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/)](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)](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)](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 [50]:
```
%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
from matplotlib import pyplot as plt
import mxnet as mx

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

1. Logistic Regression

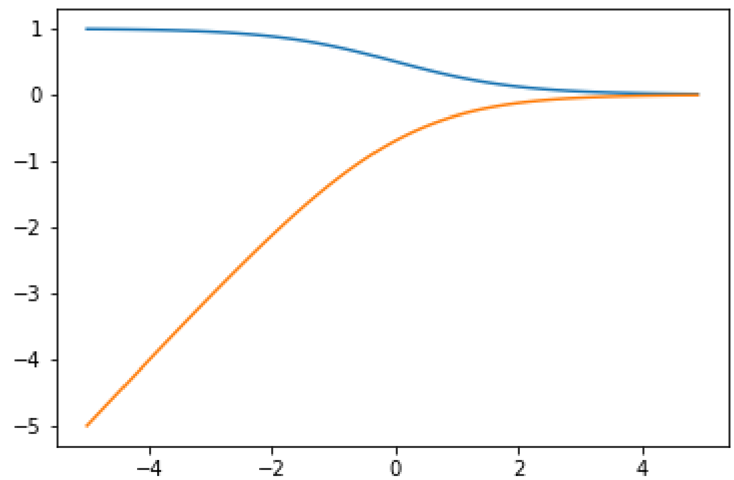
1. Implement the logistic loss function $l(y, f) = -\log(1 + \exp(-yf))$ in Gluon.
2. Plot its values and its derivative for $y = 1$ and $f \in [-5, 5]$, using automatic differentiation in Gluon.
3. Generate training and test datasets for a binary classification problem using Fashion-MNIST with class 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 [51]:
```
def logisticLoss(f, y):
    return -nd.log(1 + nd.exp(-y * f))

y = nd.ones(1)
f = nd.arange(-5, 5, 0.1)
f.attach_grad()
with autograd.record():
    l = logisticLoss(f, y)
l.backward()

plt.plot(f.asnumpy(), f.grad.asnumpy())
plt.plot(f.asnumpy(), l.asnumpy())
plt.show()
```



In [95]:
```
train_features = mnist_train[:][0]
train_labels = mnist_train[:][1]
test_features = mnist_test[:][0]
test_labels = mnist_test[:][1]
```

In [62]:
```
train_shoes = [nd.array(x, dtype='float32') for x, y in zip(train_featur
es, train_labels) if y == 5 or y == 7]
train_cloths = [nd.array(x, dtype='float32') for x, y in zip(train_featu
res, train_labels) if y == 2 or y == 6]
train = train_shoes + train_cloths

a = nd.repeat(nd.array([-1], dtype='float32'), repeats = 12000)
b = nd.repeat(nd.array([1], dtype='float32'), repeats = 12000)
train_label = nd.concat(a, b, dim=0)
```

In [63]:
```
test_shoes = [nd.array(x, dtype='float32') for x, y in zip(test_features
, test_labels) if y == 5 or y == 7]
test_cloths = [nd.array(x, dtype='float32') for x, y in zip(test_feature
s, test_labels) if y == 2 or y == 6]
test = test_shoes + test_cloths

a = nd.repeat(nd.array([-1], dtype='float32'), repeats = 2000)
b = nd.repeat(nd.array([1], dtype='float32'), repeats = 2000)
test_label = nd.concat(a, b, dim=0)
```

In [64]:
```
batch_size = 64
```

In [87]:
```
def get_iter(train, train_label, test, test_label):
    train_iter = gdata.DataLoader(gdata.ArrayDataset(
        train, train_label), batch_size, shuffle=True)

    test_iter = gdata.DataLoader(gdata.ArrayDataset(
        test, test_label), batch_size, shuffle=False)
```

In [88]:
```
def getnet():
    net = nn.Sequential()
    net.add(nn.Dense(1))
    net.initialize()
    return net
```

In [ ]:
```
train_iter, test_iter = get_iter(train, train_label, test, test_label)

num_epochs = 5
# los = logisticLoss
los = gloss.LogisticLoss(label_format='signed')
net = getnet()
trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate':0.
05})
```

In [74]:
```
def accuracy(y_hat, y):
    return (y_hat.argmax(axis=1) == y.astype('float32')).mean().asscalar
()
def evaluate_accuracy(data_iter, net):
    acc_sum, n = 0.0, 0
    for X, y in data_iter:
        y = y.astype('float32')
        acc_sum += (net(X) > 0).sum().asscalar()
        n += y.size
    return acc_sum / n
```

In [77]:
```
for epoch in range(num_epochs):
    for X, y in train_iter:
        with autograd.record():
#             print(net(X[1:4]))
#             print(y[1:4])
            l = los(net(X), y)
        l.backward()
        trainer.step(batch_size)
    test_acc = evaluate_accuracy(test_iter, net)
    print(test_acc)
```

```
0.5
0.5
0.5
0.5
0.5
```

In [78]:
```
# net[0].weight.data()
```

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 [79]:
```
lambd = nd.concat(nd.array([0.05]) , nd.arange(0.1, 1, 0.1), nd.array([
0.95]), dim=0)
lambd
one_minus_lambd = 1 - lambd
one_minus_lambd
lambd
```

Out[79]:
```
[0.05       0.1        0.2        0.3        0.4        0.5
 0.6        0.70000005 0.8        0.90000004 0.95      ]
<NDArray 11 @cpu(0)>
```

In [80]:
```
fraction = nd.array(6000 * lambd, dtype='int')
fraction
the_rest = 6000 - fraction
the_rest
```

Out[80]:
```
[5700 5400 4800 4200 3600 3000 2400 1800 1200  600  300]
<NDArray 11 @cpu(0)>
```

In [96]:
```
train_dataset = []
train_label = []
for k in range(len(lambd)):
    frac = fraction[k].asscalar()
    rest = the_rest[k].asscalar()
    sandal = [nd.array(x, dtype='float32') for x, y in zip(train_feature
s, train_labels) if y == 5]
    sandal = sandal[:frac]
    sneaker = [nd.array(x, dtype='float32') for x, y in zip(train_featur
es, train_labels) if y == 7]
    sneaker = sneaker[:rest]
    train_shoes = sandal + sneaker
    label_shoes = nd.repeat(nd.array([-1], dtype='float32'), repeats = 6
000)
    shirt = [nd.array(x, dtype='float32') for x, y in zip(train_features
, train_labels) if y == 6]
    shirt = shirt[:frac]
    sweater = [nd.array(x, dtype='float32') for x, y in zip(train_featur
es, train_labels) if y == 2]
    sweater = sweater[:rest]
    train_cloths = shirt + sweater
    label_cloths = nd.repeat(nd.array([1], dtype='float32'), repeats = 6
000)
    # print(len(train_shoes))
    train_dataset.append(train_shoes + train_cloths)
    train_label.append(nd.concat(label_shoes, label_cloths, dim=0))
    # print(len(nd.concat(label_shoes, label_cloths, dim=0)))
```

In [97]:
```
test_shoes = [nd.array(x, dtype='float32') for x, y in zip(test_features
, test_labels) if y == 5 or y == 7]
test_cloths = [nd.array(x, dtype='float32') for x, y in zip(test_feature
s, test_labels) if y == 2 or y == 6]
test_dataset = test_shoes + test_cloths
a = nd.repeat(nd.array([-1], dtype='float32'), repeats = 2000)
b = nd.repeat(nd.array([1], dtype='float32'), repeats = 2000)
test_label = nd.concat(a, b, dim=0)
```

In [99]:
```
for k in range(11):
    train_iter, test_iter = get_iter(train_dataset[k], train_label[k], t
est_dataset, test_label)
    los = gloss.LogisticLoss(label_format='signed')
    net = getnet()
    num_epochs = 5
    trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rat
e': 0.1})
    train(train_iter , test_iter)
```

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.

$$\int dxq(x)f(x) = \int dxp(x)\frac{q(x)}{p(x)}f(x) = \int dxp(x)\alpha(x)f(x)$$

$$r(x, y) = \frac{1}{2}[p(x)\delta(y, 1) + q(x)\delta(y, -1)]$$

$$r(y = 1|x) = \frac{p(x)}{p(x)+q(x)} \text{ and hence } \alpha = \frac{q(x)}{p(x)} = \frac{r(y=-1|x)}{r(y=1|x)}$$

$$r(y = 1|x) = \frac{1}{1+\exp(-f(x))}$$

$$\alpha(x) = \frac{r(y=-1|x)}{r(y=1|x)} = \exp(f(x))$$

the weighting is

$$\exp(f(x_i))$$

In [ ]:
```
weights = []
for k in range(11):
    train_iter, test_iter = get_iter(train_dataset[k], train_label[k], t
est_dataset, test_label)
    los = gloss.SigmoidBinaryCrossEntropyLoss()
    net = getnet()
    num_epochs = 5
    trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rat
e': 0.1})
    train(train_iter , test_iter)
    weight = []
    for X, y in train_iter:
        weight.append(nd.exp(net(X)))
    weights.append(weight)
```

In [ ]:
```
len(weights)
```

In [ ]:
```
def train2(train_iter , test_iter, weight):
    for epoch in range(num_epochs):
        for X, y in train_iter:
            with autograd.record():
                yhat = net(X)
                l = weight * los(yhat, y)
            l.backward()
            trainer.step(batch_size)
        test_acc = evaluate_accuracy(test_iter, net)
        print(test_acc)

for k in range(11):
    net = getnet()
    train_iter, test_iter = get_iter(train_dataset[k], train_label[k], t
est_dataset, test_label)
    los = gloss.SigmoidBinaryCrossEntropyLoss()
    net = getnet()
    num_epochs = 5
    trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rat
e': 0.1})
    train2(train_iter, test_iter, weights[k])
```