Homework 5 - Berkeley STAT 157

Your name: XX, SID YY (Please add your name, and SID to ease Ryan and Rachel to grade.)

Please submit your homework through gradescope (http://gradescope.com/) instead of Github, so you will get the score distribution for each question. Please enroll in the class (https://www.gradescope.com/courses/42432) by the Entry code: MXG5G5

Handout 2/19/2019, due 2/26/2019 by 4pm in Git by committing to your repository.

In this homework, we will model covariate shift and attempt to fix it using logistic regression. This is a fairly realistic scenario for data scientists. To keep things well under control and understandable we will use Fashion-MNIST (http://d2l.ai/chapter_linear-networks/fashion-mnist.html) as the data to experiment on.

Follow the instructions from the Fashion MNIST notebook to get the data.

In [1]:
```
%matplotlib inline
from mxnet import autograd, gluon, init, nd
from mxnet.gluon import data as gdata, loss as gloss, nn, utils
import numpy as np
import d2l

mnist_train = gdata.vision.FashionMNIST(train=True)
mnist_test = gdata.vision.FashionMNIST(train=False)
```

1. Logistic Regression

1. Implement the logistic loss function $l(y,f) = -\log(1 + \exp(-yf))$ in Gluon.
2. Plot its values and its derivative for $y = 1$ and $f \in [-5, 5]$, using automatic differentiation in Gluon.
3. Generate training and test datasets for a binary classification problem using Fashion-MNIST with class 1 being a combination of `shirt` and `sweater` and class -1 being the combination of `sandal` and `sneaker` categories.
4. Train a binary classifier of your choice (it can be linear or a simple MLP such as from a previous lecture) using half the data (i.e. $12,000$ observations mixed as abvove) and one using the full dataset (i.e. $24,000$ observations as arising from the 4 categories) and report its accuracy.

Hint - you should encapsulate the training and reporting code in a callable function since you'll need it quite a bit in the following.

In [2]:
```
def loss(y,f):
    l = -nd.log(1+nd.exp(-y * f))
    return l

def dloss(y, f):
    f.attach_grad()
    with autograd.record():
        z = loss(y, f)
    z.backward()
    return f.grad
```

In [3]:
```
print(len(mnist_train), len(mnist_test))
print(len(mnist_train[0]), len(mnist_test[0]))
print(len(mnist_train[0][0]), len(mnist_test[0][0]))
print(len(mnist_train[0][0][0]), len(mnist_test[0][0][0]))
print(len(mnist_train[0][0][0][0]), len(mnist_test[0][0][0][0]))
```

```
60000 10000
2 2
28 28
28 28
1 1
```

In [4]:
```
def get_fashion_mnist_labels(labels):
    text_labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat',
                   'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
    return [text_labels[int(i)] for i in labels]

def get_new_labels(labels):
    return ['shirt' if i == -1 else 'shoe' for i in labels]

def converter(label):
    if label == 2 or label == 6:
        return -1
    return 1

def convert_data(mnist):
    indices = (mnist[:][1] == 5) | (mnist[:][1] == 6) \
    | (mnist[:][1] == 7) | (mnist[:][1] == 2)
    data, labels = mnist[:]
    data = data.asnumpy()[indices].astype('float32')
    labels = labels[indices].astype('float32')
    labels = np.array(list(map(converter, labels))).astype('float32')
    return data, labels

train_data, train_labels = convert_data(mnist_train)
test_data, test_labels = convert_data(mnist_test)
print(train_data.shape, train_labels.shape, \
      test_data.shape, test_labels.shape)
```

```
(24000, 28, 28, 1) (24000,) (4000, 28, 28, 1) (4000,)
```

In [5]:
```
def show_fashion_mnist(images, labels):
    d2l.use_svg_display()
    # Here _ means that we ignore (not use) variables
    _, figs = d2l.plt.subplots(1, len(images), figsize=(12, 12))
    for f, img, lbl in zip(figs, images, labels):
        f.imshow(img.reshape((28, 28)))
        f.set_title(lbl)
        f.axes.get_xaxis().set_visible(False)
        f.axes.get_yaxis().set_visible(False)
X, y = mnist_train[0:9]
show_fashion_mnist(train_data[:9], get_new_labels(train_labels[:9]))
print(X.shape)
```

```
(9, 28, 28, 1)
```

In [ ]:

In [ ]:

In [ ]:

In [6]:
```
batch_size = 256
train_iter = \
gdata.DataLoader(gluon.data.ArrayDataset(train_data, train_labels)\
                 , batch_size=batch_size, shuffle=True)
test_iter = \
gdata.DataLoader(gluon.data.ArrayDataset(test_data, test_labels)\
                 , batch_size=batch_size, shuffle=True)
```

In [7]:
```
net = nn.Sequential()
net.add(nn.Dense(1))
net.initialize(init.Normal(sigma=0.01))

loss = gloss.LogisticLoss()
trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate'
: 0.1})

num_epochs, lr = 5, 0.1

def ev_ac(out, y):
    e = lambda x: -1 if x < 0.5 else 1
    return ((nd.array(list(map(e, out))) == y).sum().asscalar())
def evaluate_accuracy(data_iter, net):
    acc_sum, n = 0.0, 0
    for X, y in data_iter:
        y = y.astype('float32')
        #print('net', net(X)[:5], 'y', y[:5])
        acc_sum += ev_ac(net(X), y)
        n += y.size
    return acc_sum / n

# This function has been saved in the d2l package for future use
def train_ch3(net, train_iter, test_iter, loss, num_epochs, batch_siz
e,
              params=None, lr=None, trainer=None, v=True, a=False):
    for epoch in range(num_epochs):
        train_l_sum, train_acc_sum, n = 0.0, 0.0, 0
        for X, y in train_iter:
            with autograd.record():
                y_hat = net(X)
                l = loss(y_hat, y).sum()
            l.backward()
            if trainer is None:
                d2l.sgd(params, lr, batch_size)
            else:
                # This will be illustrated in the next section
                trainer.step(batch_size)
            y = y.astype('float32')
            train_l_sum += l.asscalar()
            train_acc_sum += ev_ac(y_hat, y)
            n += y.size
        test_acc = evaluate_accuracy(test_iter, net)
        if v:
            print('epoch %d, loss %.4f, train acc %.3f, test acc %.3f
'
              % (epoch + 1, train_l_sum / n, train_acc_sum / n, test_
acc))
    if a:
        print('loss %.4f, test acc %.3f'
          % (train_l_sum / n, test_acc))
```

In [8]:
```
train_ch3(net, train_iter, test_iter, loss, num_epochs,
          batch_size, None, lr, trainer)
```

```
epoch 1, loss 420.5114, train acc 0.980, test acc 0.996
epoch 2, loss 31.7801, train acc 0.997, test acc 0.998
epoch 3, loss 21.1314, train acc 0.998, test acc 0.998
epoch 4, loss 17.7031, train acc 0.998, test acc 0.998
epoch 5, loss 14.8010, train acc 0.999, test acc 0.999
```

In [ ]:

2. Covariate Shift

Your goal is to introduce covariate shit in the data and observe the accuracy. For this, compose a dataset of $12,000$ observations, given by a mixture of `shirt` and `sweater` and of `sandal` and `sneaker` respectively, where you use a fraction $\lambda \in \{0.05, 0.1, 0.2, \ldots 0.8, 0.9, 0.95\}$ of one and a fraction of $1-\lambda$ of the other datasets respectively. For instance, you might pick for $\lambda = 0.1$ a total of 600 `shirt` and $5,400$ `sweater` images and likewise 600 `sandal` and $5,400$ `sneaker` photos, yielding a total of $12,000$ images for training. Note that the test set remains unbiased, composed of $2,000$ photos for the `shirt` + `sweater` category and of the `sandal` + `sneaker` category each.

1. Generate training sets that are appropriately biased. You should have 11 datasets.
2. Train a binary classifier using this and report the test set accuracy on the unbiased test set.

In [9]:
```
def convert_data(mnist):
    indices = (mnist[:][1] == 5) | (mnist[:][1] == 6) \
    | (mnist[:][1] == 7) | (mnist[:][1] == 2)
    data, labels = mnist[:]
    data = data.asnumpy()[indices].astype('float32')
    labels = labels[indices].astype('float32')
    labels = np.array(list(map(converter, labels))).astype('float32')
    return data, labels

def gen(train, labels, l):
    num = 6000
    permuted = np.random.permutation(len(train))
    train, labels = train[permuted], labels[permuted]
    indices = labels == -1
    tn1, ln1 = train[indices], labels[indices]
    t1, l1 = train[~indices], labels[~indices]
    b = int(num*l)
    r1, r2 = np.concatenate([t1[b:num], tn1[:b]]),\
             np.concatenate([l1[b:num], ln1[:b]])

    permuted = np.random.permutation(num)
    return r1[permuted], r2[permuted]
```

In [10]:
```
gen(train_data, train_labels, .1); print('hi')
```

```
hi
```

In [21]:

```
def train(ls=[.05] + [x/10 for x in range(1,10)] + [.95]):
    for l in ls:
        print("lambda:", l)

        td, tl = gen(train_data, train_labels, l)
        print(sum(tl == -1))
        train_iter = \
        gdata.DataLoader(gluon.data.ArrayDataset(td, tl)\
                 , batch_size=batch_size, shuffle=True)
        test_iter = \
        gdata.DataLoader(gluon.data.ArrayDataset(test_data, test_labe
ls)\
                         , batch_size=batch_size, shuffle=True)


        net = nn.Sequential()
        net.add(nn.Dense(1))
        net.initialize(init.Normal(sigma=0.01))

        loss = gloss.LogisticLoss()
        trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learni
ng_rate': 0.01})

        train_ch3(net, train_iter, test_iter, loss, num_epochs,
          batch_size, None, lr, trainer, False, True)

train()
```

```
lambda: 0.05
300
loss 1.9714, test acc 0.985
lambda: 0.1
600
loss 1.9366, test acc 0.992
lambda: 0.2
1200
loss 1.0862, test acc 0.996
lambda: 0.3
1800
loss 1.6848, test acc 0.997
lambda: 0.4
2400
loss 1.5750, test acc 0.997
lambda: 0.5
3000
loss 2.8005, test acc 0.996
lambda: 0.6
3600
loss 3.5741, test acc 0.997
lambda: 0.7
4200
loss 4.3835, test acc 0.995
lambda: 0.8
4800
loss 3.3879, test acc 0.993
lambda: 0.9
5400
loss 4.8682, test acc 0.989
lambda: 0.95
5700
loss 6.1234, test acc 0.971
```

3. Covariate Shift Correction

Having observed that covariate shift can be harmful, let's try fixing it. For this we first need to compute the appropriate propensity scores $\frac{dp(x)}{dq(x)}$. For this purpose pick a biased dataset, let's say with $\lambda = 0.1$ and try to fix the covariate shift.

1. When training a logistic regression binary classifier to fix covariate shift, we assumed so far that both sets are of equal size. Show that re-weighting data in training and test set appropriately can help address the issue when both datasets have different size. What is the weighting?
2. Train a binary classifier (using logistic regression) distinguishing between the biased training set and the unbiased test set. Note - you need to weigh the data.
3. Use the scores to compute weights on the training set. Do they match the weight arising from the biasing distribution λ?
4. Train a binary classifier of the covariate shifted problem using the weights obtained previously and report the accuracy. Note - you will need to modify the training loop slightly such that you can compute the gradient of a weighted sum of losses.

1) We want number_of_negative_class * weight = number_of_positive_class

```
In [ ]:
```

In [22]:
```
# This function has been saved in the d2l package for future use
def train_with_weights(f, net, train_iter, test_iter, loss, num_epoch
s, batch_size,
              params=None, lr=None, trainer=None, v=True, a=False):
    for epoch in range(num_epochs):
        train_l_sum, train_acc_sum, n = 0.0, 0.0, 0
        for X, y in train_iter:
            with autograd.record():
                y_hat = nd.multiply(net(X), f(X))
                l = loss(y_hat, y).sum()
            l.backward()
            if trainer is None:
                d2l.sgd(params, lr, batch_size)
            else:
                # This will be illustrated in the next section
                trainer.step(batch_size)
            y = y.astype('float32')
            train_l_sum += l.asscalar()
            train_acc_sum += ev_ac(y_hat, y)
            n += y.size
        test_acc = evaluate_accuracy(test_iter, net)
        if v:
            print('epoch %d, loss %.4f, train acc %.3f, test acc %.3f
'
              % (epoch + 1, train_l_sum / n, train_acc_sum / n, test_
acc))
    if a:
        print('loss %.4f, test acc %.3f'
          % (train_l_sum / n, test_acc))


def train(ls=[.05] + [x/10 for x in range(1,10)] + [.95]):
    for l in ls:
        print("lambda:", l)

        td, tl = gen(train_data, train_labels, l)
        print(sum(tl == -1))

        train_iter = \
        gdata.DataLoader(gluon.data.ArrayDataset(\
        np.concatenate([td, test_data]),\
        np.concatenate([np.ones(len(td)).astype('float32'),\
        (np.ones(len(test_data))*-1).astype('float32')]))\
                 , batch_size=batch_size, shuffle=True)
        test_iter = \
        gdata.DataLoader(gluon.data.ArrayDataset(\
        np.concatenate([td, test_data]),\
        np.concatenate([np.ones(len(td)).astype('float32'),\
        (np.ones(len(test_data))*-1).astype('float32')]))\
                 , batch_size=batch_size, shuffle=True)
        f = nn.Sequential()
        f.add(nn.Dense(1))
        f.initialize(init.Normal(sigma=0.01))

        loss = gloss.LogisticLoss()
        trainer = gluon.Trainer(f.collect_params(), 'sgd', {'learning
```

```
_rate': 0.01})

        train_ch3(f, train_iter, test_iter, loss, num_epochs,
          batch_size, None, lr, trainer, False, True)



        train_iter = \
        gdata.DataLoader(gluon.data.ArrayDataset(td, tl)\
                 , batch_size=batch_size, shuffle=True)
        test_iter = \
        gdata.DataLoader(gluon.data.ArrayDataset(test_data, test_labe
ls)\
                         , batch_size=batch_size, shuffle=True)



        net = nn.Sequential()
        net.add(nn.Dense(1))
        net.initialize(init.Normal(sigma=0.01))

        loss = gloss.LogisticLoss()
        trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learni
ng_rate': 0.01})

        train_with_weights(f, net, train_iter, test_iter, loss, num_e
pochs,
          batch_size, None, lr, trainer, False, True)



train()
```

```
lambda: 0.05
300
loss 1791.4049, test acc 0.722
loss 43609246.3147, test acc 0.416
lambda: 0.1
600
loss 2069.5493, test acc 0.707
loss 78256105.0453, test acc 0.401
lambda: 0.2
1200
loss 3003.4900, test acc 0.672
loss 354647593.9840, test acc 0.477
lambda: 0.3
1800
loss 3976.1804, test acc 0.400
loss 478789471.5733, test acc 0.004
lambda: 0.4
2400
loss 4641.7990, test acc 0.600
loss 125925087.2863, test acc 0.997
lambda: 0.5
3000
loss 5324.4973, test acc 0.550
loss 149066612.3947, test acc 0.529
lambda: 0.6
3600
loss 5371.7406, test acc 0.600
loss 777692288.3413, test acc 0.995
lambda: 0.7
4200
loss 5468.5438, test acc 0.600
loss 8589824308.5653, test acc 0.992
lambda: 0.8
4800
loss 5243.2512, test acc 0.658
loss 161604589.1413, test acc 0.467
lambda: 0.9
5400
loss 5411.0195, test acc 0.738
loss 32919559.4773, test acc 0.501
lambda: 0.95
5700
loss 4839.1843, test acc 0.701
loss 85047661385.0453, test acc 0.500
```

In [ ]: