SOLUTION MANUAL
Math 189r
Homework 1
September 19, 2016

There are 9 problems in this set. You need to do 3 problems (due in class on Monday) every week for 3 weeks. Note that this means you must eventually complete all problems. Feel free to work with other students, but make sure you write up the homework and code on your own (no copying homework *or* code; no pair programming). Feel free to ask students or instructors for help debugging code or whatever else, though. When implementing algorithms you may not use any library (such as `sklearn`) that already implements the algorithms but you may use any other library for data cleaning and numeric purposes (`numpy` or `pandas`). Use common sense. Problems are in no specific order.

**1** (**regression**). Download the data at `https://math189r.github.io/hw/data/online_news_popularity/online_news_popularity.csv` and the info file at `https://math189r.github.io/hw/data/online_news_popularity/online_news_popularity.txt`. Read the info file. Split the csv file into a training and test set with the first two thirds of the data in the training set and the rest for testing. Of the testing data, split the first half into a 'validation set' (used to optimize hyperparameters while leaving your testing data pristine) and the remaining half as your test set. We will use this data for the remainder of the problem. The goal of this data is to predict the **log** number of shares a news article will have given the other features.

(a) (**math**) Show that the maximum a posteriori problem for linear regression with a zero-mean Gaussian prior $\mathbb{P}(\mathbf{w}) = \prod_j \mathcal{N}(w_j|0, \tau^2)$ on the weights,

$$\arg\max_{\mathbf{w}} \sum_{i=1}^{N} \log \mathcal{N}(y_i|w_0 + \mathbf{w}^\top \mathbf{x}_i, \sigma^2) + \sum_{j=1}^{D} \log \mathcal{N}(w_j|0, \tau^2)$$

is equivalent to the ridge regression problem

$$\arg\min \frac{1}{N} \sum_{i=1}^{N} (y_i - (w_0 + \mathbf{w}^\top \mathbf{x}_i))^2 + \lambda ||\mathbf{w}||_2^2$$

with $\lambda = \sigma^2/\tau^2$.

(b) (**math**) Find a closed form solution $\mathbf{x}^\star$ to the ridge regression problem:

minimize: $||A\mathbf{x} - \mathbf{b}||_2^2 + ||\Gamma\mathbf{x}||_2^2$.

(c) (**implementation**) Attempt to predict the log shares using ridge regression from the previous problem solution. Make sure you include a bias term and *don't regularize the bias term*. Find the optimal regularization parameter $\lambda$ from the validation set. Plot both $\lambda$ versus the validation RMSE (you should have tried at least 150 parameter

settings randomly chosen between 0.0 and 150.0 because the dataset is small) and $\lambda$ versus $||\boldsymbol{\theta}^\star||_2$ where $\boldsymbol{\theta}$ is your weight vector. What is the final RMSE on the test set with the optimal $\lambda^\star$?

(d) (**math**) Consider regularized linear regression where we pull the basis term out of the feature vectors. That is, instead of computing $\hat{\mathbf{y}} = \boldsymbol{\theta}^\top \mathbf{x}$ with $\mathbf{x}_0 = 1$, we compute $\hat{\mathbf{y}} = \boldsymbol{\theta}^\top \mathbf{x} + b$. This corresponds to solving the optimization problem

$$\text{minimize: } ||A\mathbf{x} + b\mathbf{1} - \mathbf{y}||_2^2 + ||\Gamma\mathbf{x}||_2^2.$$

Solve for the optimal $\mathbf{x}^\star$ explicitly. Use this close form to compute the bias term for the previous problem (with the same regularization strategy). Make sure it is the same.

(e) (**implementation**) We can also compute the solution to the least squares problem using gradient descent. Consider the same bias-relocated objective

$$\text{minimize: } f = ||A\mathbf{x} + b\mathbf{1} - \mathbf{y}||_2^2 + ||\Gamma\mathbf{x}||_2^2.$$

Compute the gradients and run gradient descent. Plot the $\ell_2$ norm between the optimal $(\mathbf{x}^\star, b^\star)$ vector you computed in closed form and the iterates generated by gradient descent. Hint: your plot should move down and to the left and approach zero as the number of iterations increases. If it doesn't, try decreasing the learning rate.

(a) We are given the maximum a posteriori problem $\max_{\mathbf{w}} \mathbb{P}(\mathbf{w}|\mathcal{D}) = \max_{\mathbf{w}} \mathbb{P}(\mathcal{D}|\mathbf{w})\mathbb{P}(\mathbf{w})$ in the form

$$\arg\max_{\mathbf{w}} \sum_{i=1}^{N} \log \mathcal{N}(y_i | w_0 + \mathbf{w}^\top \mathbf{x}_i, \sigma^2) + \sum_{j=1}^{D} \log \mathcal{N}(w_j | 0, \tau^2).$$

Using the Gaussian density we have the equivalent problem

$$\arg\max_{\mathbf{w}} \sum_{i=1}^{N} -\frac{(y_i - w_0 - \mathbf{w}^\top \mathbf{x}_i)^2}{2\sigma^2} - \log\sqrt{2\pi}\sigma + \sum_{i=1}^{D} -\frac{w_j^2}{2\tau^2} - \log\sqrt{2\pi}\sigma.$$

Because the constant $-(N+D)\log\sqrt{2\pi}\sigma$ doesn't change our optimal value, because we can similarly scale our objective by $2\sigma^2$ without changing $\mathbf{w}^\star$, and because maximizing a function is equivalent to minimizing it's negative, we arrive at the equivalent optimization problem

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{N} (y_i - w_0 - \mathbf{w}^\top \mathbf{x}_i)^2 + \sum_{i=1}^{D} \frac{\sigma^2}{\tau^2} w_j^2.$$

Defining $\lambda = \sigma^2/\tau^2$ we have our final equivalent form:

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{N} (y_i - w_0 - \mathbf{w}^\top \mathbf{x}_i)^2 + \lambda ||\mathbf{w}||_2^2.$$

(b) The Ridge Regression problem

$$\text{minimize } f = ||A\mathbf{x} - \mathbf{b}||_2^2 + ||\Gamma\mathbf{x}||^2$$

gives

$$\begin{aligned}
\nabla f &= \nabla \left[ (A\mathbf{x} - \mathbf{b})^\top (A\mathbf{x} - \mathbf{b}) + \mathbf{x}^\top \Gamma^\top \Gamma \mathbf{x} \right] && (1)\\
&= \nabla \left[ \mathbf{x}^\top A^\top A\mathbf{x} - 2\mathbf{x}^\top A^\top \mathbf{b} + \mathbf{b}^\top \mathbf{b} + \mathbf{x}^\top \Gamma^\top \Gamma \mathbf{x} \right] && (2)\\
\nabla f = 0 &= 2A^\top A\mathbf{x} - 2A^\top \mathbf{b} + 2\Gamma^\top \Gamma \mathbf{x} && (3)\\
& && (4)
\end{aligned}$$

so

$$\mathbf{x}^\star = (A^\top A + \Gamma^\top \Gamma)^{-1} A^\top \mathbf{b}$$

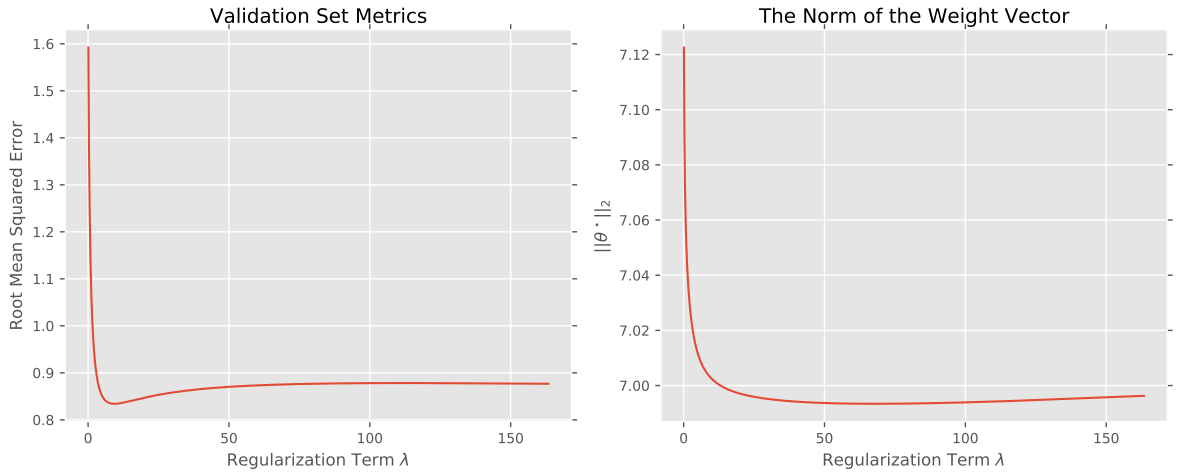
If we let $\Gamma = \sqrt{\lambda}\,\mathbf{I}$ this gives an objective of the form

$$\text{minimize } f = ||A\mathbf{x} - \mathbf{b}||_2^2 + \lambda \mathbf{x}^\top \mathbf{x}$$

and an optimal solution

$$\mathbf{x}^\star = (A^\top A + \lambda \mathbf{I})^{-1} A^\top \mathbf{b}$$

(c) We have an optimal regularization parameter of $\lambda^\star = 9.306$ with a RMSE on the validation set of 0.834 and a test set RMSE of 0.862. Here is the plot:




And here is the code (not including plotting and figuring out the optimal value):

```
df = pd.read_csv('https://math189r.github.io/hw/data/online_news_popularity/'
                 'online_news_popularity.csv',
                 sep=', ', engine='python')
```

```
# do train/test/validation split
df['cohort'] = 'train'
df.iloc[int(0.666*len(df)):int(0.833*len(df)),-1] = 'validation'
df.iloc[int(0.833*len(df)):,-1] = 'test'
df.describe()

X_train, y_train = df[df.cohort == 'train'][
[col for col in df.columns if col not in ['url', 'shares', 'cohort']]
], np.log(df[df.cohort == 'train'].shares).reshape(-1,1)
X_val, y_val = df[df.cohort == 'validation'][
[col for col in df.columns if col not in ['url', 'shares', 'cohort']]
], np.log(df[df.cohort == 'validation'].shares).reshape(-1,1)
X_test, y_test = df[df.cohort == 'test'][
[col for col in df.columns if col not in ['url', 'shares', 'cohort']]
], np.log(df[df.cohort == 'test'].shares).reshape(-1,1)

X_train = np.hstack((np.ones_like(y_train), X_train))
X_val = np.hstack((np.ones_like(y_val), X_val))
X_test = np.hstack((np.ones_like(y_test), X_test))

def linreg(X, y, reg=0.0):
    eye = np.eye(X.shape[1])
    eye[0,0] = 0. # don't regularize bias term!
    return np.linalg.solve(
        X.T @ X + reg * eye,
        X.T @ y
    )

theta_optimal = linreg(X_train, y_train, reg=9.34604374950277)
```

(d) We want to solve

$$\text{minimize: } f = ||A\mathbf{x} + b\mathbf{1} - \mathbf{y}||_2^2 + ||\Gamma\mathbf{x}||_2^2.$$

The objective becomes

$$\begin{aligned} f &= ||A\mathbf{x} + b\mathbf{1} - \mathbf{y}||_2^2 + ||\Gamma\mathbf{x}||_2^2 && (5)\\ &= (A\mathbf{x} + b\mathbf{1} - \mathbf{y})^\top (A\mathbf{x} + b\mathbf{1} - \mathbf{y}) + \mathbf{x}^\top \Gamma^\top \Gamma\mathbf{x} && (6)\\ &= \mathbf{x}^\top A^\top A\mathbf{x} + 2b\mathbf{1}^\top A\mathbf{x} - 2\mathbf{y}^\top A\mathbf{x} - 2b\mathbf{1}^\top \mathbf{y} + b^2 n + \mathbf{y}^\top \mathbf{y} + \mathbf{x}^\top \Gamma^\top \Gamma\mathbf{x} && (7) \end{aligned}$$

At optimality we have $\nabla f = 0$, so

$$\begin{aligned} \nabla_{\mathbf{x}} f &= 2A^\top A\mathbf{x} + 2bA^\top \mathbf{1} - 2A^\top \mathbf{y} + 2\Gamma^\top \Gamma\mathbf{x} = 0, \text{ and} && (8)\\ \nabla_b f &= 2\mathbf{1}^\top A\mathbf{x} - 2\mathbf{1}^\top \mathbf{y} + 2bn = 0. && (9) \end{aligned}$$

Solving for $b^\star$ this gives

$$b^\star = \frac{\mathbf{1}^\top(\mathbf{y} - A\mathbf{x})}{n}$$

This makes sense because if we assume the line is flat ($\mathbf{x} = \mathbf{0}$) the bias term becomes the mean value of the output, as desired.

Plugging this back in to solve for $\mathbf{x}^\star$, we find

$$(A^\top A + \Gamma^\top\Gamma)\mathbf{x} + \frac{\mathbf{1}^\top(\mathbf{y} - A\mathbf{x})}{n}A^\top\mathbf{1} - A^\top\mathbf{y} = 0 \tag{10}$$

$$\left[A^\top A + \Gamma^\top\Gamma - \frac{1}{n}A^\top\mathbf{1}\mathbf{1}^\top A\right]\mathbf{x} = A^\top\mathbf{y} - \frac{1}{n}A^\top\mathbf{1}\mathbf{1}^\top y \tag{11}$$

$$\left[A^\top(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top)A + \Gamma^\top\Gamma\right]\mathbf{x} = A^\top(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top)\mathbf{y} \tag{12}$$

$$\mathbf{x}^\star = \left[A^\top\left(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top\right)A + \Gamma^\top\Gamma\right]^{-1}A^\top\left(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top\right)\mathbf{y}. \tag{13}$$

Note that $\mathbf{I}$ is the Identity matrix and $\mathbf{1}$ is the vector of all ones. $\mathbf{y} \in \mathbf{R}^n$.

```
X_train_no_b = X_train[:,1:]
X_val_no_b = X_val[:,1:]
reg_opt = reg[np.argmin(rmse)]
feats = X_train_no_b.shape[1]
n = X_train_no_b.shape[0]

print('==> Computing A_mod (takes a while to construct 1_{n x n})')
A_mod = X_train_no_b.T @ (np.eye(n) - 1./n)
print('==> Computing optimal theta')
theta_opt = np.linalg.solve(
    A_mod @ X_train_no_b + reg_opt * np.eye(A_mod.shape[0]),
    A_mod @ y_train,
)
print('==> Computing optimal intercept')
b_opt = (y_train - X_train_no_b @ theta_opt).sum() / n

original = linreg(X_train, y_train, reg=reg_opt)

print('==> Distance between intercept and orig: {}'.format(np.abs(original[0] - b_opt
print('==> Distance between theta and original: {}'.format(
    np.linalg.norm(theta_opt - original[1:])
))
# ==> Distance between intercept and orig: 5.697042837482513e-11
# ==> Distance between theta and original: 1.6231311589256314e-10
```

(e) For the optimization problem above,

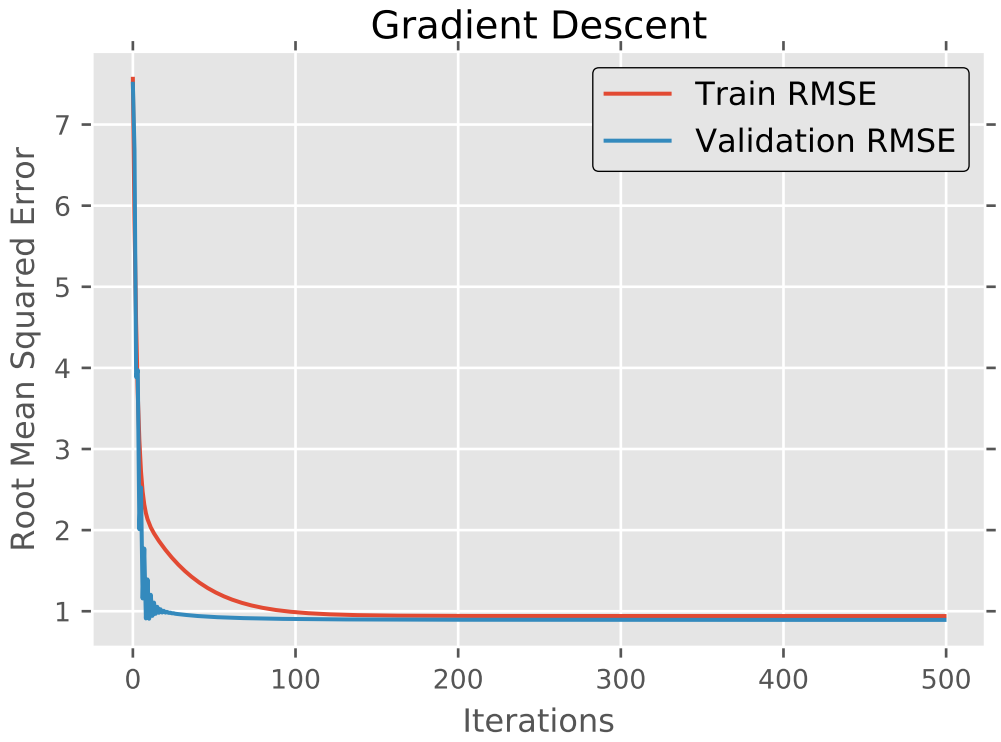
$$\text{minimize: } f = ||A\mathbf{x} + b\mathbf{1} - \mathbf{y}||_2^2 + ||\Gamma\mathbf{x}||_2^2.$$

we have the gradients

$$\nabla_{\mathbf{x}} f = \left(A^\top A + \Gamma^\top \Gamma\right)\mathbf{x} + A^\top (b\mathbf{1} - \mathbf{y}), \text{ and} \tag{14}$$

$$\nabla_b f = \mathbf{1}^\top A\mathbf{x} - \mathbf{1}^\top \mathbf{y} + bn, \tag{15}$$

as shown above. Convergence plot:




Code:

```
shape = (X_train_no_b.shape[1],1)
n = X_train_no_b.shape[0]
eps = 1e-6
max_iters = 150
lr_theta = 2.5e-12
lr_b = 0.2
reg_opt = reg[np.argmin(rmse)]
theta_ = np.zeros(shape)
b_ = 0.

grad_theta = np.ones_like(theta_)
grad_b = np.ones_like(b_)
```

```
objective_train = []
objective_val = []

print('==> Training.')
while np.linalg.norm(grad_theta) > eps and \
     np.abs(grad_b) > eps and \
     len(objective_train) < max_iters:
    objective_train.append(
        np.sqrt(
            np.linalg.norm(
                (X_train_no_b @ theta_).reshape(-1,1) + b_ - y_train,
            )**2 / y_train.shape[0]
        )
    )
    objective_val.append(
        np.sqrt(
            np.linalg.norm(
                (X_val_no_b @ theta_).reshape(-1,1) + b_ - y_val,
            )**2 / y_val.shape[0]
        )
    )

    grad_theta = (
        (X_train_no_b.T @ X_train_no_b + reg_opt * np.eye(shape[0])) @ theta_ +
                 X_train_no_b.T @ (b_ - y_train)
    ) / X_train_no_b.shape[0]
    grad_b = (
        (X_train_no_b @ theta_).sum() - y_train.sum() + b_ * n
    ) / X_train_no_b.shape[0]

    theta_ = theta_ - lr_theta * grad_theta
    b_ = b_ - lr_b * grad_b

    if len(objective_train) % 25 == 0:
        print('-- finishing iteration {} - objective {:5.4f} - grad {}'.format(
             len(objective_train), objective_train[-1], np.linalg.norm(grad_theta)
        ))

print('==> Distance between intercept and orig: {}'.format(np.abs(theta_optimal[0] -
print('==> Distance between theta and original: {}'.format(
    np.linalg.norm(theta_ - theta_optimal[1:])
))
# ==> Distance between intercept and orig: 0.482700615906861
# ==> Distance between theta and original: 0.7936643430206658
```

**2** (**MNIST**) Download the training set at `http://pjreddie.com/media/files/mnist_train.csv` and test set at `http://pjreddie.com/media/files/mnist_test.csv`. This dataset, the MNIST dataset, is a classic in the deep learning literature as a toy dataset to test algorithms on. The problem is this: we have $28 \times 28$ images of handwritten digits as well as the label of which digit $0 \leq$ `label` $\leq 9$ the written digit corresponds to. Given a new image of a handwritten digit, we want to be able to predict which digit it is. The format of the data is `label, pix-11, pix-12, pix-13, ...` where `pix-ij` is the pixel in the `ith` row and `jth` column.

(a) (**logistic**) Restrict the dataset to only the digits with a label of 0 or 1. Implement L2 regularized logistic regression as a model to compute $\mathbb{P}(y = 1|\mathbf{x})$ for a different value of the regularization parameter $\lambda$. Plot the learning curve (objective vs. iteration) when using Newton's Method *and* gradient descent. Plot the accuracy, precision ($p = \mathbb{P}(y = 1|\hat{y} = 1)$), recall ($r = \mathbb{P}(\hat{y} = 1|y = 1)$), and F1-score ($F1 = 2pr/(p + r)$) for different values of $\lambda$ (try at least 10 different values including $\lambda = 0$) on the test set and report the value of $\lambda$ which maximizes the accuracy on the test set. What is your accuracy on the test set for this model? Your accuracy should definitely be over 90%.

(b) (**softmax**) Now we will use the whole dataset and predict the label of each digit using L2 regularized softmax regression (multinomial logistic regression). Implement this using gradient descent, and plot the accuracy on the test set for different values of $\lambda$, the regularization parameter. Report the test accuracy for the optimal value of $\lambda$ as well as it's learning curve. Your accuracy should be over 90%.

(c) (**KNN**) Solve the same problem posed in part (b) but use K-Nearest Neighbors instead of softmax regression and vary $k$ instead of $\lambda$. Only try 3 values for $k$ (1, 5, and 10) and the $\ell_2$ norm as your metric. Plot and report the same results as part (b).

(a) For the logistic model we have $P(y = 1|\mathbf{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^T\mathbf{x})$ and, with a Gaussian prior on the weights we have the augmented log likelihood

$$\ell(\boldsymbol{\theta}) = \sum_i y_i \log \sigma(\boldsymbol{\theta}^T x) + (1 - y_i) \log \left(1 - \sigma(\boldsymbol{\theta}^\top \mathbf{x})\right) + \frac{\lambda}{2}||\boldsymbol{\theta}||_2^2.$$

Taking gradients we find

$$\nabla_{\boldsymbol{\theta}} \ell = \sum_i y_i \left(1 - \sigma(\boldsymbol{\theta}^\top \mathbf{x})\right) \mathbf{x} - (1 - y_i)\sigma(\boldsymbol{\theta}^\top \mathbf{x})\mathbf{x} + \lambda\boldsymbol{\theta} \tag{16}$$

$$= \sum_i \left[y_i - \sigma(\boldsymbol{\theta}^\top \mathbf{x}_i)\right] \mathbf{x}_i + \lambda\boldsymbol{\theta} \tag{17}$$

$$= X^\top (y - \sigma(X\boldsymbol{\theta})) + \lambda\boldsymbol{\theta}. \tag{18}$$

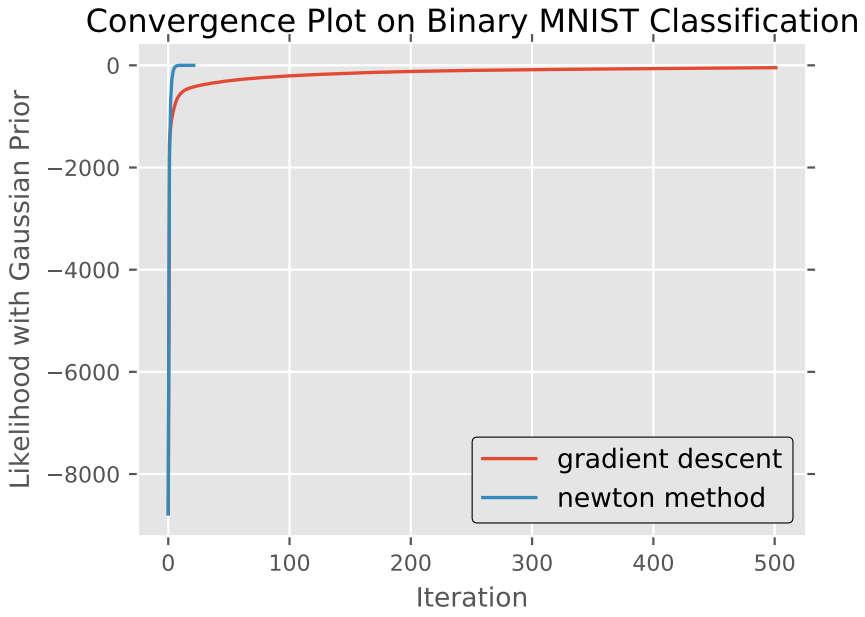
From this we can find a Hessian of

$$\nabla^2 \ell = \frac{d}{d\boldsymbol{\theta}} \nabla \ell^\top \tag{19}$$

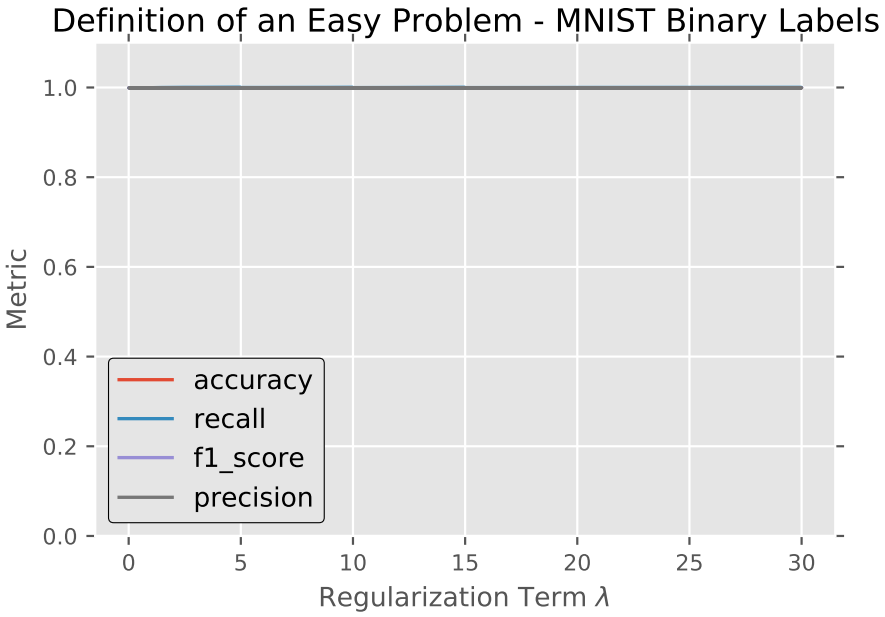
$$= \sum_i \nabla_{\boldsymbol{\theta}} \sigma(\boldsymbol{\theta}^\top \mathbf{x}) \mathbf{x}_i^\top + \lambda I \tag{20}$$

$$= X^T \text{diag}\left[\sigma(X\boldsymbol{\theta})\left(1 - \sigma(X\boldsymbol{\theta})\right)\right] X + \lambda I \tag{21}$$

As it turns out, this problem is intrinsically easy (from a modelling point of view) as ones and zeros are easily told apart. We can see the convergence plot below:



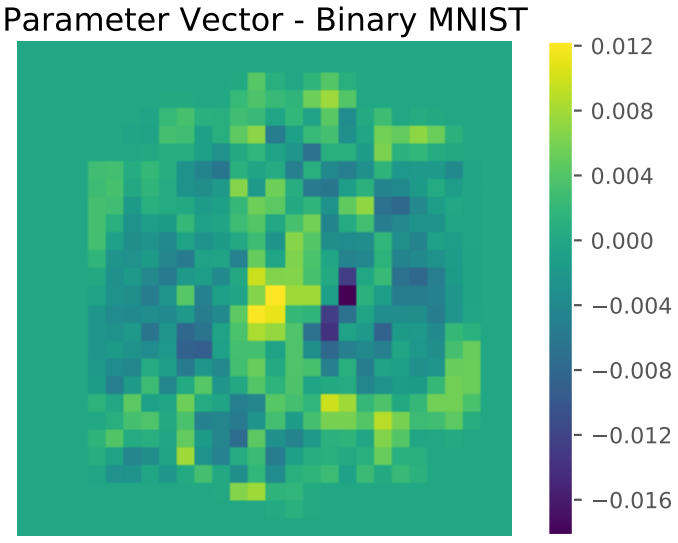

Notice how Newton's Method is *much, much* faster than raw gradient descent by orders of magnitude. As was (or will be depending on when you read this) discussed in class/review, this speedup stems from the scale invariance of the algorithm. As was (or will be depending on when you read this) discussed in class/review, this speedup stems from the scale invariance of the algorithm. We can tell this problem is easy by looking at the plots of test metrics for different regularization parameters:




Indeed, all of them are perfect for any reasonable regularization parameter. This is what is known as a *easy problem*. If we even try out linear regression on this classification problem (mapping binary labels to $\{-1, +1\}$) we have a relatively solid accuracy with an even simpler method:

```
theta = linreg(X_bin_train, y_bin_train*2 - 1, reg=1e-2)
y_pred = X_bin_train @ theta > 0
accuracy(y_bin_test, y_pred) # 0.994788
```

Another interesting insight comes from looking at the optimal logistic regression parameter vector:



We can see that having a pixel turned on right at the center of the image will heavily influence the probability of having a digit of 1 (ignoring correlations between parameters themselves). Similarly, we can observe a circle representing the digit zero which is darker and centered around the center of the image.

Here is the code:

```
import numpy as np
from scipy import sparse
# assumes data loaded into X_bin_* and y_bin_*

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def log_likelihood(X, y_bool, theta, reg=1e-6):
    mu = sigmoid(X @ theta)
    mu[~y_bool] = 1 - mu[~y_bool]
    return np.log(mu).sum() - reg*np.inner(theta, theta)/2

def grad_log_likelihood(X, y, theta, reg=1e-6):
    return X.T @ (sigmoid(X @ theta) - y) + reg * theta

def newton_step(X, y, theta, reg=1e-6):
    mu = sigmoid(X @ theta)
    # using a cholesky solve is exactly twice as fast as a regular np.linalg.solve.
    # also, using scipy.sparse.diags will be much more efficient than constructing
    # the entire diagonal scaling matrix. Same with sparse.eye.
    return linalg.cho_solve(
        linalg.cho_factor(
            X.T @ sparse.diags(mu * (1 - mu)) @ X + reg * sparse.eye(X.shape[1]),
        ),
        grad_log_likelihood(X, y, theta, reg=reg),
    )

def lr_grad(
        X, y,
        reg=1e-6, lr=1e-3, tol=1e-6,
        max_iters=300, verbose=False,
        print_freq=5,
    ):
    y = y.astype(bool)
    theta = np.zeros(X.shape[1])
    objective = [log_likelihood(X, y, theta, reg=reg)]
    grad = grad_log_likelihood(X, y, theta, reg=reg)
```

```
    while len(objective)-1 <= max_iters and \
          np.linalg.norm(grad) > tol:
        if verbose and (len(objective)-1) % print_freq == 0:
            print('[i={}] likelihood: {}. grad norm: {}'.format(
                len(objective)-1, objective[-1], np.linalg.norm(grad),
            ))

        grad = grad_log_likelihood(X, y, theta, reg=reg)
        theta = theta - lr * grad
        objective.append(log_likelihood(X, y, theta, reg=reg))

    if verbose:
        print('[i={}] done. grad norm = {:0.2f}'.format(
            len(objective)-1, np.linalg.norm(grad)
        ))
    return theta, objective

def lr_newton(
        X, y,
        reg=1e-6, tol=1e-6, max_iters=300,
        verbose=False, print_freq=5,
    ):
    y = y.astype(bool)
    theta = np.zeros(X.shape[1])
    objective = [log_likelihood(X, y, theta, reg=reg)]
    step = newton_step(X, y, theta, reg=reg)

    while len(objective)-1 <= max_iters and \
          np.linalg.norm(step) > tol:
        if verbose and (len(objective)-1) % print_freq == 0:
            print('[i={}] likelihood: {}. step norm: {}'.format(
                len(objective)-1, objective[-1], np.linalg.norm(step)
            ))

        step = newton_step(X, y, theta, reg=reg)
        theta = theta - step
        objective.append(log_likelihood(X, y, theta, reg=reg))

    if verbose:
        print('[i={}] done. step norm = {:0.2f}'.format(
            len(objective)-1, np.linalg.norm(step)
        ))
    return theta, objective
```

(b) For softmax regression we have $\mathbb{P}(y = c|\mathbf{x}, W) = \frac{1}{Z}\exp(\mathbf{w}_c^\top \mathbf{x}) = \frac{\exp(\mathbf{w}_c^\top \mathbf{x})}{\sum_i \exp(\mathbf{w}_i^\top \mathbf{x})}$. Assuming a Gaussian prior on each column of $W$ we have a log likelihood

$$\ell(W) = \log \prod_i \prod_c \mu_{ic}^{y_{ic}} - \lambda \mathrm{tr}(W^\top W) \tag{22}$$

$$= \sum_i \sum_c y_{ic} \log \mu_{ic} - \lambda \mathrm{tr}(W^\top W) \tag{23}$$

$$= \sum_i \left[ \left( \sum_c y_{ic} \mathbf{w}_c^\top \mathbf{x}_i \right) - \log \left( \sum_c \exp(\mathbf{w}_c^\top \mathbf{x}_i) \right) \right] + \lambda \mathrm{tr}(W^\top W). \tag{24}$$

This is also known as the negative cross entropy loss ($f(W) = -\ell(W)$) as discussed in Murphy page 255. We can find

$$\nabla_{\mathbf{w}_c} \ell = \sum_i (\boldsymbol{\mu}_i - \mathbf{y}_i) \otimes \mathbf{x}_i - \lambda W \tag{25}$$

where $\otimes$ is the Kronecker product. If we want a closed form and cleaner version to use with a matrix library, we can see that

$$\nabla_W \ell = X^\top (\boldsymbol{\mu} - \mathbf{y}) \tag{26}$$

where $\mathbf{y} \in \{0, 1\}^{n \times c}$ is the "one-hot encoding" of the output $\mathbf{y}$ such that

$$\mathbf{y}_{ij} = \begin{cases} 1 & \text{if datum } i \text{ is digit } j \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

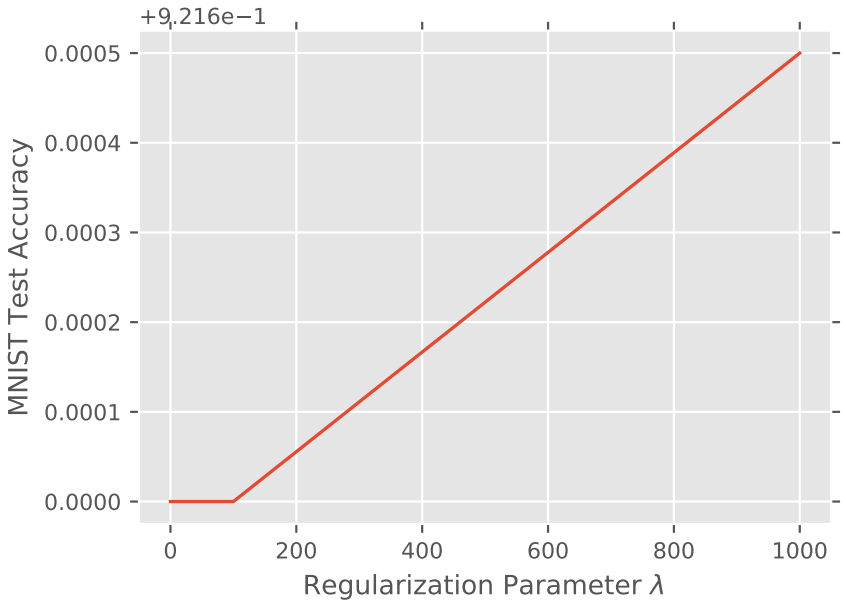
and

$$\mathbf{y}\mathbf{1}_c = \mathbf{1}_n. \tag{28}$$

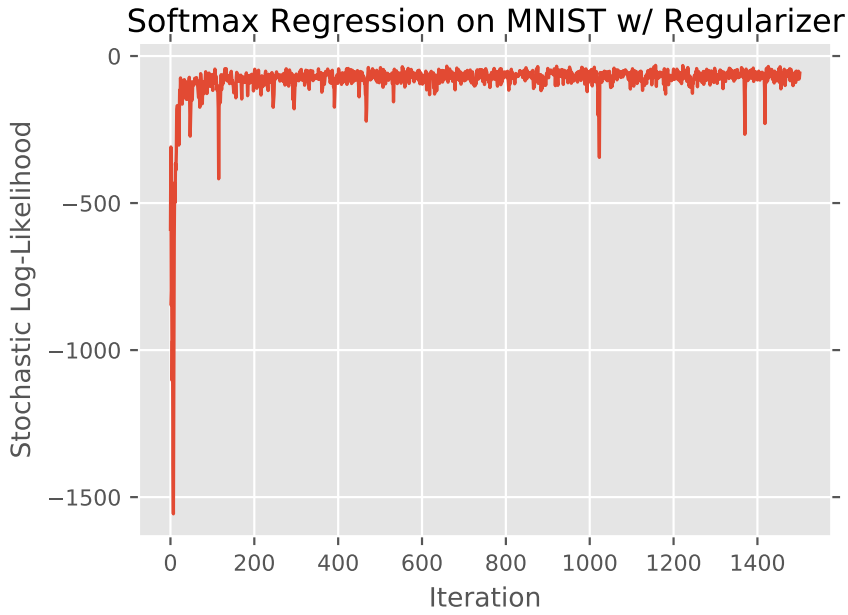
Similarly, we define $\boldsymbol{\mu} \in [0, 1]^{n \times c}$ as

$$\boldsymbol{\mu}_i = \mathcal{S}(\mathbf{x}_i) = \frac{\exp(W^\top \mathbf{x})}{\mathbf{1}^\top \exp(W^\top \mathbf{x})} \tag{29}$$

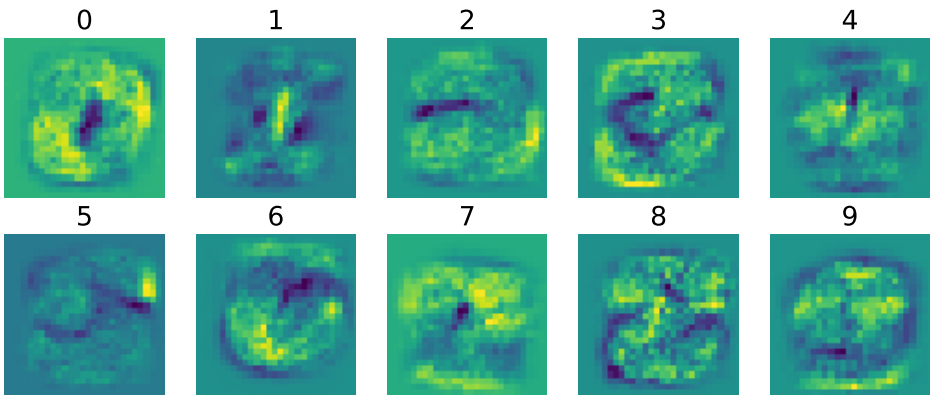
and exp is applied elementwise. Using stochastic gradient descent, we have the test accuracies over different regularization parameters as follows:




The scale is confusing, but the maximum test accuracy was 92.21% with $\lambda = 1000$. This corresponds to the For the optimal regularizer we also have the following convergence plot:




Also nice is the plot of weights for each number. Note how the model distinguishes between the different digits (dark is negative and light is positive):




And here is the code:

```
import numpy as np
from sklearn.preprocessing import OneHotEncoder

def softmax(x):
    s = np.exp(x - np.max(x, axis=1))
    return s / np.sum(s, axis=1)

def log_softmax(x):
    return x - logsumexp(x, axis=1)

def log_likelihood(X, y_one_hot, W, reg=1e-6):
    mu = X @ W
    # einsum term computes trace(W.T @ W) efficiently
    return np.sum(
        mu[y_one_hot] - logsumexp(mu, axis=1),
    ) - reg*np.einsum('ij,ji->', W.T, W)/2

def grad_log_likelihood(X, y_one_hot, W, reg=1e-6):
    mu = X @ W # n by c matrix
    mu = np.exp(mu - np.max(mu, axis=1)[:,np.newaxis])
    mu = mu / np.sum(mu, axis=1)[:,np.newaxis]
    return X.T @ (mu - y_one_hot) + reg*W

def softmax_grad(
    X, y, reg=1e-6, lr=1e-3, tol=1e-6,
    max_iters=300, batch_size=256,
    verbose=False, print_freq=5,
):
    enc = OneHotEncoder()
    y_one_hot = enc.fit_transform(
        y.copy().reshape(-1,1),
```

```
    ).astype(bool).toarray()

    W = np.zeros((X.shape[1], y_one_hot.shape[1]))

    ind = np.random.randint(0, X.shape[0], size=batch_size)
    objective = [log_likelihood(X[ind], y_one_hot[ind], W, reg=reg)]
    grad = grad_log_likelihood(X[ind], y_one_hot[ind], W, reg=reg)

    while len(objective)-1 <= max_iters and \
          np.linalg.norm(grad) > tol:

        if verbose and (len(objective)-1) % print_freq == 0:
            print('[i={}] likelihood: {}. grad norm: {}'.format(
                   len(objective)-1, objective[-1], np.linalg.norm(grad)
            ))

        ind = np.random.randint(0, X.shape[0], size=batch_size)
        grad = grad_log_likelihood(X[ind], y_one_hot[ind], W, reg=reg)
        W = W - lr * grad
        objective.append(log_likelihood(X[ind], y_one_hot[ind], W, reg=reg))

    if verbose:
        print('[i={}] done. grad norm = {:0.2f}'.format(
            len(objective)-1, np.linalg.norm(grad)
        ))
    return W, objective
```

(c) Code is below. Because this would have taken approximately forever to run we just downsampled the dataset to 2500 datapoints. It still takes a while, but is bearable. Thus we arrive at an approximate lower bound on accuracy of 91.31% from the stochastic dataset. This is an example of when nonparametric methods such as K-Nearest-Neighbors might be too slow for a relatively larger dataset. That said, this downsampled method still performs relatively similar to the softmax regression and is much simpler to implement, so that would be a plus.

```
def predict_knn(X_test, X_train, y_train, k=5, verbose=False, print_freq=1000):
    y_pred = np.zeros(X_test.shape[0])
    for i in range(X_test.shape[0]):
        if verbose and i % print_freq == 0:
            print('[i={}] done.'.format(i))
        img = X_test[i]
        ind = np.argpartition(
            1./np.linalg.norm(X_train - img[:,np.newaxis].T, axis=1),
            -k,
        )[-k:]
```

```
        y_pred[i] = np.argmax(np.bincount(y_train[ind]))
    return y_pred

# downsampling the training set to 2500 data points we get the following
# [k=1] accuracy: 0.9131
# [k=5] accuracy: 0.9113
# [k=10] accuracy: 0.9034
for k in [1,5,10]:
    print('[k={}] accuracy: {}'.format(
            k,
            accuracy(y_test, predict_knn(
                    X_test, X_train, y_train, k=k,
                    verbose=True, print_freq=1000,
            )),
    ))
```

---

**3** (**Murphy 2.11** and **2.16**)

(a) Derive the normalization constant ($Z$) for a one dimensional zero-mean Gaussian

$$\mathbb{P}(x;\sigma^2) = \frac{1}{Z}\exp\left(-\frac{x^2}{2\sigma^2}\right)$$

such that $\mathbb{P}(x;\sigma^2)$ becomes a valid density.

(b) Suppose $\theta \sim \text{Beta}(a,b)$ such that

$$\mathbb{P}(\theta;a,b) = \frac{1}{B(a,b)}\theta^{a-1}(1-\theta)^{b-1} = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^{a-1}(1-\theta)^{b-1}$$

where $B(a,b) = \Gamma(a)\Gamma(b)/\Gamma(a+b)$ is the Beta function and $\Gamma(x)$ is the Gamma function. Derive the mean, mode, and variance of $\theta$.

(a) As any probability density must integrate to one, we know

$$\int_{\mathbb{R}} \mathbb{P}(x;\sigma^2)\,dx = \int_{\mathbb{R}} \frac{1}{Z}\exp\left(-\frac{x^2}{2\sigma^2}\right)dx = 1, \tag{30}$$

or

$$Z = \int_{\mathbb{R}} \exp\left(-\frac{x^2}{2\sigma^2}\right)dx. \tag{31}$$

Now consider

$$Z^2 = \int_{\mathbb{R}} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx \int_{\mathbb{R}} \exp\left(-\frac{y^2}{2\sigma^2}\right) dy \tag{32}$$

$$= \iint_{\mathbb{R}^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \, dx \, dy \tag{33}$$

$$= \int_0^{\infty} \int_0^{2\pi} \exp\left(-\frac{r^2}{2\sigma^2}\right) r \, d\theta \, dr \tag{34}$$

$$= 2\pi \int_0^{\infty} \exp\left(-\frac{r^2}{2\sigma^2}\right) r \, dr \tag{35}$$

$$= 2\pi\sigma^2 \exp\left(-\frac{r^2}{2\sigma^2}\right)_{\infty}^{0} \tag{36}$$

$$= 2\pi\sigma^2. \tag{37}$$

Hence we have

$$Z = \sqrt{2\pi\sigma^2} = \sqrt{2\pi}\sigma,$$

as desired.

(b) Recall

$$\Gamma(x+1) = x\Gamma(x) \tag{38}$$

$$B(a,b) = \int_0^1 \theta^{a-1}(1-\theta)^{b-1} d\theta = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}. \tag{39}$$

Then we can compute the mean as

$$\mathbb{E}[\theta] = \frac{1}{B(a,b)} \int_0^1 \theta\theta^{a-1}(1-\theta)^{b-1} d\theta \tag{40}$$

$$= \frac{1}{B(a,b)} \int_0^1 \theta^{a}(1-\theta)^{b-1} d\theta \tag{41}$$

$$= \frac{B(a+1,b)}{B(a,b)} \tag{42}$$

$$= \frac{\Gamma(a+1)\Gamma(b)}{\Gamma(a+b+1)} \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \tag{43}$$

$$= \frac{a\Gamma(a)\Gamma(b)}{(a+b)\Gamma(a+b)} \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \tag{44}$$

$$= \frac{a}{a+b} \tag{45}$$

as desired. To compute the variance $\mathbb{V}[\theta] = \mathbb{E}[\theta^2] - \mathbb{E}[\theta]^2$ we can use the same technique to find

$$\mathbb{E}[\theta^2] = \frac{a(a+1)}{(a+b+1)(a+b)}. \tag{46}$$

Thus we have

$$\mathbb{V}[\theta] = \frac{a(a+1)}{(a+b+1)(a+b)} - \frac{a^2}{(a+b)^2} \tag{47}$$

$$= \frac{a(a+1)(a+b) - a^2(a+b+1)}{(a+b+1)(a+b)^2} \tag{48}$$

$$= \frac{ab}{(a+b+1)(a+b)^2}. \tag{49}$$

To compute the mode, we want to find when $\nabla_\theta \mathbb{P}(\theta; a, b) = 0$ on the interval $[0, 1]$. Because a constant term won't change the optimizing value, we can work with the unnormalized distribution (ignoring the $B(a, b)$ term). This gives

$$\nabla_\theta \mathbb{P}(\theta; a, b) = \nabla_\theta \theta^{a-1}(1-\theta)^{b-1} \tag{50}$$

$$= (a-1)\theta^{a-2}(1-\theta)^{b-1} - (b-1)\theta^{a-1}(1-\theta)^{b-2} \tag{51}$$

$$= \theta^{a-2}(1-\theta)^{b-2}\left((1-\theta)(a-1) - \theta(b-1)\right) \tag{52}$$

$$= \theta^{a-2}(1-\theta)^{b-2}\left(a - 1 - \theta(a+b-2)\right) = 0. \tag{53}$$

Therefore the right term must equal zero, or the mode

$$\theta^\star = \frac{a-1}{a+b-2}. \tag{54}$$

---

**4** (**Murphy 2.15**) Let $\mathbb{P}_{emp}(x)$ be the empirical distribution and let $q(x|\boldsymbol{\theta})$ be some model. Show that $\arg\min_q \mathbb{KL}(\mathbb{P}_{emp}||q)$ is obtained by $q(x) = q(x; \hat{\boldsymbol{\theta}})$ where $\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(q, \mathcal{D})$ is the maximum likelihood estimate.

Let $\mathbb{P}_{emp}(x)$ be the empirical distribution for the discrete data $D = \{x_1, x_2, \ldots, x_n\}$. Consider

$$\mathbb{KL}(\mathbb{P}_{emp} \| q) = \int_{\mathcal{S}} \mathbb{P}_{emp}(x) \log \frac{\mathbb{P}_{emp}(x)}{q(x;\theta)} dx, \tag{55}$$

which can be written as

$$\int_{\mathcal{S}} \mathbb{P}_{emp}(x) \log \mathbb{P}_{emp}(x) dx - \int_{\mathcal{S}} \mathbb{P}_{emp}(x) \log q(x;\theta) dx. \tag{56}$$

From equation 2.40 in Murphy, the empirical density can be written as

$$\mathbb{P}_{emp}(x) = \frac{1}{n} \sum_{i=1}^{n} \delta(x - x_i) \tag{57}$$

such that

$$\mathbb{KL}(\mathbb{P}_{emp} \| q) = \int_{\mathcal{S}} \mathbb{P}_{emp}(x) \log \mathbb{P}_{emp}(x)\ dx - \int_{\mathcal{S}} \frac{1}{n} \sum_{i=1}^{n} \delta(x - x_i) \log q(x; \theta)\ dx. \tag{58}$$

Because $\int f(x)\delta(x-t)\ dt = f(t)$ this becomes

$$\int_{\mathcal{S}} \mathbb{P}_{emp}(x) \log \mathbb{P}_{emp}(x) dx - \frac{1}{n} \sum_{i=1}^{n} \log(q(x_i; \theta)). \tag{59}$$

To pick $\theta$ to minimize the above value, we only have to consider the term

$$\arg\min_{\theta} - \sum_{i=1}^{n} \log(q(x_i; \theta)) = \arg\max_{\theta} \sum_{i=1}^{n} \log(q(x_i; \theta) \tag{60}$$

because all other therms don't depend on the parameter $\theta$ of $q$. We can see that the term to maximize is precisely the log likelihood of the probability distribution $q$ on data $D$, as desired.

---

**5 (Linear Transformation)** Let $\mathbf{y} = A\mathbf{x} + \mathbf{b}$ be a random vector. show that expectation is linear:

$$\mathbb{E}[\mathbf{y}] = \mathbb{E}[A\mathbf{x} + \mathbf{b}] = A\mathbb{E}[\mathbf{x}] + \mathbf{b}.$$

Also show that

$$\text{cov}[\mathbf{y}] = \text{cov}[A\mathbf{x} + \mathbf{b}] = A\text{cov}[\mathbf{x}]A^\top = A\mathbf{\Sigma}A^\top.$$

(a) We can see that

$$\begin{aligned}
\mathbb{E}[\mathbf{y}] &= \int_{\mathcal{S}} (A\mathbf{x} + \mathbf{b})\, \mathbb{P}(x)\ dx && (61)\\
&= A \int_{\mathcal{S}} \mathbf{x}\, \mathbb{P}(\mathbf{x})\ dx + \mathbf{b} \int_{\mathcal{S}} \mathbb{P}(x)\ dx && (62)\\
&= A\mathbb{E}[\mathbf{x}] + \mathbf{b}, && (63)
\end{aligned}$$

as desired.

(b) We start with the definition of covariance $\text{cov}[\mathbf{x}] = \mathbf{\Sigma} = \mathbb{E}[(\mathbf{x} - \mathbb{E}[x])(\mathbf{x} - \mathbb{E}[x])^\top]$. We can see

$$\begin{aligned}
\text{cov}[\mathbf{y}] &= \text{cov}[A\mathbf{x} + \mathbf{b}] && (64)\\
&= \mathbb{E}[(A\mathbf{x} + \mathbf{b} - \mathbb{E}[A\mathbf{x} + \mathbf{b}])(A\mathbf{x} + \mathbf{b} - \mathbb{E}[A\mathbf{x} + \mathbf{b}])^\top] && (65)\\
&= \mathbb{E}[(A\mathbf{x} + \mathbf{b} - A\mathbb{E}[\mathbf{x}] - \mathbf{b})(A\mathbf{x} + \mathbf{b} - A\mathbb{E}[\mathbf{x}] - \mathbf{b})^\top] && (66)\\
&= \mathbb{E}[A(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^\top A^\top] && (67)\\
&= A\mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^\top]A^\top && (68)\\
&= A\text{cov}[\mathbf{x}]A^\top, && (69)
\end{aligned}$$

as desired.

**6** Given the dataset $\mathcal{D} = \{(x, y)\} = \{(0,1), (2,3), (3,6), (4,8)\}$

(a) Find the least squares estimate $y = \boldsymbol{\theta}^\top \mathbf{x}$ by hand using Cramer's Rule.

(b) Use the normal equations to find the same solution and verify it is the same as part (a).

(c) Plot the data and the optimal linear fit you found.

(d) Find randomly generate 100 points near the line with white Gaussian noise and then compute the least squares estimate (using a computer). Verify that this new line is close to the original and plot the new dataset, the old line, and the new line.

(a) Let the matrix

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}. \tag{70}$$

We then have

$$X^\top X = \begin{bmatrix} 4 & 9 \\ 9 & 29 \end{bmatrix}. \tag{71}$$

Similarly we have

$$X^\top \mathbf{y} = X^\top \begin{bmatrix} 1 \\ 3 \\ 6 \\ 8 \end{bmatrix} = \begin{bmatrix} 18 \\ 56 \end{bmatrix}. \tag{72}$$

By the normal equations we know that $X^\top X \boldsymbol{\theta}^\star = X^\top \mathbf{y}$. Using Cramer's Rule we see that

$$\boldsymbol{\theta}_0^\star = \frac{\begin{vmatrix} 18 & 9 \\ 56 & 29 \end{vmatrix}}{\begin{vmatrix} 4 & 9 \\ 9 & 29 \end{vmatrix}} = \frac{18}{35} \tag{73}$$

$$\boldsymbol{\theta}_1^\star = \frac{\begin{vmatrix} 4 & 18 \\ 9 & 56 \end{vmatrix}}{\begin{vmatrix} 4 & 9 \\ 9 & 29 \end{vmatrix}} = \frac{62}{35} \tag{74}$$

$$\tag{75}$$

with our line

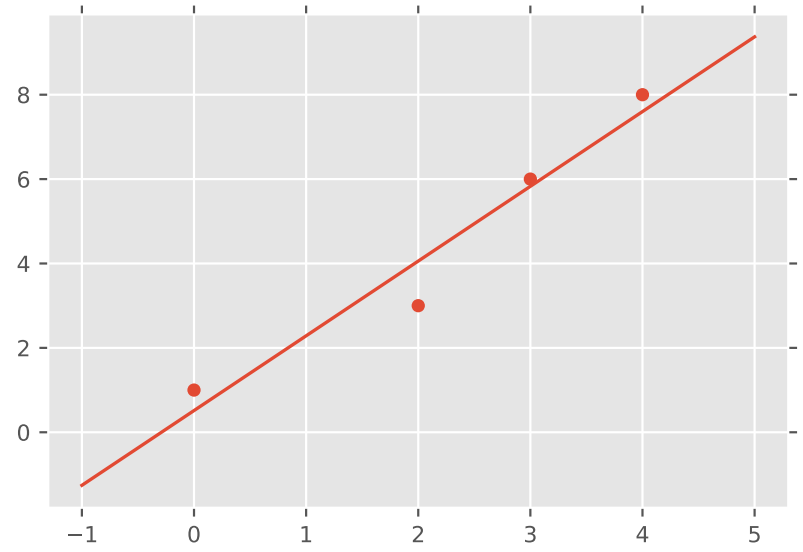
$$\hat{y} = \theta_0 + \theta_1 x. \tag{76}$$

(b) On the computer we have by the normal equation

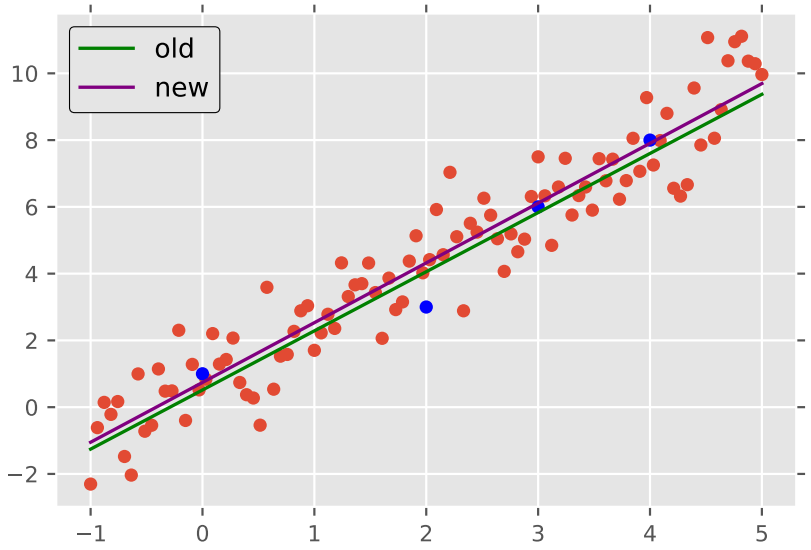
$$\boldsymbol{\theta} = (X^\top X)^{-1} X^\top \mathbf{y} = \begin{bmatrix} 0.514 \\ 1.771 \end{bmatrix}, \tag{77}$$

as desired.

(c) Plot is below:




(d) Plot is below. New line is close.



**7** (**Murphy 8.3**) Gradient and Hessian of the log-likelihood for logistic regression.

(a) Let $\sigma(x) = \frac{1}{1+e^{-x}}$ be the sigmoid function. Show that

$$\sigma'(x) = \sigma(x)\left[1 - \sigma(x)\right].$$

(b) Using the previous result and the chain rule of calculus, derive an expression for the gradient of the log likelihood for logistic regression.

(c) The Hessian can be written as $\mathbf{H} = \mathbf{X}^\top \mathbf{S}\mathbf{X}$ where $\mathbf{S} = \text{diag}(\mu_1(1-\mu_1), \ldots, \mu_n(1-\mu_n))$. Derive this and show that $\mathbf{H} \succeq 0$ ($A \succeq 0$ means that $A$ is positive semidefinite).

(a) We can see that

$$\begin{aligned}
\sigma'(x) &= \nabla \left(1 + e^{-x}\right)^{-1} && (78)\\
&= e^{-x}\left(1 + e^{-x}\right)^{-2} && (79)\\
&= \frac{1}{1+e^{-x}} \frac{1 + e^{-x} - 1}{1 + e^{-x}} && (80)\\
&= \sigma(x)\left(1 - \sigma(x)\right), && (81)
\end{aligned}$$

as desired.

(b) We have

$$\ell(\boldsymbol{\theta}) = \sum_i y_i \log \sigma(\boldsymbol{\theta}^\top \mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\boldsymbol{\theta}^\top \mathbf{x}_i)).$$

Taking gradients, we find that

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) &= \sum_i y_i \left(1 - \sigma(\boldsymbol{\theta}^\top \mathbf{x})\right) \mathbf{x}_i + (1 - y_i)\sigma(\boldsymbol{\theta}^\top \mathbf{x})\mathbf{x}_i && (82)\\
&= \sum_i \left(y_i - \sigma(\boldsymbol{\theta}^\top \mathbf{x}_i)\right) \mathbf{x}_i && (83)\\
&= X^\top(\mathbf{y} - \boldsymbol{\mu}) && (84)
\end{aligned}$$

where $\boldsymbol{\mu} = \sigma(X\boldsymbol{\theta})$.

(c) For the Hessian we have

$$\begin{aligned}
-\nabla^2 \ell(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \left[X^\top \boldsymbol{\mu} - X\right]^\top && (85)\\
&= \nabla \boldsymbol{\mu}^\top X = \nabla \sigma(X\boldsymbol{\theta})^\top X && (86)\\
&= \left(\text{diag}(\boldsymbol{\mu}')X\right)^\top X && (87)\\
&= X^\top \text{diag}\left(\boldsymbol{\mu}(1 - \boldsymbol{\mu})\right) X && (88)
\end{aligned}$$

by the chain rule.

(d) To show $\nabla^2 \ell(\boldsymbol{\theta})$ is trivial. It is a simple excercise to show that saying $-\nabla^2 \ell(\boldsymbol{\theta})$ is positive semi-definite is equivalent to showing diag $(\boldsymbol{\mu}(1-\boldsymbol{\mu}))$ is positive definite. Because the eigenvalues of a diagonal matrix are just the diagonal elements, we simply need to show that

$$\boldsymbol{\mu}_i \left(1-\boldsymbol{\mu}_i\right) = \sigma(\boldsymbol{\theta}^\top \mathbf{x}_i)\left(1-\sigma(\theta^\top \mathbf{x}_i)\right) \geq 0. \tag{89}$$

We know $0 < \sigma(\cdot) < 1$, and $\eta(1-\eta) \geq 0$ only when $0 \leq \eta \leq 1$, so we are done and the hessian of the log likelihood is negative definite (the negative log likelihood hessian is positive definite). Note that this implies our problem is convex.

---

**8** (**Murphy 9**) Show that the multinomial distribution

$$\text{Cat}(x|\boldsymbol{\mu}) = \prod_{k=1}^{K} \mu_k^{x_k}$$

is in the exponential family and show that the generalized linear model corresponding to this distribution is the same as multinomial logistic regression.

To show that the multinomial distribution is in the exponential family, we simply need to rewrite the distribution to include an exponential and logarithm:

$$\prod_{i=1}^{K} \mu_k^{x_k} \;=\; \exp\left(\log\left(\prod_{i=1}^{K} \mu_i^{x_i}\right)\right).$$

The next step is to use the fact $\log(ab) = \log(a) + \log(b)$ and apply it $K$-many times to the product in the argument to obtain

$$\exp\left(\log\left(\prod_{i=1}^{K} \mu_i^{x_i}\right)\right) \;=\; \exp\left(\sum_{k=1}^{K} \log\left(\mu_i^{x_i}\right)\right).$$

Next we use the rule $\log(a^b) = b\log(a)$ to shift the exponents down and obtain

$$\exp\left(\sum_{i=1}^{K} \log\left(\mu_i^{x_i}\right)\right) \;=\; \exp\left(\sum_{i=1}^{K} x_i \log(\mu_i)\right).$$

Now observe that since $\sum_{i=1}^{K} \mu_i = 1$, we then need only specify the first $K-1$ of these terms, since the term$\mu_K$ will automatically be determined at the end. We can therefore split our summation up into

$$
\begin{aligned}
\exp\left(\sum_{i=1}^{K} x_i \log(\mu_i)\right) &= \exp\left(\sum_{i=1}^{K-1} x_i \log(\mu_i) + x_K \log(\mu_K)\right) \\
&= \exp\left[\sum_{i=1}^{K-1} x_i \log(\mu_i) + \left(1 - \sum_{i=1}^{K-1} x_i\right) \log\left(1 - \sum_{i=1}^{K-1} \mu_i\right)\right] \\
&= \exp\left[\sum_{i=1}^{K-1} x_i \left(\log(\mu_i) - \log\left(1 - \sum_{i=1}^{K-1} \mu_i\right)\right) + \log\left(1 - \sum_{i=1}^{K-1} \mu_i\right)\right] \\
&= \exp\left[\sum_{i=1}^{K-1} x_i \log\left(\frac{\mu_i}{\mu_K}\right) + \log(\mu_K)\right]
\end{aligned}
$$

where we use the substitution $\mu_K = 1 - \sum_{i=1}^{K-1} \mu_i$. Therefore taking the vector

$$
\boldsymbol{\theta} = \begin{bmatrix} \log\left(\frac{\mu_1}{\mu_K}\right) \\ \vdots \\ \log\left(\frac{\mu_{K-1}}{\mu_K}\right) \end{bmatrix}
$$

We can therefore make the substitution that $\mu_i = \mu_K e^{\boldsymbol{\theta}_i}$ and

$$
\mu_K = 1 - \mu_K \sum_{i=1}^{K-1} e^{\boldsymbol{\theta}_i}
$$

implying that

$$
\mu_K = \frac{1}{1 + \sum_{i=1}^{K-1} e^{\boldsymbol{\theta}_i}}
$$

hence

$$
\mu_i = \frac{e^{\boldsymbol{\theta}_i}}{1 + \sum_{i=1}^{K-1} e^{\boldsymbol{\theta}_i}}.
$$

Writing the distribution as $Cat(\mathbf{x}|\mu) = \exp(\boldsymbol{\theta}^\top \mathbf{x} - A(\boldsymbol{\theta}))$ we conclude that

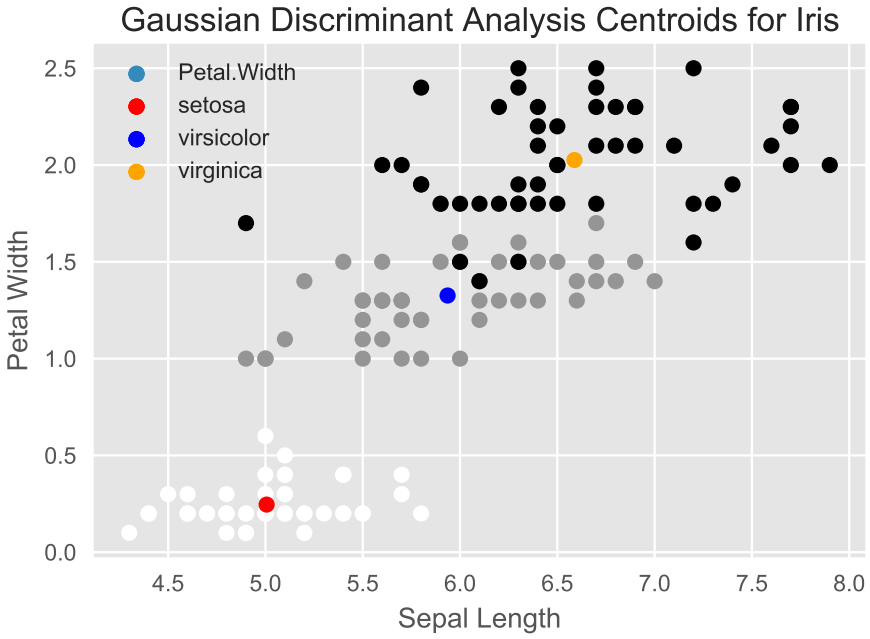
$$A(\boldsymbol{\theta}) = -\log(\mu_K) \quad = \quad \log\left(1 + \boldsymbol{\theta}^T \mathbf{1}\right)$$

where the boldface **1** is the vector of all ones. This example is written out on page 283 of Murphy, as well.

---

**9** Download the Iris dataset from `https://vincentarelbundock.github.io/Rdatasets/csv/datasets/iris.csv` (you can read about the history behind this dataset at `https://en.wikipedia.org/wiki/Iris_flower_data_set`). Our goal is to predict the subspecies of the Iris flower given the sepal length and petal width using Gaussian Discriminant Analysis (Murphy 4.2). Plot the dataset (with different colors for different classes) along with the mean parameters for regular (unlinked/nonlinear) Gaussian Discriminant Analysis. Report the accuracy on the entire dataset for running {linear discriminant analysis with a bunch of different parameters of the regularization parameter $\lambda$, the nonlinear discriminant analysis you plotted above}.

You should read all of 4.2 from Murphy for this problem. Code and plot are below.




```
def discriminant_analysis(X, y, linear=False, reg=0.0):
    labels = np.unique(y)
    mu = {}
    cov = {}
    pi = {}
    for label in labels:
```

```
        pi[label] = (y == label).mean()
        mu[label] = X[y == label].mean(axis=0)
        diff = X[y == label] - mu[label]
        cov[label] = diff.T @ diff / (y == label).sum()

    if linear:
        # tie covariance matrices
        cov = sum((y == label).sum() * cov[label] for label in labels)
        cov = cov / y.shape[0]
        cov = reg*np.diag(np.diag(cov)) + (1-reg)*cov

    return pi, mu, cov

def normal_density(X, mu, cov):
    # predict class probability
    diff = X - mu
    return np.exp(-diff.T @ np.linalg.inv(cov) @ diff / 2) / (
        (2 * np.pi)**(-X.shape[0]/2) * np.sqrt(np.linalg.det(cov))
    )

def predict_proba(X, pi, mu, cov):
    prob = np.zeros((X.shape[0], len(pi)))
    if type(cov) is not dict:
        covariance = cov
        cov = defaultdict(lambda: covariance)
    for i, x in enumerate(X):
        for j in range(len(pi)):
            prob[i,j] = pi[j] * normal_density(x, mu[j], cov[j])

    prob = prob / prob.sum(axis=1)[:,np.newaxis]
    return prob

X = df[['Sepal.Length','Petal.Width']].as_matrix()
y = df.species.as_matrix()
pi, mu, cov = discriminant_analysis(
    X,
    y,
    linear=False,
)
print('[linear=True, reg=0.00] accuracy={:0.4f}'.format(
    (np.argmax(predict_proba(X, pi, mu, cov), axis=1) == y).mean()
))

for reg in np.linspace(0.0,1.0,20):
    pi, mu, cov = discriminant_analysis(
```

```
        X,
        y,
        linear=True, reg=reg,
    )
    print('[linear=False,reg={:0.2f}] accuracy={:0.4f}'.format(
        reg, (np.argmax(predict_proba(X, pi, mu, cov), axis=1) == y).mean()
    ))
```

This outputs the following accuracies:

```
[linear=True, reg=0.00] accuracy=0.9667
[linear=False,reg=0.00] accuracy=0.9600
[linear=False,reg=0.05] accuracy=0.9600
[linear=False,reg=0.11] accuracy=0.9533
[linear=False,reg=0.16] accuracy=0.9533
[linear=False,reg=0.21] accuracy=0.9533
[linear=False,reg=0.26] accuracy=0.9533
[linear=False,reg=0.32] accuracy=0.9533
[linear=False,reg=0.37] accuracy=0.9533
[linear=False,reg=0.42] accuracy=0.9533
[linear=False,reg=0.47] accuracy=0.9533
[linear=False,reg=0.53] accuracy=0.9600
[linear=False,reg=0.58] accuracy=0.9600
[linear=False,reg=0.63] accuracy=0.9600
[linear=False,reg=0.68] accuracy=0.9600
[linear=False,reg=0.74] accuracy=0.9600
[linear=False,reg=0.79] accuracy=0.9600
[linear=False,reg=0.84] accuracy=0.9600
[linear=False,reg=0.89] accuracy=0.9600
[linear=False,reg=0.95] accuracy=0.9600
[linear=False,reg=1.00] accuracy=0.9600
```