 3, 5, 7, 9, 11. Then using 5-fold cross validation out of sample classification error is determined.
These methods use `KNeighborsClassifier` and `KFold` modules from `scikit-learn`.

```python
from preface import *

# KNN Classification
def KNN_classification(numNeighbors, X_trn, y_trn, X_val):
    neigh = KNeighborsClassifier(n_neighbors=numNeighbors, n_jobs=-1)
    neigh.fit(X_trn, y_trn)
    y_pred = neigh.predict(X_val)
    return y_pred

# Evaluate what capacity of KNN would be optimum
def KNNModelSelection() :
    data = np.load("data.npz")
    X_trn = data["X_trn"]
    y_trn = data["y_trn"]
    X_tst = data["X_tst"]

    kf = KFold(n_splits=5,shuffle=True)
    kf.get_n_splits(X_trn)

    kFoldForTraining = []
    kFoldForTesting = []

    # Split and store the splits so that the same splits can be used for all values of K.
    for train_index, test_index in kf.split(X_trn):
        kFoldForTraining.append(train_index)
        kFoldForTesting.append(test_index)

    K = [1, 3, 5, 7, 9, 11]
    classificationError = []
    for i in K:
        # For each of the splits, run KNN classification
        classErrForThisValueOfK=0
        for splitNum in range(len(kFoldForTraining)) :
            X_train, X_test = X_trn[kFoldForTraining[splitNum]], X_trn[kFoldForTesting[splitNum]]
            y_train, y_test = y_trn[kFoldForTraining[splitNum]], y_trn[kFoldForTesting[splitNum]]
            y_pred = KNN_classification(i, X_train, y_train, X_test)
            classErrForThisValueOfK = classErrForThisValueOfK + (1 - accuracy_score(y_test, y_pred))

        # Store the mean classification error for this value of K in the numpy array
        classificationError.append([i, classErrForThisValueOfK/5.0])
```

```

    prettyPrintTable(classificationError, ["Num neighbors","Errors"])

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

The below table lists the out of sample classification error for different values of K.

K	**Mean classification error**
1	0.456667
3	0.453167
5	0.449000
7	0.439000
9	0.432667
11	0.432167

Table 2: Mean out of sample classification error for KNN classifier

Answer 8

1. We chose K to be **11**.
2. Generalized error using 5-fold validation for this value of K was **0.432167**.
3. On the public part of the leader-board the accuracy observed was **0.61666**.

Answer 9

The methods defined below perform logistic regression and hinge classification respectively for the given datasets with $\lambda \in \{10^{-4}, 10^{-2}, 1, 10, 100\}$, validate the results according to 5-fold cross validation , and print the required metrics (0-1 misclassification error, hinge loss, and logistic loss.)

These methods use `LogisticRegression`, `LinearSVC` and `KFold` modules from `scikit-learn`.

```
# Custom softmax function to obtain probabilities from confidence scores
def softmax(x):
    return np.exp(x)/sum(np.exp(x))


# Function to return required mtrics (0-1 misclassification error, Logistic loss, Hinge Loss)
def getMetrics(l, kf, clf):
    misc_error, ll_aggr, hl_aggr  = 0, 0, 0
    for trn_index, tst_index in kf.split(X_trn):
        # Split training data accordingly to training and validation data sets
        cv_x_trn, cv_x_tst = X_trn[trn_index], X_trn[tst_index]
        cv_y_trn, cv_y_tst = y_trn[trn_index], y_trn[tst_index]

        # Fit model to training set
        model = clf.fit(cv_x_trn, cv_y_trn)

        # Decision function to generate confidence values
        decisions = model.decision_function(cv_x_tst)
        cv_pred_tst = np.argmax(decisions, axis=1)

        # Probability values for decisions
        probs = softmax(decisions)

        # Calculate miscalculation error for this particular fold
        error_1 = 0
        for x,y in zip(cv_y_tst, cv_pred_tst):
            if x!=y:
                error_1+=1
        misc_error += error_1/len(cv_pred_tst)


        # Calculate Log Loss for this particular fold
        ll = log_loss(cv_y_tst, probs)
        ll_aggr += ll

        # Calculate Hinge Loss for this particular fold
        hl = hinge_loss(cv_y_tst, decisions)
        hl_aggr += hl

```

```
    # Calculate average metrics across splits
    avg_misc_error = misc_error/kf.n_splits
    avg_ll = ll_aggr/kf.n_splits
    avg_hl = hl_aggr/kf.n_splits

    return avg_misc_error, avg_ll, avg_hl


# Validate Logistic Regression Model
def logistic_regressor():
    l_vals=[0.0001, 0.01, 1, 10, 100]

    # Define splits for K-Fold Cross-Validation
    n_splits=5
    # KFold Cross Validator
    kf = KFold(n_splits=n_splits, shuffle=True)

    misc_err = []
    log_loss = []
    hinge_loss = []

    for l in l_vals:
        log_reg = LogisticRegression(penalty='l2', C=1/(2*l), solver='sag', max_iter=100, multi_class='multinomial',
                                                n_jobs = -1)
        me, ll, hl = getMetrics(l, kf, log_reg)
        misc_err.append(me)
        log_loss.append(ll)
        hinge_loss.append(hl)

    print("Lambda\t\t\tError Value")
    for l in range(len(l_vals)):
        print(f"{l_vals[l]}\t\t\t{misc_err[l]}")
    print("\n")
    print("Lambda\t\t\tLog Loss")
    for l in range(len(l_vals)):
        print(f"{l_vals[l]}\t\t\t{log_loss[l]}")
    print("\n")
    print("Lambda\t\t\tHinge Loss")
    for l in range(len(l_vals)):
        print(f"{l_vals[l]}\t\t\t{hinge_loss[l]}")


# Validate Hinge Classification Model
def hinge_classifier():
     l_vals=[0.0001, 0.01, 1, 10, 100]

     # Define splits for K-Fold Cross-Validation
     n_splits=5
     # KFold Cross Validator
     kf = KFold(n_splits=n_splits, shuffle=True)

```

```
    misc_err = []
    log_loss = []
    hinge_loss = []

    for l in l_vals:
        hinge_clf = LinearSVC(loss= 'hinge', C = 1/(2*l), max_iter=1000)
        me, ll, hl = getMetrics(l, kf, hinge_clf)
        misc_err.append(me)
        log_loss.append(ll)
        hinge_loss.append(hl)

    print("Lambda\t\t\tError Value")
    for l in range(len(l_vals)):
        print(f"{l_vals[l]}\t\t\t{misc_err[l]}")
    print("\n")
    print("Lambda\t\t\tLog Loss")
    for l in range(len(l_vals)):
        print(f"{l_vals[l]}\t\t\t{log_loss[l]}")
    print("\n")
    print("Lambda\t\t\tHinge Loss")
    for l in range(len(l_vals)):
        print(f"{l_vals[l]}\t\t\t{hinge_loss[l]}")


if __name__ == "__main__":
  logistic_regressor()
  hinge_classifier()
```

The below tables are the metrics observed for **logistic regression** model -

Lambda	Error Value
0.0001	0.41233333333333333
0.01	0.41216666666666667
1	0.41
10	0.39566666666666667
100	0.38666666666666666

Table 3: 0-1 Misclassification Error for Logistic Regression

Lambda	Log Loss
0.0001	1.9744296274108781
0.01	1.9387817406836756
1	1.8253285331018811
10	1.483022446742442
100	1.1412247852627455

Table 4: Log Loss for Logistic Regression

Lambda	Hinge Loss
0.0001	1.1703172713760126
0.01	1.1645862979908836
1	1.1242905902918365
10	0.9723185930244626
100	0.8656582863025036

Table 5: Hinge Loss for Logistic Regression

The below tables are the metrics observed for **hinge classification** model -

Lambda	Error Value
0.0001	0.48949999999999994
0.01	0.49416666666666664
1	0.4491666666666667
10	0.4073333333333333
100	0.39899999999999997

Table 6: 0-1 Misclassification Error for Hinge Classification

Lambda	Log Loss
0.0001	1.658265210815916
0.01	1.6599533809685176
1	1.5731331893304552
10	1.1616912901671272
100	1.0157286277097335

Table 7: Log Loss for Hinge Classification

Lambda	Hinge Loss
0.0001	1.4622453356275216
0.01	1.4936376941641398
1	1.3380753667748453
10	0.990812192361957
100	0.871463119505194

Table 8: Hinge Loss for Hinge Classification

Answer 10

1. We chose to use logistic loss model, with λ set to be **100**.

2. Generalized error using 5-fold validation for this value of λ was **0.3867**. Log loss for this model was **1.1412**, and hinge loss for this model was **0.8657**.

3. On the public part of the leader-board, the accuracy observed was **0.62500**.

Answer 11

```python
def prediction_loss(x,y,W,V,b,c):
    # Compute f(x) for all x.
    Wx = np.matmul(W,x)
    bPlusWx = np.add(b, Wx)

    # fx = c + V.tanh(b + W.x)
    fx = np.add(c, np.matmul(V, np.tanh(bPlusWx)))

    expFx = 0
    for fi in fx :
        expFx = expFx + np.exp(fi)

    # Loss = -f_y(x) + log(sum(exp(f_i(x))))
    L = 0 - fx[y] + np.log(expFx)

    return L
```

Answer 12

```python
def prediction_grad(x,y,W,V,b,c):
    # Compute f(x) for all x. f(x) = c + V.tanh(b + W.x)
    Wx = np.matmul(W,x)
    bPlusWx = np.add(b, Wx)
    fx = np.add(c, np.matmul(V, np.tanh(bPlusWx)))

    # Compute dLdf = - e_hat_y + g(f(x)). g(f(x)) = exp(f_y) / sumOfAll(exp(f))
    # e_hat_y is a unit vector with value 1 for y and 0 for rest. Compute negative unit vector
    numLables = c.shape[0]
    e = np.zeros((numLables, numLables))
    for i in range(numLables) :
        e[i][i] = -1

    # Compute g(f(x))
    sigma_exp_fv =  0
    for i in range(numLables) :
        sigma_exp_fv = sigma_exp_fv + np.exp(fx[i])
    gfx = np.zeros(numLables)
    for i in range(numLables) :
        gfx[i] = np.exp(fx[i]) / sigma_exp_fv

    dLdf = e[y] + gfx

    dLdc = dLdf

    h = np.tanh(bPlusWx)
    dLdV = np.outer(dLdf,h)

    derivativeOfTanh = 1 - np.tanh(bPlusWx)**2

    dLdb = derivativeOfTanh * np.matmul(np.transpose(V), dLdf)
    dLdW = np.outer(dLdb,x)

    return dLdW, dLdV, dLdb, dLdc
```

Answer 13

1. $\frac{dL}{dW} = \begin{pmatrix} -0.18070664 & -0.36141328 \\ -0.18070664 & -0.36141328 \\ 0.0 & 0.0 \end{pmatrix}$

2. $\frac{dL}{dV} = \begin{pmatrix} -0.45257413 & 0.45257413 & 0.48201379 \\ 0.45257413 & -0.45257413 & -0.48201379 \end{pmatrix}$

3. $\frac{dL}{db} = \begin{bmatrix} -0.18070664 & -0.18070664 & 0.0 \end{bmatrix}$

4. $\frac{dL}{dc} = \begin{bmatrix} 0.5 & -0.5 \end{bmatrix}$

Answer 14

```python
from autograd import grad
from autograd import numpy as np

def prediction_loss(x,y,W,V,b,c):
    Wx = np.matmul(W, x)
    b_plus_Wx = np.add(b, Wx)
    sigma = np.tanh(b_plus_Wx)
    f = np.add(c, np.matmul(V, sigma))
    softmax = 0
    for item in f:
        softmax += np.exp(item)

    softmax = np.log(softmax)
    return -f[y] + softmax

def prediction_grad_autograd(x,y,W,V,b,c):
    dLdW = grad(prediction_loss, 2)(x,y,W,V,b,c)
    dLdV = grad(prediction_loss, 3)(x,y,W,V,b,c)
    dLdb = grad(prediction_loss, 4)(x,y,W,V,b,c)
    dLdc = grad(prediction_loss, 5)(x,y,W,V,b,c)
    return dLdW, dLdV, dLdb, dLdc
```

Answer 15

```
from autograd import numpy as np

def prediction_loss_full(X,Y,W,V,b,c,l):
    WX = np.matmul(W, np.transpose(X))
    b = np.array([b, ] * X.shape[0]).transpose()
    c = np.array([c, ] * X.shape[0]).transpose()
    b_plus_WX = np.add(b, WX)
    sigma = np.tanh(b_plus_WX)
    f = np.add(c, np.matmul(V, sigma))

    L = 0
    for i in range(Y.shape[0]) :
        softmax = np.log(np.sum(np.exp(f[:,i])))
        y = Y[i]
        L += -f[y][i] + softmax

    L += l*(np.sum(np.square(V)) + np.sum(np.square(W)))
    return L
```

Answer 16

```
from autograd import grad

def prediction_grad_full(X,Y,W,V,b,c,l):
    dLdW = grad(prediction_loss_full, 2)(X, Y, W, V, b, c, l)
    dLdV = grad(prediction_loss_full, 3)(X, Y, W, V, b, c, l)
    dLdb = grad(prediction_loss_full, 4)(X, Y, W, V, b, c, l)
    dLdc = grad(prediction_loss_full, 5)(X, Y, W, V, b, c, l)

    return dLdW, dLdV, dLdb, dLdc
```

Answer 17

1. Below is the table describing the total training time (in ms) for all iterations.

M	Total time taken in Milliseconds
5	23919335
40	24029664
70	24103272

Table 9: Total time taken for 1000 iterations of gradient descent

2. Plot of regularized loss as a function of number of iterations -