Nicholas Trieu
MATH 189r
Homework 1 Part 3
September 26, 2016

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:

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

(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 λ from the validation set. Plot both λ 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 λ 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 ℓ_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.

Answers for 1:

(a) The maximum a posteriori problem is given as

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

Plugging in Gaussian density, we obtain

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

Note that the constant term $-(N\sigma + D\tau)\log\sqrt{2\pi}\sigma$ is not dependent on $\mathbf{w}$. So the above is equivalent to

$$\begin{aligned}
&\arg\max_{\mathbf{w}} \left(\sum_{i=1}^{N} -\frac{(y_i - w_0 - \mathbf{w}^\top \mathbf{x}_i)^2}{2\sigma^2} \right) + \left(\sum_{i=1}^{D} -\frac{w_j^2}{2\tau^2} \right) \\
&= \arg\max_{\mathbf{w}} \left(\sum_{i=1}^{N} -(y_i - w_0 - \mathbf{w}^\top \mathbf{x}_i)^2 \right) + \left(\sum_{i=1}^{D} -\frac{\sigma^2}{\tau^2} w_j^2 \right) \quad \text{scaling by } 2\sigma^2 \\
&= \arg\min_{\mathbf{w}} \left(\sum_{i=1}^{N} (y_i - w_0 - \mathbf{w}^\top \mathbf{x}_i)^2 \right) + \left(\sum_{i=1}^{D} \frac{\sigma^2}{\tau^2} w_j^2 \right) \quad \text{max is equivalent to min of the negative} \\
&= \arg\min_{\mathbf{w}} \left(\sum_{i=1}^{N} (y_i - w_0 - \mathbf{w}^\top \mathbf{x}_i)^2 \right) + \lambda ||\mathbf{w}||_2^2 \quad \text{subbing } \lambda = \sigma^2/\tau^2
\end{aligned}$$

as desired.

(b) The solution to the Ridge Regression problem

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

can be found by setting the gradient to 0:

$$\begin{aligned}
\nabla f &= \nabla \left[(A\mathbf{x} - \mathbf{b})^\top (A\mathbf{x} - \mathbf{b}) + (\mathbf{\Gamma x})^\top (\mathbf{\Gamma x}) \right] \quad \text{by definition of the L2 norm} \\
&= \nabla \left[(\mathbf{x}^\top A^\top - \mathbf{b}^\top)(A\mathbf{x} - \mathbf{b}) + \mathbf{x}^\top \mathbf{\Gamma}^\top \mathbf{\Gamma x} \right] \\
&= \nabla \left[(\mathbf{x}^\top A^\top A\mathbf{x} - \mathbf{x}^\top A^\top \mathbf{b} - \mathbf{b}^\top A\mathbf{x} + \mathbf{b}^\top \mathbf{b} + \mathbf{x}^\top \mathbf{\Gamma}^\top \mathbf{\Gamma x} \right] \\
&= \nabla \left[(\mathbf{x}^\top A^\top A\mathbf{x} - 2\mathbf{x}^\top A^\top \mathbf{b} + \mathbf{b}^\top \mathbf{b} + \mathbf{x}^\top \mathbf{\Gamma}^\top \mathbf{\Gamma x} \right] \quad \text{since } \mathbf{b}^\top A\mathbf{x} = \mathbf{x}^\top A^\top \mathbf{b} \\
&= 2A^\top A\mathbf{x} - 2A^\top \mathbf{b} + 2\mathbf{\Gamma}^\top \mathbf{\Gamma x} \\
0 &= 2A^\top A\mathbf{x}^\star - 2A^\top \mathbf{b} + 2\mathbf{\Gamma}^\top \mathbf{\Gamma x}^\star.
\end{aligned}$$

Solving for $\mathbf{x}$, we find that

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

Substituting $\mathbf{\Gamma} = \sqrt{\lambda}\mathbf{I}$, our problem becomes

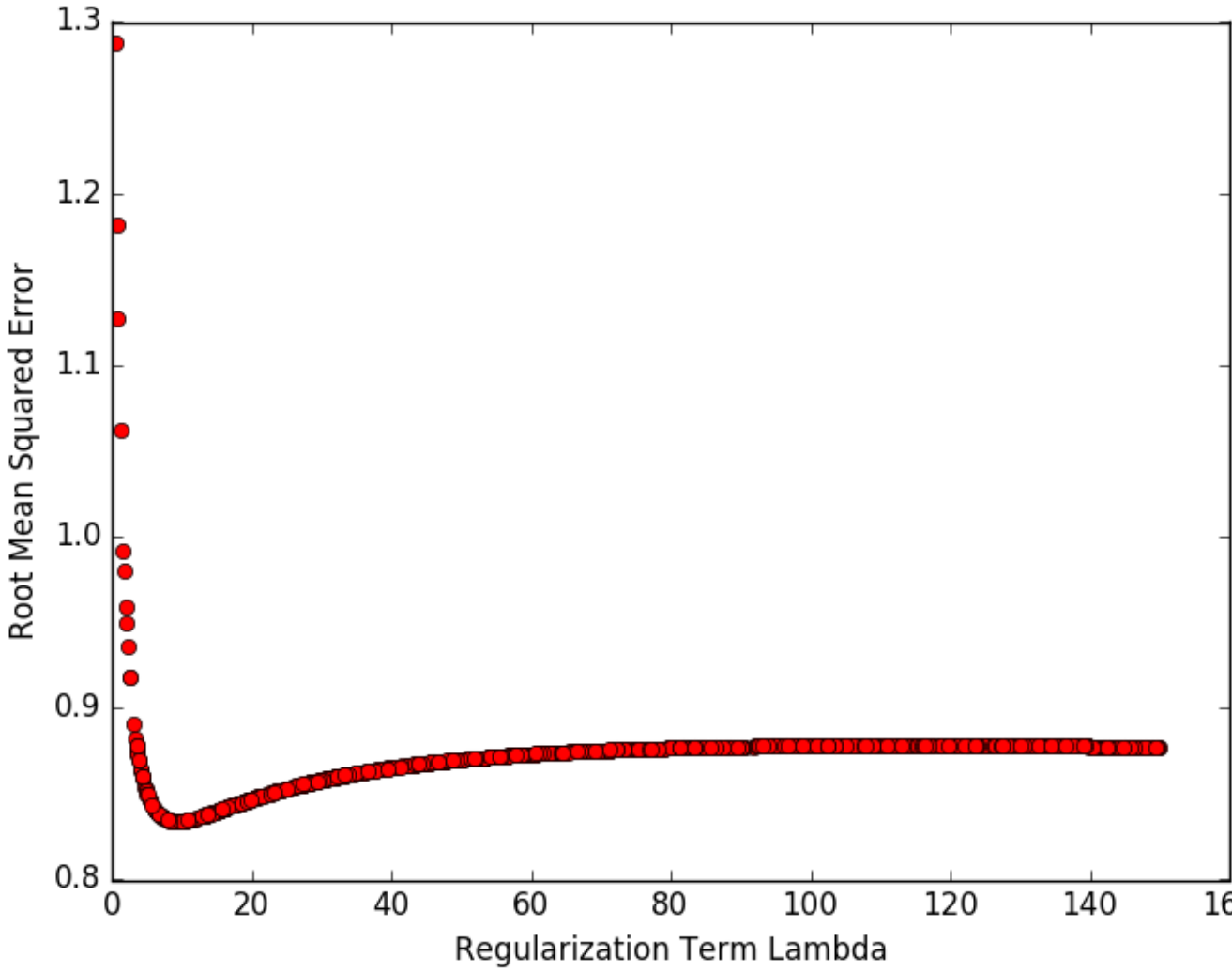
$$\text{minimize } f = ||A\mathbf{x} - \mathbf{b}||_2^2 + \lambda \mathbf{x}^\top \mathbf{x}$$

with the solution

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

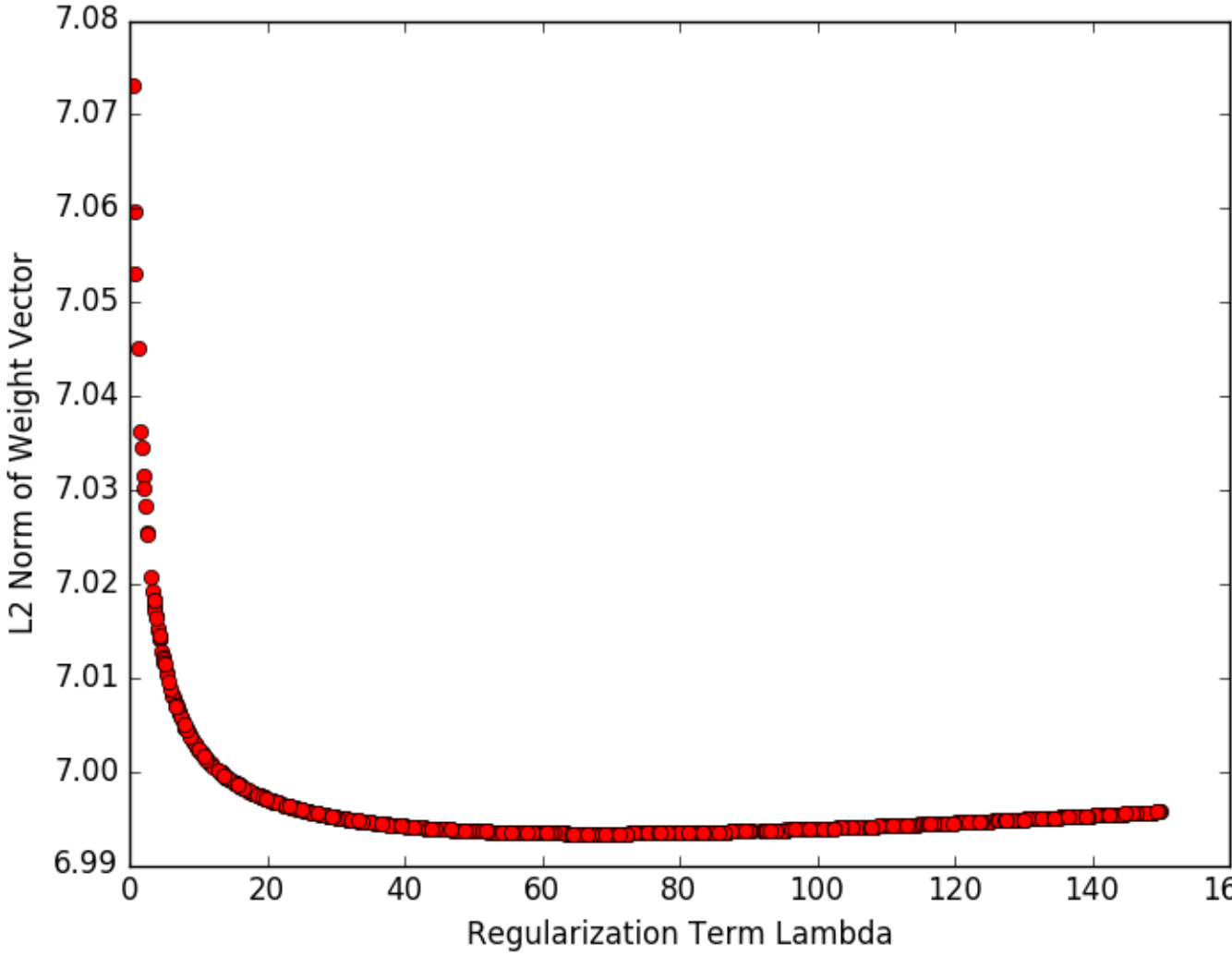
(c) Graphs:

Lambda vs validation RMSE



The optimal regularization parameter is $\lambda = 9.325579611587115$. The final RMSE on the test set with optimal $\lambda^{\star}$ is 0.82399780251275756.

Lambda vs Weight Vector



The code is below (indentation got cleared in verbatim):

```
import pandas as pd
df = pd.read_csv('https://math189r.github.io/hw/data/online_news_popularity/online_ne
import numpy as np
from scipy import sparse
import random

# 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 ridgereg(A, b, lam):
eye = np.eye(A.shape[1])
eye[0,0] = 0. # don't regularize bias term!
return np.linalg.solve(
A.T @ A + lam * eye,
A.T @ b
)

def rmse(predictions, targets):
return np.sqrt(((predictions - targets) ** 2).mean())

norms = []
val_RMSEs = []
lams = []
for i in range(1000):
```

```
lam = random.random()*150
theta_optimal = ridgereg(X_train, y_train, lam)
norms.append(np.linalg.norm(theta_optimal, 2)) # get the L2 norm of the weight vector
val_RMSEs.append(rmse(X_val @ theta_optimal, y_val))
lams.append(lam)

lam_opt = lams[np.argmin(val_RMSEs)]
theta_test = ridgereg(X_test, y_test, lam_opt)
rmse_test = rmse(X_test @ theta_test, y_test)
print(rmse_test)
```

(d) To solve the optimization problem

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

we set the gradient to 0 as in part (b). We first expand the above:

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

Taking the gradients, we get:

$$\begin{aligned}
0 = \nabla_{\mathbf{x}} f &= 2A^\top A\mathbf{x} + 2bA^\top\mathbf{1} - 2A^\top\mathbf{y} + 2\mathbf{\Gamma}^\top\mathbf{\Gamma x} \\
\mathbf{x}^\star &= (A^\top A + \mathbf{\Gamma}^\top\mathbf{\Gamma})(A^\top\mathbf{y} - bA^\top\mathbf{1}) \\
0 = \nabla_b f &= 2\mathbf{x}^\top A^\top\mathbf{1} + 2b\mathbf{1}^\top\mathbf{1} - 2\mathbf{1}^\top\mathbf{y} \\
&= 2\mathbf{x}^\top A^\top\mathbf{1} + 2bn - 2\mathbf{1}^\top\mathbf{y} \quad \text{where } n \text{ is the dimension of } \mathbf{1} \\
b^\star &= \frac{1}{n}(\mathbf{1}^\top\mathbf{y} - \mathbf{x}^\top A^\top\mathbf{1})
\end{aligned}$$

Solving this system of equations for $\mathbf{x}^\star$, we get:

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

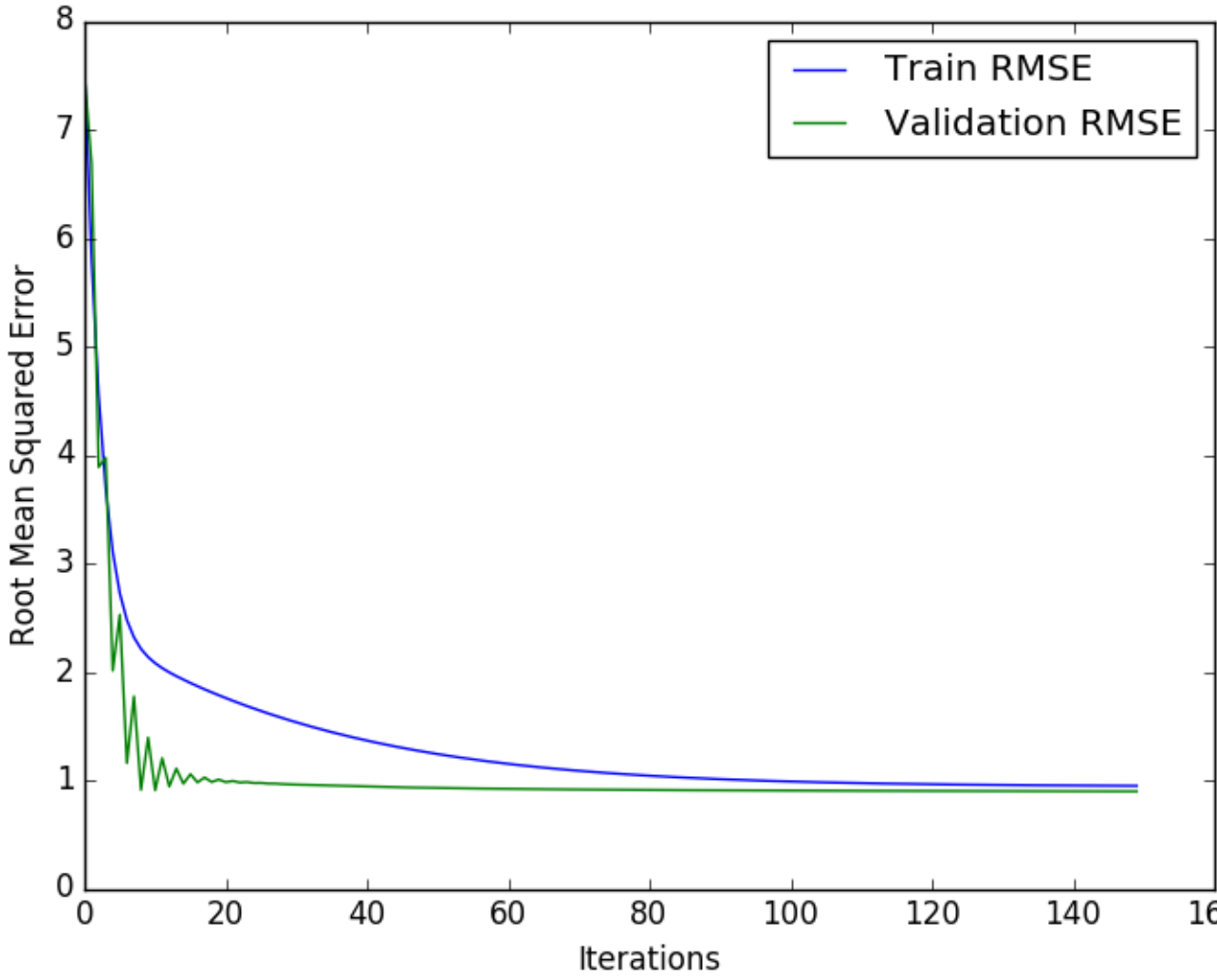
Code:

```
# ==> Distance between intercept and orig: 8.896883230136154e-12
# ==> Distance between theta and original: 1.774217265184486e-11
X_train_no_b = X_train[:,1:]
X_val_no_b = X_val[:,1:]
reg_opt = lams[np.argmin(val_RMSEs)]
```

```
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 = ridgereg(X_train, y_train, lam=reg_opt)
print('==> Distance between intercept and orig: {}'.format(
np.abs(original[0] - b_opt)[0]
))
print('==> Distance between theta and original: {}'.format(
np.linalg.norm(theta_opt - original[1:])
))
```

(e) **Gradient Descent**



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 = lams[np.argmin(val_RMSEs)]
theta_ = np.zeros(shape)
theta_optimal = ridgereg(X_train, y_train, lam = lams[np.argmin(val_RMSEs)])
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_tra
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.
)/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] - b_)[0]))
print('==> Distance between theta and orig: {}'.format(
np.linalg.norm(theta_-theta_optimal[1:])))
```

■

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 28×28 images of handwritten digits as well as the label of which digit $0 \leq$ `label` ≤ 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 `i`th row and `j`th 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 λ. 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 λ (try at least 10 different values including $\lambda = 0$) on the test set and report the value of λ 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 λ, the regularization parameter. Report the test accuracy for the optimal value of λ 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 λ. Only try 3 values for k (1, 5, and 10) and the ℓ_2 norm as your metric. Plot and report the same results as part (b).

(a)

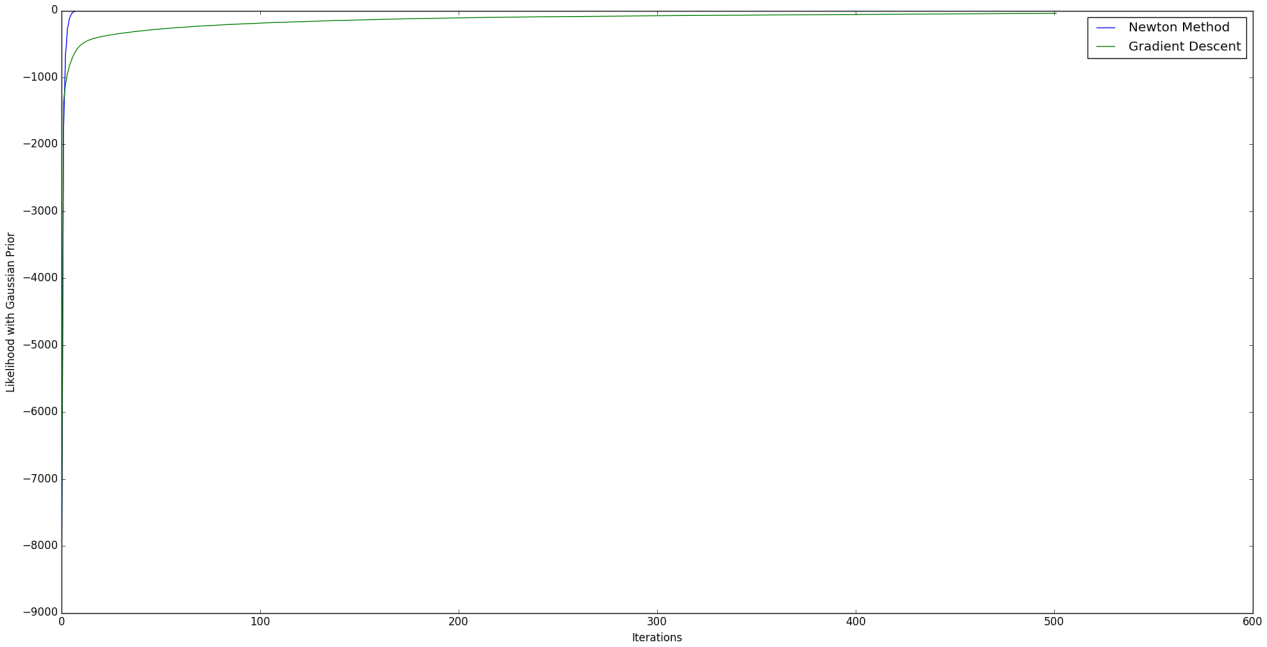
The logistic model $P(y = 1|\mathbf{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \mathbf{x})$ with a Gaussian prior on the weights has the log likelihood

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

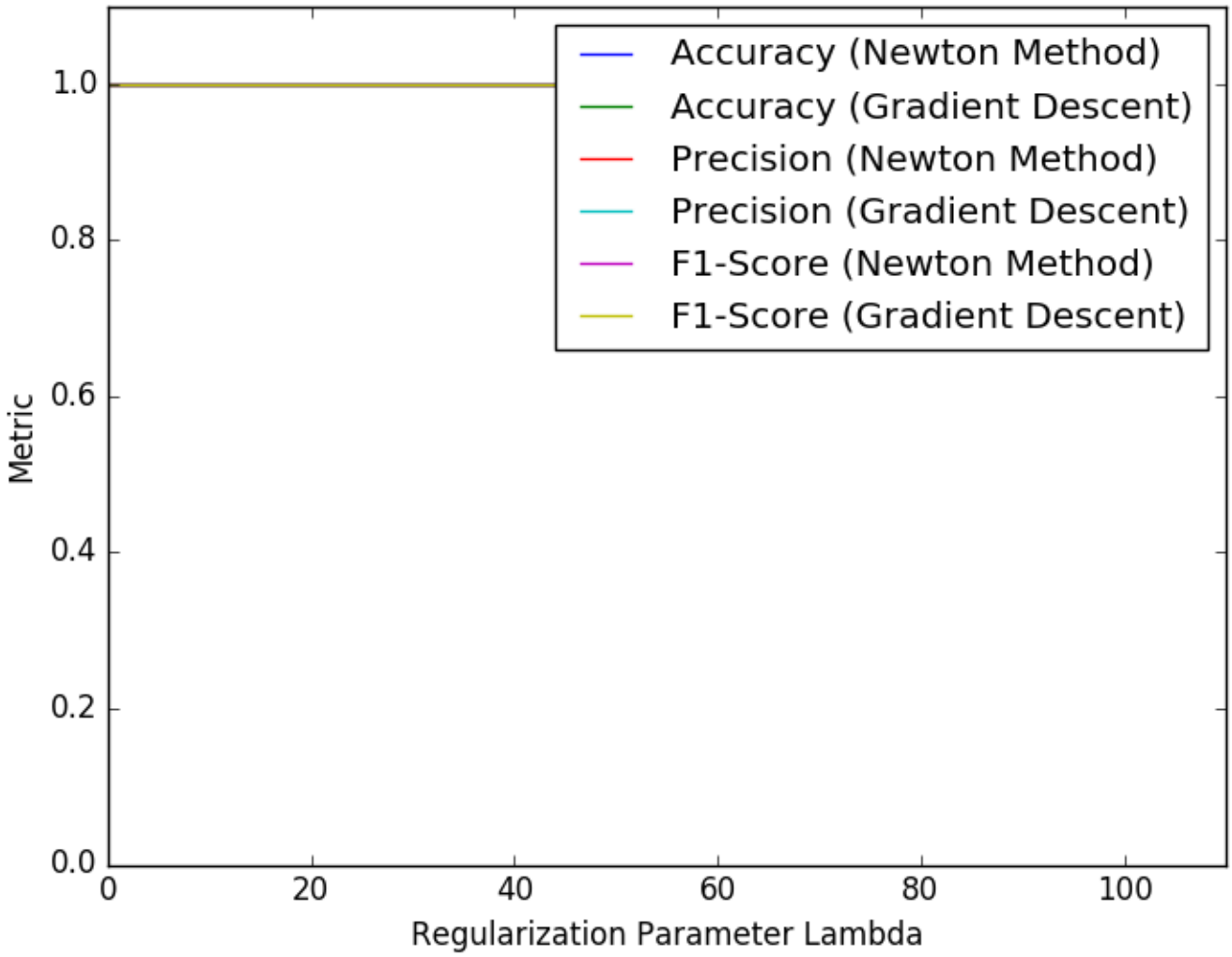
We solve first for the gradient and then the Hessian:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} l &= \sum_i y_i(1 - \sigma(\boldsymbol{\theta}^\top \mathbf{x})\mathbf{x} - (1 - y_i)\sigma(\boldsymbol{\theta}^\top \mathbf{x})\mathbf{x} + \lambda\boldsymbol{\theta} \\
&= \sum_i (y_i - \sigma(\boldsymbol{\theta}^\top \mathbf{x}_i))\mathbf{x}_i + \lambda\boldsymbol{\theta} \\
&= X^\top(y - \sigma(X\boldsymbol{\theta})) + \lambda\boldsymbol{\theta} \\
\nabla^2 l &= \frac{d}{d\boldsymbol{\theta}} \nabla l^\top \\
&= \sum_i \nabla_{\boldsymbol{\theta}} \sigma(\boldsymbol{\theta}^\top \mathbf{x})\mathbf{x}_i^\top + \lambda I \\
&= X^\top \text{diag}[\sigma(X\boldsymbol{\theta})(1 - \sigma(X\boldsymbol{\theta}))]X + \lambda I
\end{aligned}$$

Learning Curves



Metric Plot



We can see that all (reasonable) values of λ produce perfect results (accuracy ≈ 1.0).

Code:

```
import pandas as pd
d_test = pd.read_csv('http://pjreddie.com/media/files/mnist_test.csv',engine='python')
d_train = pd.read_csv('http://pjreddie.com/media/files/mnist_train.csv',engine='python')

m_test = d_test.as_matrix()
m_train = d_train.as_matrix()
y_train = m_train[:,0]
y_test = m_test[:,0]
X_train = m_train[:,1:]
X_test = m_test[:,1:]

y_bin_train = np.array([row for row in y_train if row == 0 or row == 1])
y_bin_test = np.array([row for row in y_test if row ==0 or row == 1])
X_bin_train = np.array([row[1:] for row in m_train if row[0] == 0 or row[0] == 1])
X_bin_test = np.array([row[1:] for row in m_test if row[0] == 0 or row[0] == 1])

import numpy as np
from scipy import sparse
from scipy import linalg

def linreg(A, b, reg):
eye = np.eye(A.shape[1])
return np.linalg.solve(
A.T @ A + reg * eye,
A.T @ b
)

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-8, tol=1e-6,
max_iters=500, 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
ll = log_likelihood(X, y, theta, reg=reg)
objective.append(ll)

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

def accuracy(predictions, targets):
numCorrect = 0.0
for i in range(len(predictions)):
if predictions[i] == targets[i]:
numCorrect = numCorrect + 1
return numCorrect/len(predictions)

def precision(predictions, targets):
numCorrect = 0.0
numPred1s = 0
for i in range(len(predictions)):
if predictions[i] == 1:
numPred1s = numPred1s + 1
if (targets[i] == 1):
numCorrect = numCorrect + 1
return numCorrect/numPred1s

def rPrecision(predictions, targets):
numCorrect = 0.0
numTar1s = 0
for i in range(len(predictions)):
if targets[i] == 1:
numTar1s = numTar1s + 1
if (predictions[i] == 1):
numCorrect = numCorrect + 1
return numCorrect/numTar1s

def f1(predictions, targets):
p = precision(predictions, targets)
r = rPrecision(predictions, targets)
```

```
return 2*p*r/(p+r)

#theta = linreg(X_bin_train, y_bin_train*2 - 1, reg=1e-2)
#y_pred = X_bin_train @ theta > 0
#accuracy(y_pred, y_bin_train) # 0.997789

regs = [0,10,20,30,40,50,60,70,80,90,100]
accs_newt = []
ps_newt = []
f1s_newt = []
accs_grad = []
ps_grad = []
f1s_grad = []

for i in range(len(regs)):
reg = regs[i]
theta_newt, obj_newt = lr_newton(X_bin_train, y_bin_train)
theta_grad, obj_grad = lr_grad(X_bin_train, y_bin_train)
y_pred_newt = X_bin_test @ theta_newt > 0
y_pred_grad = X_bin_test @ theta_grad > 0
accs_newt.append(accuracy(y_pred_newt, y_bin_test))
accs_grad.append(accuracy(y_pred_grad, y_bin_test))
ps_newt.append(precision(y_pred_newt, y_bin_test))
ps_grad.append(precision(y_pred_grad, y_bin_test))
f1s_newt.append(f1(y_pred_newt, y_bin_test))
f1s_grad.append(f1(y_pred_grad, y_bin_test))

y_pred_newt = X_bin_test @ theta_newt > 0
y_pred_grad = X_bin_test @ theta_grad > 0
acc_newt_test = accuracy(y_pred_newt, y_bin_test)
acc_grad_test = accuracy(y_pred_grad, y_bin_test)
print(acc_newt_test)
print(acc_grad_test)
```

(b)
The softmax regression problem has log likelihood

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

Taking the gradient, we find that

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

where $\mathbf{y} \in \{0,1\}^{n \times c}$ is defined as

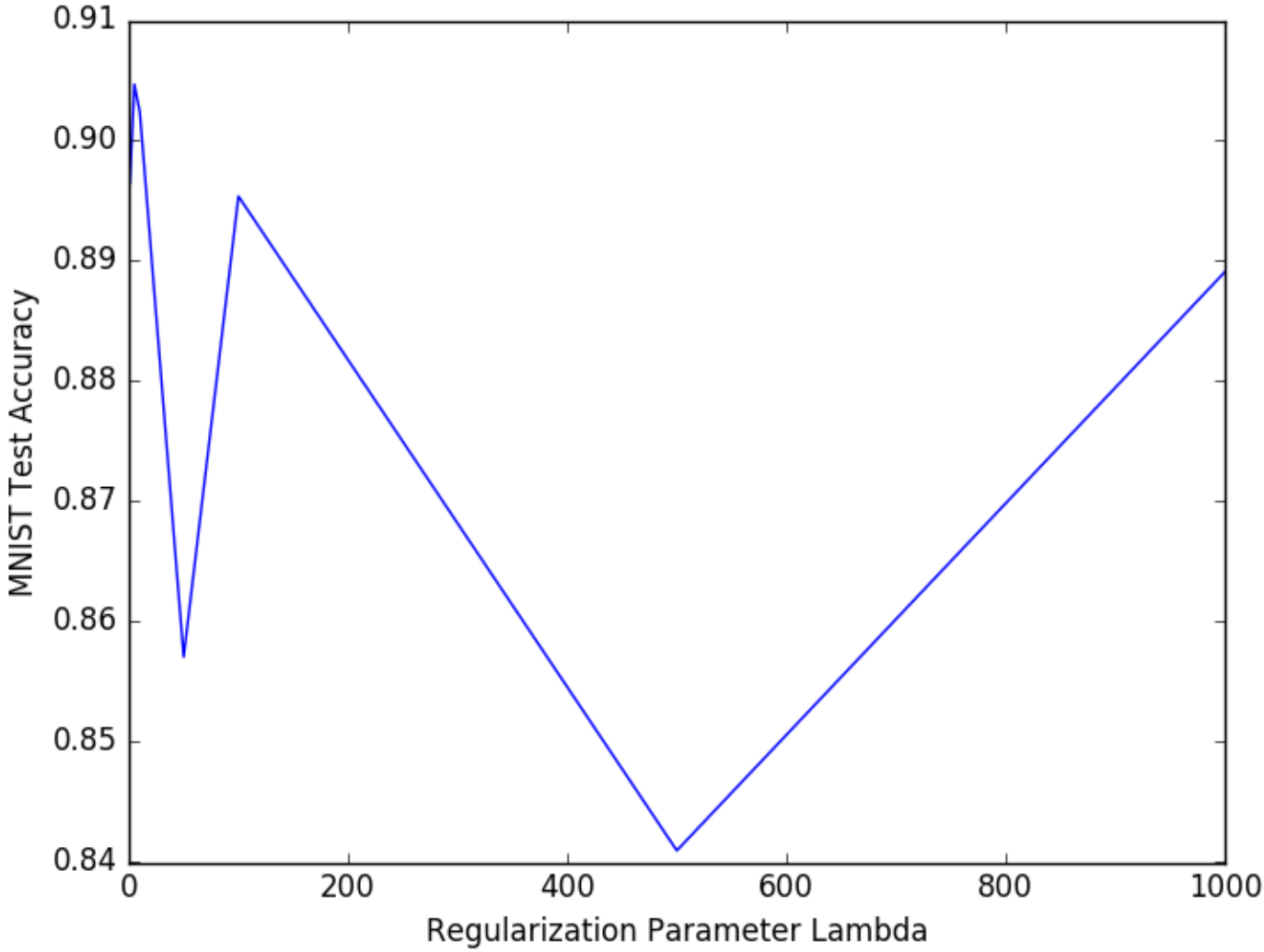
$$\mathbf{y}_{ij} = \begin{cases} 1 & \text{if datum } i \text{ is digit } j \\ 0 & \text{otherwise} \end{cases}$$

and $\mathbf{y}\mathbf{1}_c = \mathbf{1}_n$, and $\boldsymbol{\mu} \in [0,1]^{n \times c}$ is defined as

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

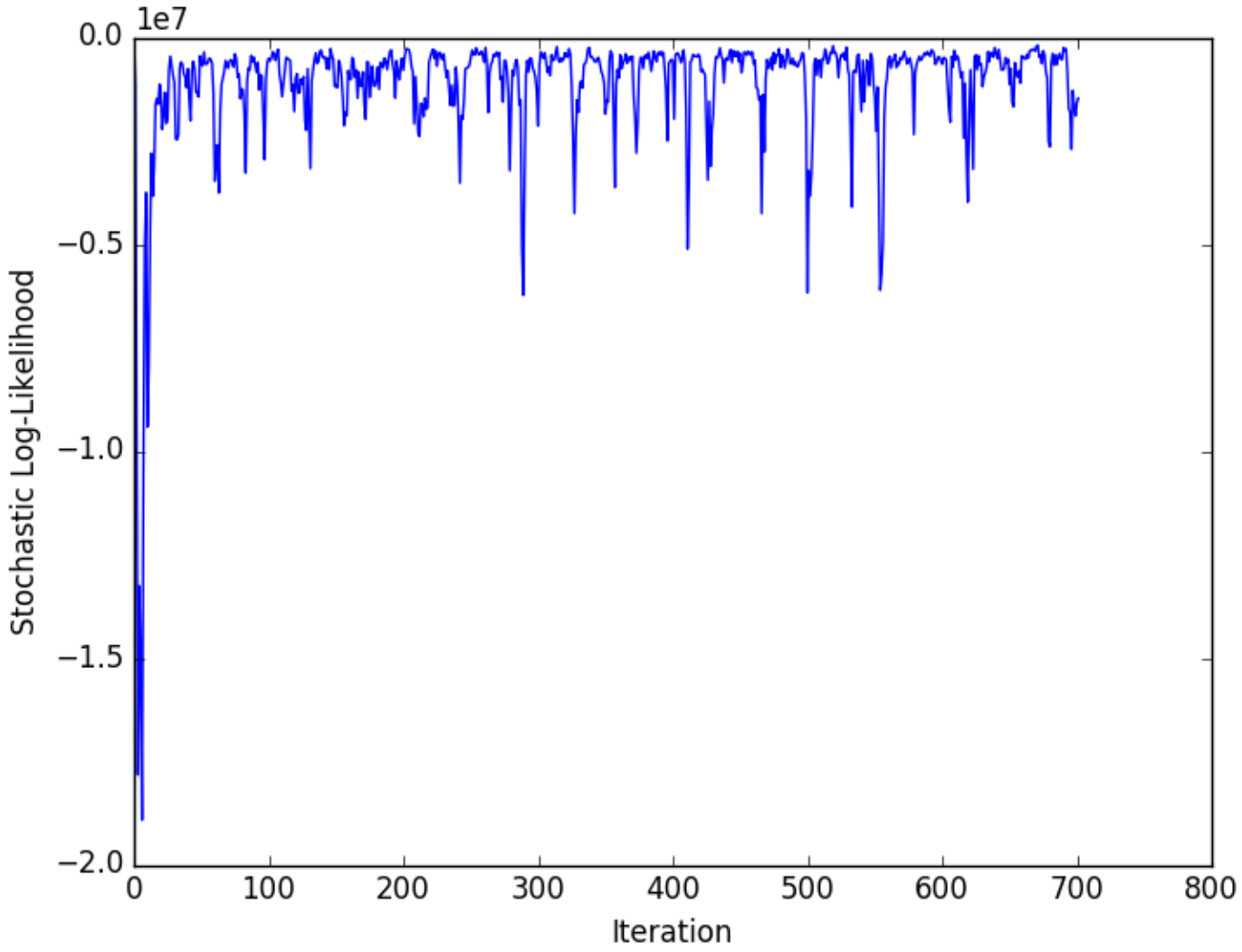
where exp applies elementwise to the matrix arguments.

Lambda vs MNIST Test Accuracy



The maximum test accuracy was 90.47% with $\lambda = 5$.

Softmax Regression on MNIST w/ Optimal Regularizer $\lambda = 5$



Code:

```
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from scipy.misc import logsumexp

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 softmax_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 softmax_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=700, batch_size=256,
verbose=False, print_freq=50,
):
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 = [softmax_log_likelihood(X[ind], y_one_hot[ind], W, reg=reg)]
grad = softmax_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(
```

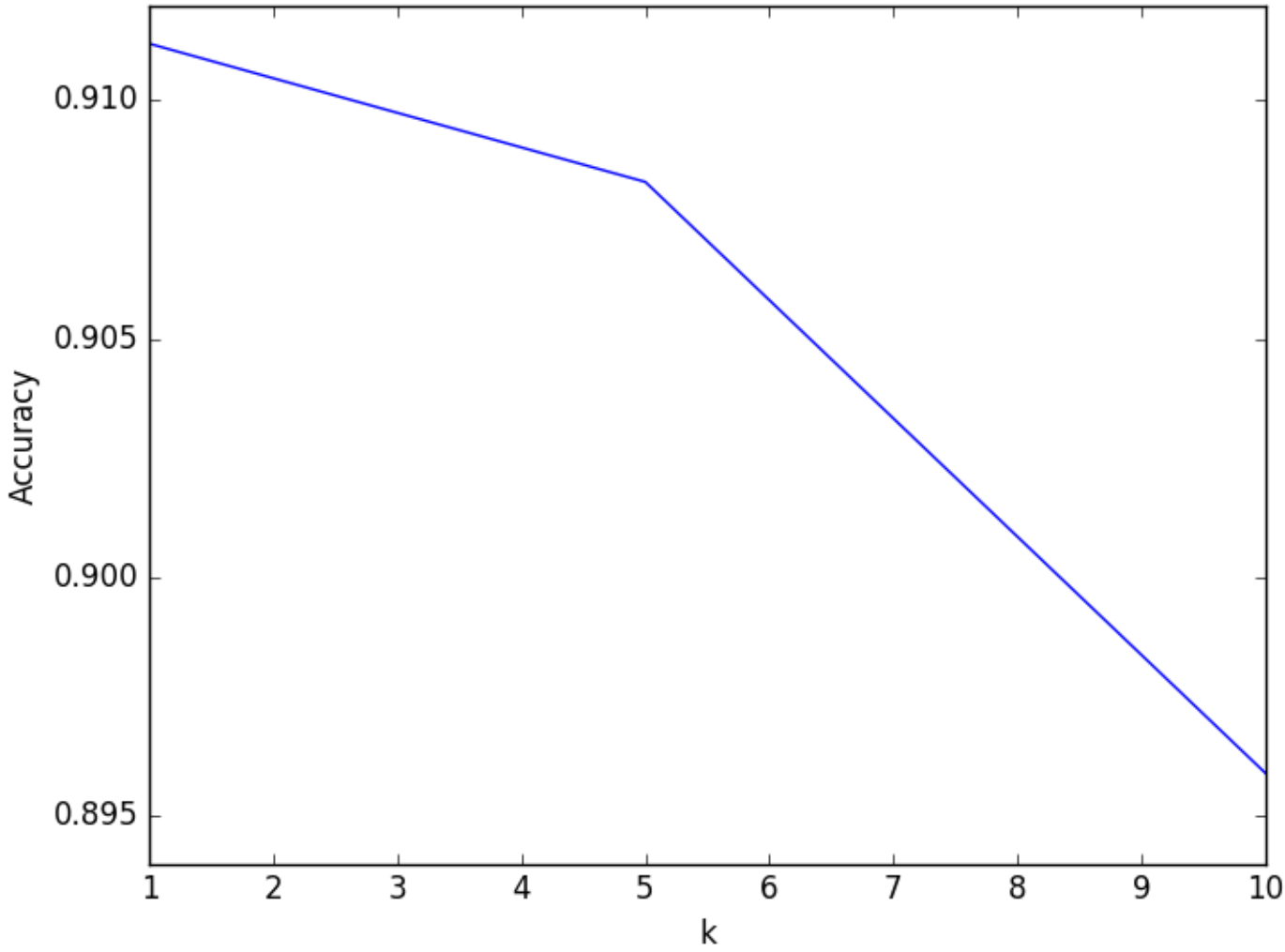
```
len(objective)-1, objective[-1], np.linalg.norm(grad)
))
ind = np.random.randint(0, X.shape[0], size=batch_size)
grad = softmax_grad_log_likelihood(X[ind], y_one_hot[ind], W, reg=reg)
W = W - lr * grad
objective.append(softmax_log_likelihood(X[ind], y_one_hot[ind], W, reg=reg))

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

regs = [1,5,10,50,100,500,1000]
accs = []
for reg in regs:
w, obj = softmax_grad(X_train, y_train, reg = reg)
ytest_pred = []
for row in (X_test @ w):
max_ll = row[0]
max_digit = 0
for i in range(len(row)):
if (row[i] > max_ll):
max_ll = row[i]
max_digit = i
ytest_pred.append(max_digit)
accs.append(accuracy(ytest_pred,y_test))
```

(c)
I down-sampled the dataset to 3000 datapoints because KNN performs slow.

Accuracies for k



Code:

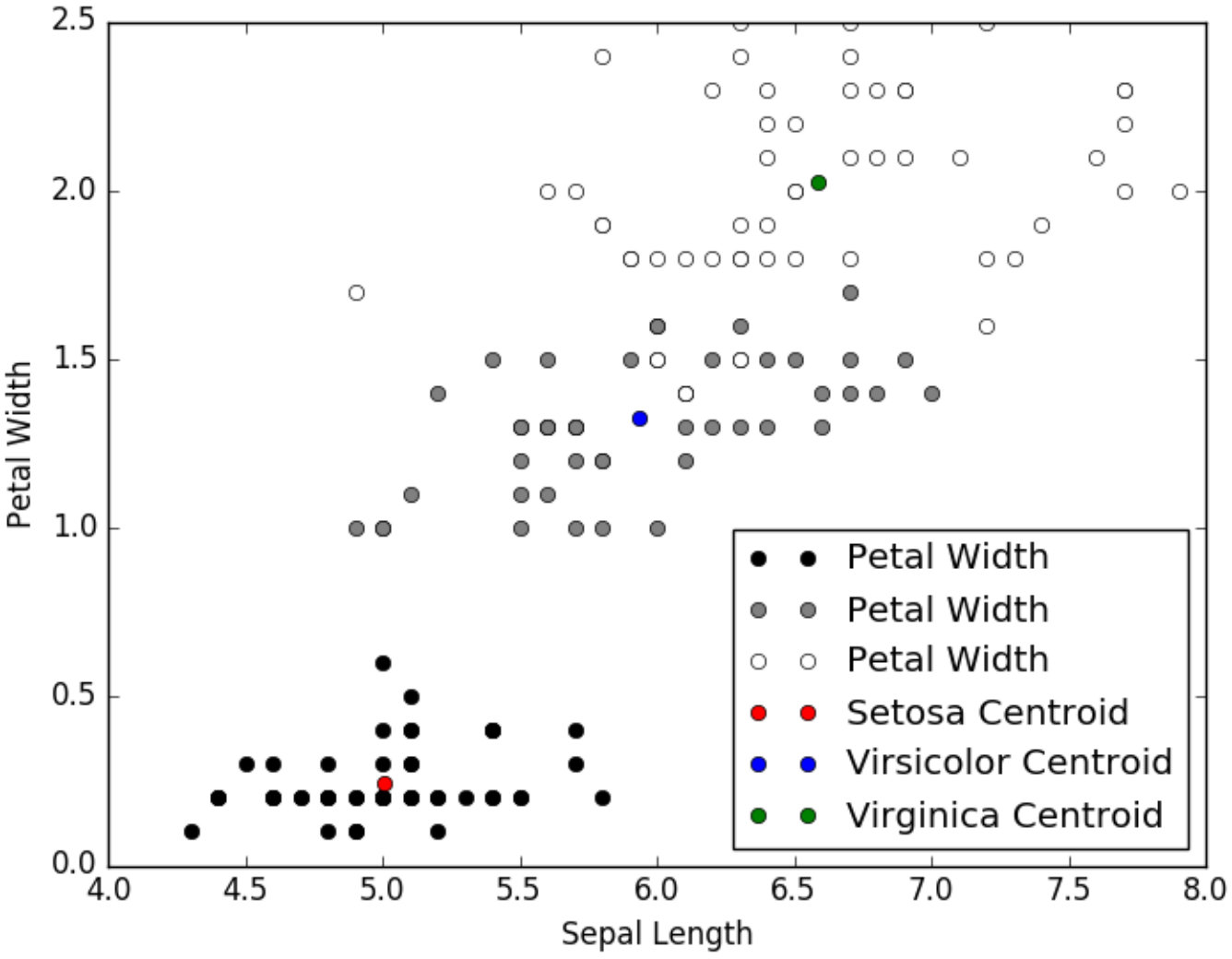
```
# downsampling the training set to 3000 data points we get the following
# [k=1] accuracy: 0.9111911191119112
# [k=5] accuracy: 0.9082908290829083
# [k=10] accuracy: 0.8958895889588959

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

X_train_down = np.array(X_train[::20,::])
y_train_down = np.array(y_train[::20])
ks = [1,5,10]
accuracies = []
for k in ks:
acc_k = accuracy(y_test, predict_knn(
X_test, X_train_down, y_train_down, k=k,
verbose=True, print_freq=1000,
))
accuracies.append(acc_k)
print('[k={}] accuracy: {}'.format(
k,
acc_k,
))
```

■

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 λ, the nonlinear discriminant analysis you plotted above}.



Code:

```
import numpy as np
import pandas as pd
from collections import defaultdict
```

```
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_prob(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

df = pd.read_csv('https://vincentarelbundock.github.io/Rdatasets/csv/datasets/iris.csv')
X = df[['Sepal.Length','Petal.Width']].as_matrix()
y = df['Species'].as_matrix()
for i in range(len(y)):
if y[i] == 'setosa':
y[i] = 0
elif y[i] == 'versicolor':
y[i] = 1
else:
```

```
y[i] = 2
pi, mu, cov = discriminant_analysis(
X,
y,
linear=False,
)
print('[linear=False, reg=0.00] accuracy={:0.4f}'.format(
(np.argmax(predict_prob(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=True,reg={:0.2f}] accuracy={:0.4f}'.format(
reg, (np.argmax(predict_prob(X, pi, mu, cov), axis=1) == y).mean()
))

setosa_sepal_lengths = []
setosa_petal_widths = []
virsicolor_sepal_lengths = []
virsicolor_petal_widths = []
virginica_sepal_lengths = []
virginica_petal_widths = []
for i in range(len(X)):
sl = X[i][0]
pw = X[i][1]
if (y[i] == 0):
setosa_sepal_lengths.append(sl)
setosa_petal_widths.append(pw)
elif (y[i] == 1):
virsicolor_sepal_lengths.append(sl)
virsicolor_petal_widths.append(pw)
else:
virginica_sepal_lengths.append(sl)
virginica_petal_widths.append(pw)

import matplotlib.pyplot as plt
plt.plot(setosa_sepal_lengths, setosa_petal_widths, color = '0.0', marker = 'o', fillsty
plt.plot(virsicolor_sepal_lengths, virsicolor_petal_widths, color = '0.5', marker = 'o',
plt.plot(virginica_sepal_lengths, virginica_petal_widths, color = '1.0', marker = 'o',fi
plt.plot(mu[0][0], mu[0][1], 'ro', label='Setosa Centroid')
plt.plot(mu[1][0], mu[1][1], 'bo', label='Virsicolor Centroid')
plt.plot(mu[2][0], mu[2][1], 'go', label='Virginica Centroid')
```

```
plt.xlabel('Sepal Length')
plt.ylabel('Petal Width')
plt.legend(loc=4)
plt.show()
```

Accuracy outputs:

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

■