CSE 546 HW #2

Sam Kowash

November 1, 2018

Acknowledgments: I collaborated on parts of this homework with Tyler Blanton, Michael Ross, and Nick Ruof.

(1) A Taste of Learning Theory

1. Let $X \in \mathbb{R}^d$ a random feature vector, and $Y \in \{1, \ldots, K\}$ a random label for $K \in \mathbb{N}$ with joint distribution P_{XY}. We consider a randomized classifier $\delta(x)$ which maps a value $x \in \mathbb{R}^d$ to some $y \in \{1, \ldots, K\}$ with probability $\alpha(x, y) \equiv P(\delta(x) = y)$ subject to $\sum_{y=1}^{K} \alpha(x, y) = 1$ for all x. The risk of the classifier δ is

$$R(\delta) \equiv \mathbb{E}_{XY,\delta}\left[\mathbf{1}\{\delta(X) \neq Y\}\right],$$

which we should interpret as the expected rate of misclassification. A classifier δ is called deterministic if $\alpha(x, y) \in \{0, 1\}$ for all x, y. Further, we call a classifier δ_* a Bayes classifier if $\delta_* \in \arg\inf_\delta R(\delta)$.

If we first take the expectation over outcomes of δ (by conditioning on X and Y), we find

$$R(\delta) = \mathbb{E}_{XY}\left[1 - \alpha(X, Y)\right],$$

since the indicator function is 1 except for the single outcome where $\delta(x) = y$, which occurs with probability $\alpha(x, y)$. It is then clear that minimizing $R(\delta)$ is equivalent to *maximizing* $\mathbb{E}_{XY}[\alpha(X, Y)]$; the assignments of $\alpha(x, y)$ which do this are our Bayes optimal classifiers.

2. Suppose we grab n data samples (x_i, y_i) i.i.d. from P_{XY} where $y_i \in \{-1, 1\}$ and $x_i \in \mathcal{X}$ where $\mathcal{X}$ is some set. Let $f : \mathcal{X} \to \{-1, 1\}$ be a deterministic classifier with true risk

$$R(f) = \mathbb{E}_{XY}\left[\mathbf{1}(f(X) \neq Y)\right].$$

and empirical risk

$$\hat{R}_n(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(f(x_i) \neq y_i).$$

(a) We wish to estimate the true risk of some classifier $\tilde{f}$. If we -

(2) Programming

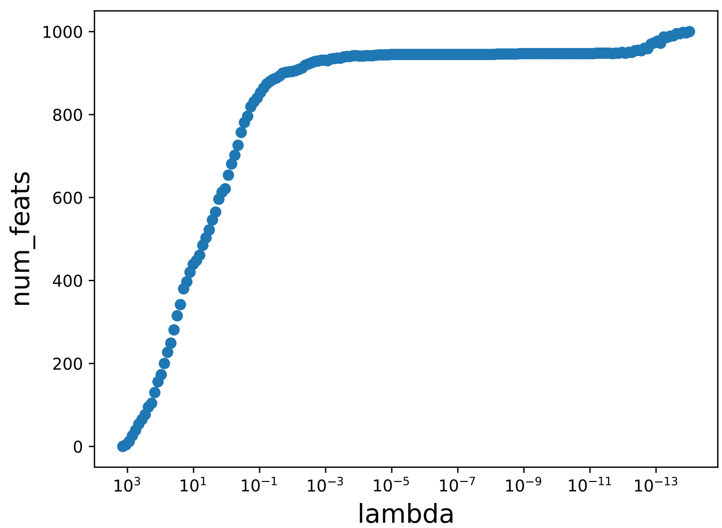

Figure 2.1: Number of nonzero features vs. λ for lasso on synthetic data

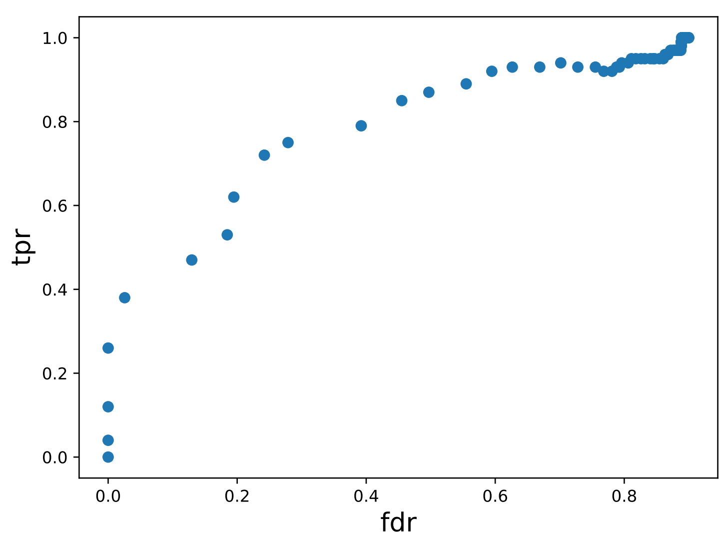

Figure 2.2: TPR vs. FDR for lasso on synthetic data
-

1.
2. Optimal λ was near 1.11.

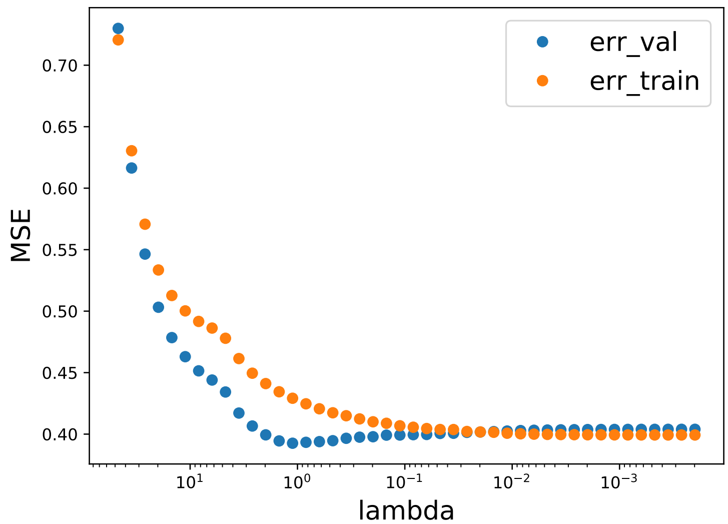

Figure 2.3: Validation and training error vs λ on Yelp data

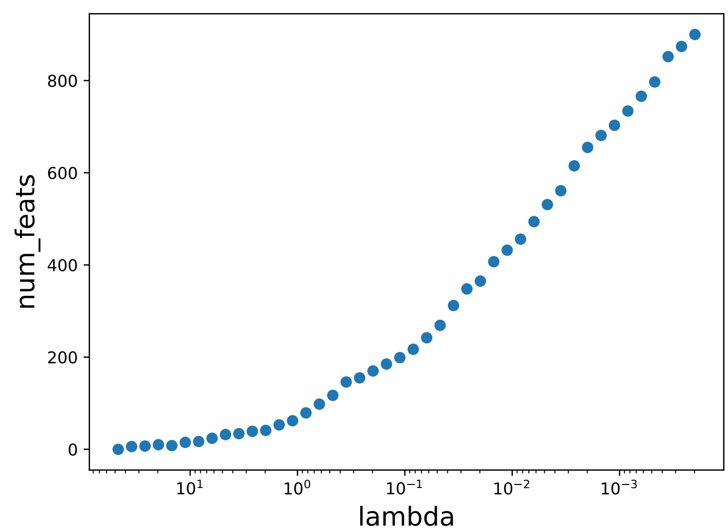

Figure 2.4: Number of nonzero features vs. λ on Yelp data

3. We now consider binary classification between 2s and 7s in the MNIST set via regularized logistic regression. We choose a balanced target set $Y \in \{-1, 1\}$, where $Y = -1$ for 2s and $Y = 1$ for 7s, so that our data are $\{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{Z}_2$. The L_2-regularized negative log likelihood objective to be minimized is

$$J(w, b) = \frac{1}{n} \sum_{i=1}^{n} \log \left[1 + \exp\left(-y_i(b + x_i^T w)\right)\right] + \lambda \|w\|_2^2.$$

For convenience, we define the functions

$$\mu_i(w, b) = \frac{1}{1 + \exp\left[-y_i(b + x_i^T w)\right]}.$$

.

(a) To do gradient descent, we need to know some gradients. First,

$$\begin{aligned}
\nabla_w J(w,b) &= \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i x_i \exp\left[-y_i(b + x_i^T w)\right]}{1 + \exp\left[-y_i(b + x_i^T w)\right]} + 2\lambda w \\
\nabla_w J(w,b) &= -\frac{1}{n}\sum_{i=1}^{n} \mu_i \left(\frac{1}{\mu_i} - 1\right) y_i x_i + 2\lambda w \\
\nabla_w J(w,b) &= \frac{1}{n}\sum_{i=1}^{n} (\mu_i - 1) y_i x_i + 2\lambda w.
\end{aligned}$$

Next,

$$\begin{aligned}
\nabla_b J(w,b) &= -\frac{1}{n}\sum_{i=1}^{n} \frac{y_i \exp\left[-y_i(b + x_i^T w)\right]}{1 + \exp\left[-y_i(b + x_i^T w)\right]} \\
\nabla_b J(w,b) &= \frac{1}{n}\sum_{i=1}^{n} (\mu_i - 1) y_i.
\end{aligned}$$

We'll also want some Hessians, for Newton's method.

$$\begin{aligned}
\nabla_w^2 J(w,b) &= \frac{1}{n}\sum_{i=1}^{n} y_i (\nabla_w \mu_i) x_i^T + 2\lambda I_d \\
\nabla_w \mu_i &= \frac{y_i x_i \exp\left[-y_i(b + x_i^T w)\right]}{\left(1 + \exp\left[-y_i(b + x_i^T w)\right]\right)^2} = \mu_i^2 \left(\frac{1}{\mu_i} - 1\right) y_i x_i = \mu_i (1 - \mu_i) y_i x_i \\
\nabla_w^2 J(w,b) &= \frac{1}{n}\sum_{i=1}^{n} \mu_i (1 - \mu_i) y_i^2 x_i x_i^T + 2\lambda I_d
\end{aligned}$$

Lastly,

$$\begin{aligned}
\nabla_b^2 J(w,b) &= \frac{1}{n}\sum_{i=1}^{n} (\nabla_b \mu_i) y_i \\
\nabla_b \mu_i &= \frac{y_i \exp\left[-y_i(b + x_i^T w)\right]}{\left(1 + \exp\left[-y_i(b + x_i^T w)\right]\right)^2} = \mu_i (1 - \mu_i) y_i \\
\nabla_b^2 J(w,b) &= \frac{1}{n}\sum_{i=1}^{n} \mu_i (1 - \mu_i) y_i^2.
\end{aligned}$$

```python
#!/usr/bin/env python
# lasso.py
import numpy as np
import matplotlib.pyplot as plt


def descend_step(X, Y, w_curr, lam):
    d = X.shape[1]
    w = w_curr
    b = np.mean(Y - np.matmul(X,np.transpose(w)))

    xks = {k: X[:,k] for k in range(d)}
    a = {k: 2*np.matmul(xks[k],xks[k]) for k in range(d)}

    for k in range(d):
        wk = np.copy(w)
        xk = xks[k]
        wk[k] = 0 #faster way to knock out a column

        c = 2*np.matmul(xk, Y - b - np.matmul(X, np.transpose(wk)))

        if np.abs(c) <= lam:
            w[k] = 0
        else:
            w[k] = (c - np.sign(c)*lam)/a[k]

    return w



def lasso_descend(X, Y, w_init, lam, thresh):
    n = X.shape[0]
    d = X.shape[1]

    if len(w_init) != d:
        print("Initial guess dimension mismatch. Setting all zeros.")
        w_init = np.zeros(d)


    #do at least one step
    w_last = np.copy(w_init)
    w_curr = descend_step(X, Y, w_init, lam)

    i=0
    while np.max(np.abs(w_curr-w_last)) > thresh:
        i += 1
        print("on descent step %s" %i)
        w_last = np.copy(w_curr)
```

```
    w_curr = descend_step(X, Y, w_curr, lam)
b = np.mean(Y - np.matmul(X,np.transpose(w_curr)))
return w_curr, b
```

```python
#!/usr/bin/env python
# synth_lasso.py
import numpy as np
import matplotlib.pyplot as plt
import sys
from datetime import datetime
from lasso import *





n = 500
d = 1000
k = 100
sig = 1

w_true = np.append(np.arange(1,k+1)/k, np.zeros(d-k))

X_train = np.random.randn(n,d)
Y_train = np.matmul(X_train, np.transpose(w_true)) + np.random.randn(n)*sig

lam_max = np.max(2*np.abs(np.matmul(Y_train - np.mean(Y_train), X_train)))


lams = []
ws = []
num_feats = []
tprs = []
fdrs = []
xdrs = []
lam = lam_max

r = float(sys.argv[1])


it = 0
while (max(num_feats) if num_feats else 0) < d:
    it += 1
    print(f"On iter {it} with {num_feats[-1] if num_feats else 0} "
          f"features and lambda={lam:.5f}")
    lams.append(lam)
    w = lasso_descend(X_train, Y_train, ws[-1] if ws else np.zeros(d),
                      lam, 1e-3)[0]

    total_feats = np.count_nonzero(w)
    true_feats = np.count_nonzero(np.logical_and(w != 0, w_true != 0))
    false_feats = np.count_nonzero(np.logical_and(w != 0, w_true == 0))
```

```
    if total_feats != (true_feats + false_feats):
        print("Something is terribly wrong.")

    num_feats.append(total_feats)
    tprs.append(true_feats/k)
    fdrs.append(false_feats/total_feats if total_feats != 0 else 0)
    ws.append(w)
    lam *= r

lams = np.array(lams)
num_feats = np.array(num_feats)
tprs = np.array(tprs)
fdrs = np.array(fdrs)


ftime = datetime.now().time()
stamp = f"{ftime.hour:02d}_{ftime.minute:02d}_{ftime.second:02d}"
with open(f'data/synth-{stamp}','w') as f:
    f.writelines([f"{lams[i]:12e} {num_feats[i]:4d} "
                  f"{tprs[i]:14.8f} {fdrs[i]:14.8f}\n"
                  for i in range(len(lams))])
```

```python
#!/usr/bin/env python
# yelp_lasso.py
import numpy as np
import matplotlib.pyplot as plt
import sys
from datetime import datetime
from lasso import *




# Load data according to provided example
X = np.genfromtxt("data/upvote_data.csv", delimiter=",")
Y = np.loadtxt("data/upvote_labels.txt", dtype=np.int)
feature_names = open("data/upvote_features.txt").read().splitlines()

print("Data loaded.")

d = X.shape[1]

X_train = X[:4000]
Y_train = np.sqrt(Y[:4000])

X_val = X[4000:5000]
Y_val = np.sqrt(Y[4000:5000])

X_test = X[5000:]
Y_test = np.sqrt(Y[5000:])


lam_max = np.max(2*np.abs(np.matmul(Y_train - np.mean(Y_train), X_train)))


lams = []
num_feats = []
val_errs = []
train_errs = []
ws = []
bs = []

lam = lam_max
r = float(sys.argv[1])

it = 0
while (max(num_feats) if num_feats else 0) < .9*d:
    it += 1
    print(f"On iter {it} with {num_feats[-1] if num_feats else 0} "
```

```
            f"features and lambda={lam:5e}")
    lams.append(lam)
    w,b = lasso_descend(X_train, Y_train, (ws[-1] if ws else np.zeros(d)),
                        lam, 5e-2)
    ws.append(w)
    bs.append(b)


    val_pred = np.matmul(X_val,w) + b
    train_pred = np.matmul(X_train, w) + b

    val_diff = val_pred - Y_val
    train_diff = train_pred - Y_train

    feats = np.count_nonzero(w)
    val_err = np.matmul(val_diff, val_diff)
    train_err = np.matmul(train_diff, train_diff)

    num_feats.append(feats)
    val_errs.append(val_err)
    train_errs.append(train_err)

    lam *= r



ftime = datetime.now().time()
stamp = f"{ftime.hour:02d}_{ftime.minute:02d}_{ftime.second:02d}"
with open(f"data/yelp_sqrt-{stamp}", "w") as f:
    f.writelines([f"{lams[i]:12e} {num_feats[i]:4d} {val_errs[i]:14.6f} "
                  f"{train_errs[i]:14.6f}\n" for i in range(len(lams))])
```

```python
#!/usr/bin/env python
# grad_descend.py
import numpy as np
import matplotlib.pyplot as plt
from mnist import MNIST
from datetime import datetime
import sys


def load_test():
    mndata = MNIST('../../mnist/data/')
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_test = X_test/255.

    return X_test, labels_test

def load_train():
    mndata = MNIST('../../mnist/data/')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_train = X_train/255.

    return X_train, labels_train

def grad_w(mu, X, Y, w, lam):
    n = len(mu)
    return np.dot((mu-1)*Y,X)/n + 2*lam*w

def grad_b(mu, X, Y, w, lam):
    n = len(mu)
    return np.dot(mu-1, Y)/n

def hess_w(mu, X, Y, w, lam):
    n = len(mu)
    d = X.shape[1]
    hess = np.zeros(d,d)
    for i in range(n):
        hess += mu[i]*(1-mu[i])*Y[i]**2*np.outer(X[i],X[i])

    return hess/n + 2*lam*np.identity(d)

def hess_b(mu, X, Y, w, lam):
    n = len(mu)

    return np.sum(mu*(1-mu)*Y**2)/n


def objective(mu, X, Y, w, lam):
    n = len(mu)
```

```
    return np.sum(-1*np.log(mu))/n + lam*np.matmul(w,w)


def pred(X, w, b):
    return np.sign(b+ np.matmul(X,w))


def gdescend(X_train, Y_train, X_test, Y_test, lam=0.1, eta = 0.4, tol = 1e-4):
    d = X_train.shape[1]
    n_train = X_train.shape[0]
    n_test = X_test.shape[0]

    j_train = []
    j_test = []
    e_train = []
    e_test = []

    w = np.zeros(d)
    b = 0


    mu_train = 1/(1+np.exp(-1*Y_train*(b + np.matmul(X_train, w))))
    mu_test = 1/(1+np.exp(-1*Y_test*(b + np.matmul(X_test, w))))
    j_train.append(objective(mu_train, X_train, Y_train, w, lam))
    j_test.append(objective(mu_test, X_test, Y_test, w, lam))
    e_train.append(np.count_nonzero(pred(X_train, w, b) - Y_train)/n_train)
    e_test.append(np.count_nonzero(pred(X_test, w, b) - Y_test)/n_test)
    i=0
    print(f"Step {i}")


    while (len(j_train) < 2 or
            (np.abs(j_train[-1]-j_train[-2]) if len(j_train) > 1 else 0) > tol):
        i += 1
        print(f"Step {i}: delta = {(np.abs(j_train[-1]-j_train[-2]) "
              f"if len(j_train) > 1 else 0)}")
        gb = grad_b(mu_train, X_train, Y_train, w, lam)
        gw = grad_w(mu_train, X_train, Y_train, w, lam)

        b -= eta*gb
        w -= eta*gw

        mu_train = 1/(1+np.exp(-1*Y_train*(b + np.matmul(X_train, w))))
        mu_test = 1/(1+np.exp(-1*Y_test*(b + np.matmul(X_test, w))))

        j_train.append(objective(mu_train, X_train, Y_train, w, lam))
        j_test.append(objective(mu_test, X_test, Y_test, w, lam))
```

```
        e_train.append(np.count_nonzero(pred(X_train, w, b) - Y_train)/n_train)
        e_test.append(np.count_nonzero(pred(X_test, w, b) - Y_test)/n_test)

    return j_train,j_test,e_train,e_test




def sgdescend(X_train, Y_train, X_test, Y_test, batch, lam=0.1, eta=0.4, tol=1e-4)
    d = X_train.shape[1]
    n_train = X_train.shape[0]
    n_test = X_test.shape[0]

    j_train = []
    j_test = []
    e_train = []
    e_test = []

    w = np.zeros(d)
    b = 0



    mu_train = 1/(1+np.exp(-1*Y_train*(b + np.matmul(X_train, w))))
    mu_test = 1/(1+np.exp(-1*Y_test*(b + np.matmul(X_test, w))))

    j_train.append(objective(mu_train, X_train, Y_train, w, lam))
    j_test.append(objective(mu_test, X_test, Y_test, w, lam))
    e_train.append(np.count_nonzero(pred(X_train, w, b) - Y_train)/n_train)
    e_test.append(np.count_nonzero(pred(X_test, w, b) - Y_test)/n_test)
    i=0
    print(f"Step {i}")


    while (len(j_train) < 2 or
           (np.abs(j_train[-1]-j_train[-2]) if len(j_train) > 1 else 0) > tol):
        i += 1
        print(f"Step {i}: delta = {(np.abs(j_train[-1]-j_train[-2]) "
              f"if len(j_train) > 1 else 0)}")

        batch_ind = np.random.randint(0,n_train,batch)
        X_batch = X_train[batch_ind]
        Y_batch = Y_train[batch_ind]
        mu_batch = 1/(1+np.exp(-1*Y_batch*(b + np.matmul(X_batch, w))))

        gb = grad_b(mu_batch, X_batch, Y_batch, w, lam)
        gw = grad_w(mu_batch, X_batch, Y_batch, w, lam)
```

```
        b -= eta*gb
        w -= eta*gw

        mu_train = 1/(1+np.exp(-1*Y_train*(b + np.matmul(X_train, w))))
        mu_test = 1/(1+np.exp(-1*Y_test*(b + np.matmul(X_test, w))))

        j_train.append(objective(mu_train, X_train, Y_train, w, lam))
        j_test.append(objective(mu_test, X_test, Y_test, w, lam))

        e_train.append(np.count_nonzero(pred(X_train, w, b) - Y_train)/n_train)
        e_test.append(np.count_nonzero(pred(X_test, w, b) - Y_test)/n_test)

    return j_train,j_test,e_train,e_test




X_train, labels_train = load_train()
X_test, labels_test = load_test()

train_twosev = np.where(np.logical_or(labels_train == 2, labels_train == 7))
test_twosev =  np.where(np.logical_or(labels_test == 2, labels_test == 7))


X_train = X_train[train_twosev]
labels_train = labels_train[train_twosev]

X_test = X_test[test_twosev]
labels_test = labels_test[test_twosev]

codes = {2: -1, 7: 1}
Y_train = np.array([codes[i] for i in labels_train])
Y_test = np.array([codes[i] for i in labels_test])



if sys.argv[1] == 'gd':
    j_train,j_test,e_train,e_test = gdescend(X_train, Y_train, X_test, Y_test,
                                             eta=float(sys.argv[2]))
    now = datetime.now().time()
    stamp = f"{now.hour:02d}_{now.minute:02d}_{now.second:02d}"
    np.savez(f'data/gdescent-{stamp}', j_train=j_train, j_test=j_test,
             e_train=e_train, e_test=e_test)

elif sys.argv[1] == 'sgd':
```

```python
j_train, j_test, e_train, e_test = sgdescend(X_train, Y_train, X_test,
                                            Y_test, eta=float(sys.argv[2]),
                                            batch=int(sys.argv[3]))
now = datetime.now().time()
stamp = f"{now.hour:02d}_{now.minute:02d}_{now.second:02d}"
np.savez(f'data/sgdescent-{sys.argv[3]}-{stamp}', j_train=j_train,
         j_test=j_test, e_train=e_train, e_test=e_test)
```