Nicholas Trieu
MATH 189r
Homework 2 Part 2
October 10, 2016

4. (Lasso Feature Selection) Ignoring undifferentiability at $x = 0$, take $\frac{\partial |x|}{\partial x} = \text{sign}(x)$. Using this, show that $\nabla \|\mathbf{x}\|_1 = \text{sign}(\mathbf{x})$ where sign is applied elementwise. Derive the gradient of the ℓ_1 regularized linear regression objective

$$\text{minimize: } \|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x}\|_1$$

Now consider the shares dataset we used in problem 1 of homework 1 (`https://math189r.github.io/hw/data/online_news_popularity/online_news_popularity.txt`). Implement a gradient descent based solution of the above optimization problem for this data. Produce the convergence plot (objective vs. iterations) for a non-trivial value of λ. In the same figure (and different axes) produce a 'regularization path' plot. Detailed more in section 13.3.4 of Murphy, a regularization path is a plot of the optimal weight on the y axis at a given regularization strength λ on the x axis. Armed with this plot, provide an ordered list of the top five features in predicting the log-shares of a news article from this dataset (with justification). We can see a more detailed analysis of this at `https://en.wikipedia.org/wiki/Proximal_gradient_methods_for_learning` and `https://web.stanford.edu/~boyd/papers/pdf/prox_algs.pdf` but you will have to wrap the gradient descent step with a threshold function

$$\text{prox}_\gamma(\mathbf{x})_i = \begin{cases} \mathbf{x}_i - \gamma & \mathbf{x}_i > \gamma \\ 0 & |\mathbf{x}_i| \leq \gamma \\ \mathbf{x}_i + \gamma & x_i < -\gamma \end{cases}$$

so that each iterate

$$\mathbf{x}_{i+1} = \text{prox}_{\lambda\gamma}\left(\mathbf{x}_i - \gamma \nabla f(\mathbf{x}_i)\right)$$

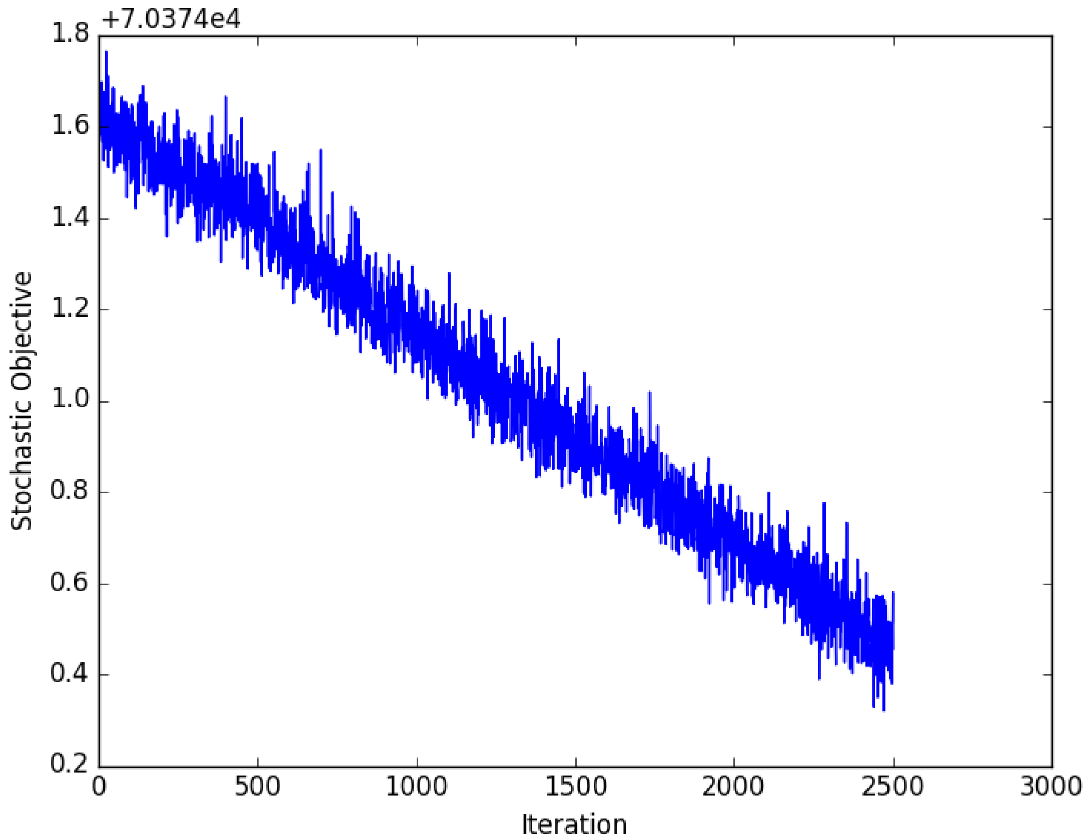
where γ is your learning rate. Tip: you can reuse most of your code from the first homework.

Answers for 4:

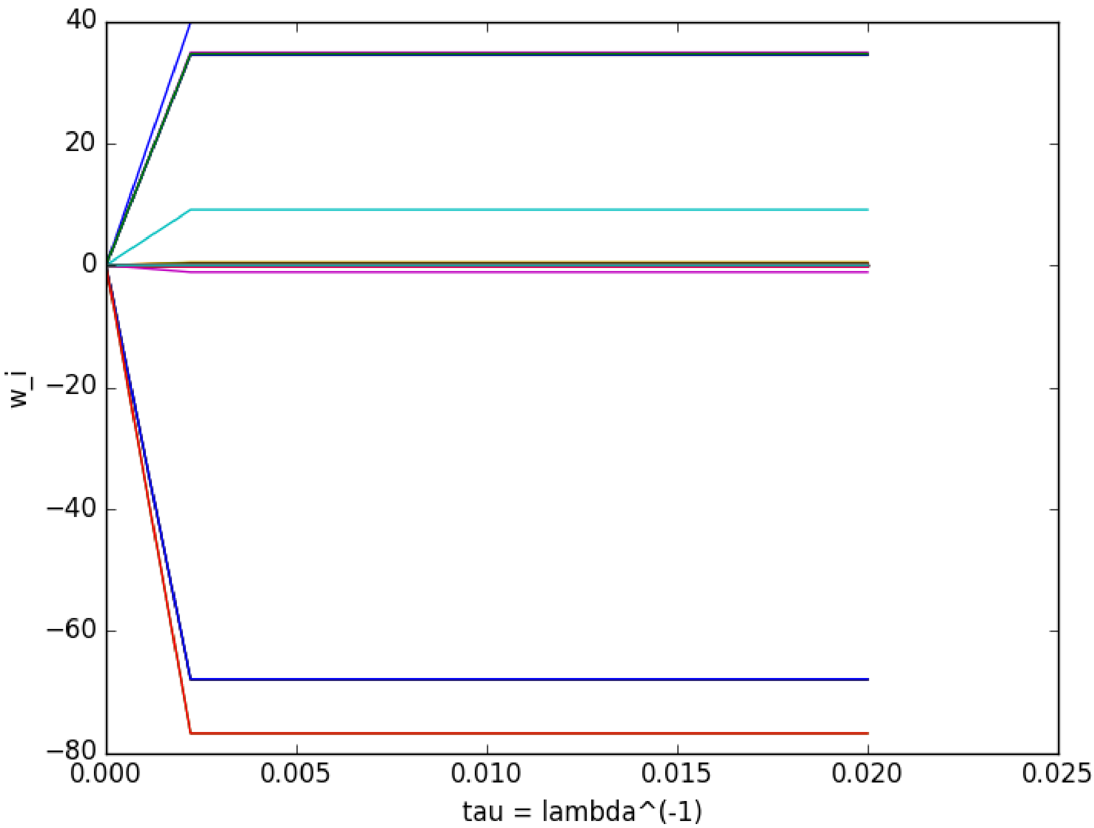
Note that $\dfrac{\delta \|\mathbf{x}\|_1}{\delta \mathbf{x}_i} = \dfrac{\delta \sum |\mathbf{x}_i|}{\delta \mathbf{x}_i} = \text{sign}(\mathbf{x}_i)$. So $\nabla \|\mathbf{x}\|_1 = \text{sign}(\mathbf{x})$. It follows that the gradient is:

$$\begin{aligned} \nabla[\|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x}\|_1] &= \nabla[\|\mathbf{x}^\top A^\top A\mathbf{x} - 2\mathbf{b}^\top A\mathbf{x} + \mathbf{b}^\top \mathbf{b} + \lambda \|\mathbf{x}\|_1] \\ &= 2A^T A\mathbf{x} - 2\mathbf{b}^\top A + \lambda \text{sign}(\mathbf{x}) \end{aligned}$$

Lasso Objective Convergence ($\lambda = 100000$):



Lasso Path:



The top five features in predicting the log-shares of a news article are:
'is_weekend',
'weekday_is_sunday',
'weekday_is_wednesday',
'weekday_is_thursday',
'weekday_is_friday'

Code:

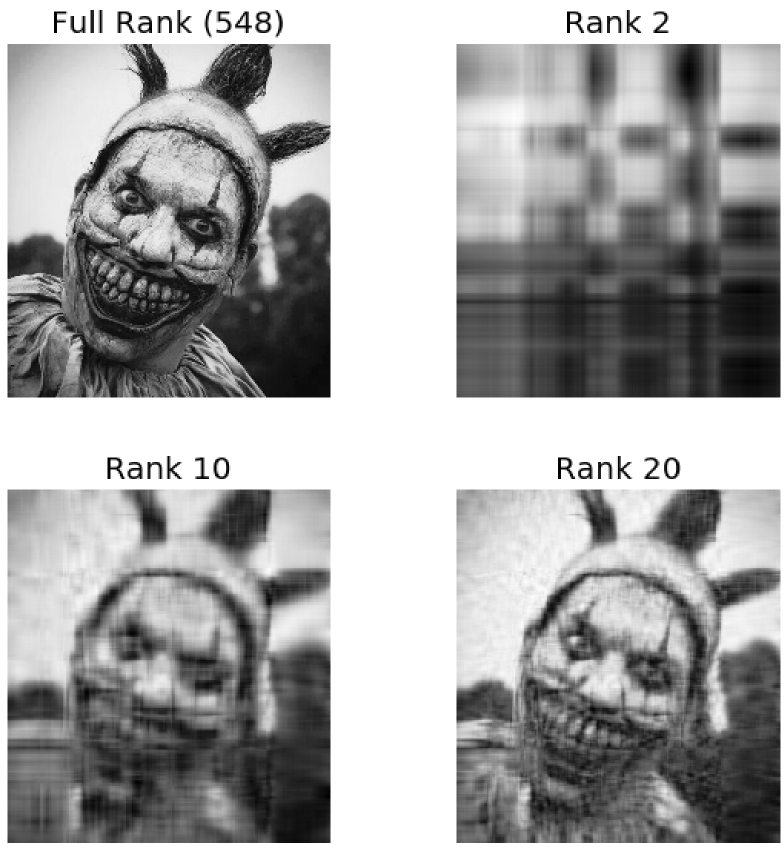
```
import pandas as pd
df = pd.read_csv('https://math189r.github.io/hw/data/online_news_popularity/online_news_
import numpy as np
from scipy import sparse
import random
import matplotlib.pyplot as plt
import math
X, y = df[[col for col in df.columns if col not in ['url', 'shares', 'cohort']]], np.log
X = np.hstack((np.ones_like(y), X))

def objective(X, y, w, reg=1e-6):
```

```
    # compute the L1 regularized lin reg obj
    err = X @ w - y # get the error vector
    err = math.sqrt(float(err.T @ err)) # compute the L2 norm of the error
    return (err + reg * np.abs(w).sum())/len(y)

def grad_objective(X, y, w):
    return X.T @ (X @ w - y) / len(y)

def prox(x, gamma):
    # Compute the lasso proximal operator.
    # Note: modifies x in-place.

    # The three cases.
    x[np.abs(x) <= gamma] = 0.
    x[x > gamma] = x[x > gamma] - gamma
    x[x < -gamma] = x[x < -gamma] + gamma
    return x

def lasso_grad(
    X, y, reg=1e-6, lr=1e-3, tol=1e-6,
    max_iters=300, batch_size=256, eps=1e-5,
    verbose=False, print_freq=5,
):
    # Same as gradient descent, but with the prox wrapper.

    y = y.reshape(-1,1)
    w = np.linalg.solve(X.T @ X, X.T @ y)

    ind = np.random.randint(0, X.shape[0], size=batch_size)
    obj = [objective(X[ind], y[ind], w, reg=reg)]
    grad = grad_objective(X[ind], y[ind], w)

    while len(obj)-1 <= max_iters and np.linalg.norm(grad) > tol:
        if verbose and (len(obj)-1) % print_freq == 0:
            print([i={}] objective: {}. sparsity = {:0.2f}.format(
                len(obj)-1, obj[-1], (np.abs(w) < reg*lr).mean()
            ))
        ind = np.random.randint(0, X.shape[0], size=batch_size)
        grad = grad_objective(X[ind], y[ind], w)

        # Wrap with prox.
        w = prox(w - lr * grad, reg*lr)
        obj.append(objective(X[ind], y[ind], w, reg=reg))

    if verbose:
```

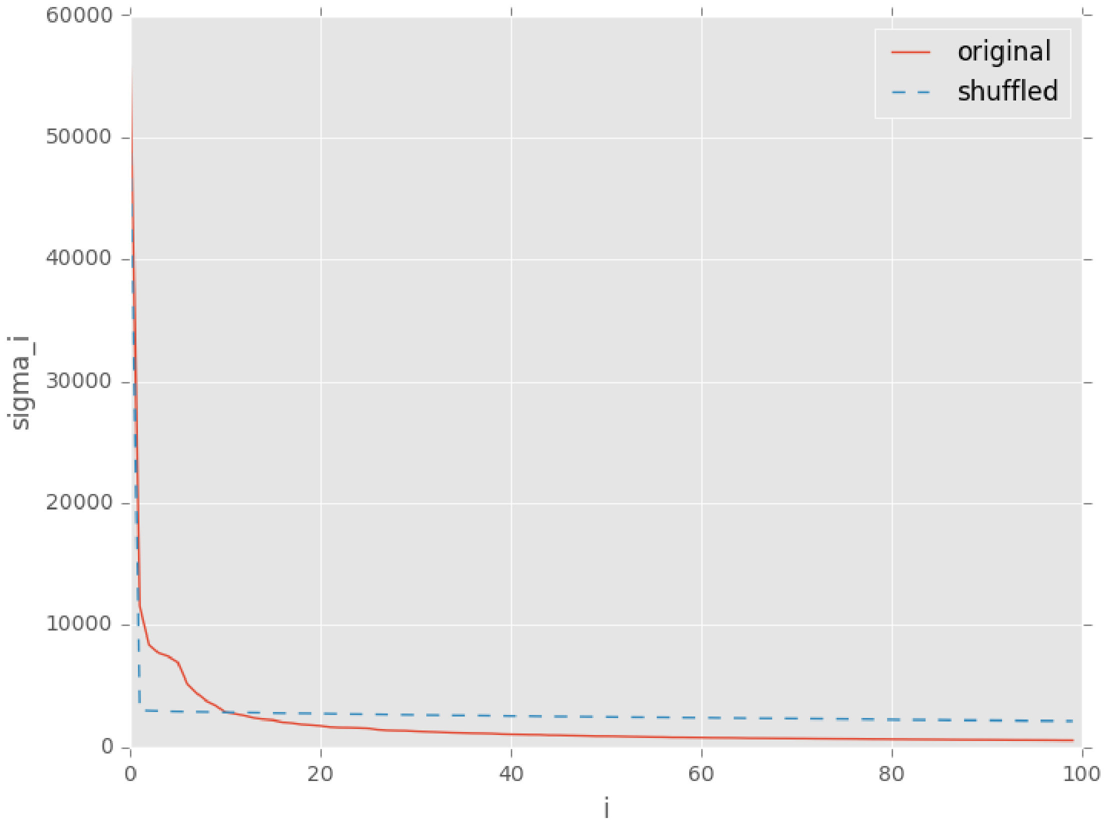
```
        print([i={}] done. sparsity = {:0.2f}.format(
            len(obj)-1, (np.abs(w) < reg*lr).mean()
        ))
    return w, obj

def lasso_path(
    X, y, reg_min=1e-8, reg_max=10,
    regs=10, **grad_args
):
    W = np.zeros((X.shape[1], regs))
    tau = np.linspace(reg_min, reg_max, regs)
    for i in range(regs):
        W[:,i] = lasso_grad(
            X, y, reg=1/tau[i], max_iters=1000,
            batch_size=1024, **grad_args
        )[0].flatten()
    return tau, W

# plot the lasso path
tau, W = lasso_path(X, y, reg_min=1e-15, reg_max=0.02, regs=10, lr=1e-12)
plt.xlabel("tau = lambda^(-1)")
plt.ylabel("w_i")
plt.plot(tau, W.T)
plt.show()

# find the top 5 important features
important_features=np.array(df.columns)[np.argsort(W[:,9])[:5]+1]
print(important_features)

# plot the stochastic objective
lr = 1e-12
reg = 1e5
w, obj = lasso_grad(
    X, y, reg=reg, lr=lr, eps=1e-2,
    max_iters=2500, batch_size=1024,
    verbose=True, print_freq=250,
)

plt.ylabel("Stochastic Objective")
plt.xlabel("Iteration")
plt.plot(obj)
plt.show()
```

5. (SVD Image Compression) Load the image of a scary clown at `http://i.imgur.com/X017qGH.jpg` into a matrix/array. Plot the progression of the 100 largest singular values for the original image and a randomly shuffled version of the same image (all on the same plot). In a single figure plot a grid of four images: the original image, and a rank k truncated SVD approximation of the original image for $k \in \{2, 10, 20\}$.

Answers to 5:



Singular Value Dropoff:



Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import ndimage
import urllib

# Load the clown image.
plt.style.use('ggplot')
orig_img = ndimage.imread(
   urllib.request.urlopen('http://i.imgur.com/X017qGH.jpg'),
   flatten=True,
)

# Create the randomly shuffled image.
random_shuffle = orig_img.copy().flatten()
np.random.shuffle(random_shuffle)
random_shuffle = random_shuffle.reshape(orig_img.shape)
```

```
# Do SVD on both images.
_, s, _ = np.linalg.svd(orig_img)
_, s_, _ = np.linalg.svd(random_shuffle)

# Plot the Singular Value Dropoff.
k = 100
plt.plot(s[:k], label='original')
plt.plot(s_[:k], '--', label='shuffled')
plt.legend()
plt.ylabel('sigma_i')
plt.xlabel('i')
plt.show()

# Setup for the SVD truncated approximations.
U_, s_, V_ = np.linalg.svd(orig_img)
S_ = np.zeros((U_.shape[0],V_.shape[0]))
S_[:V_.shape[0], :V_.shape[0]] = np.diag(s_)

# Plot the SVD approximations for the 4 ranks.
plt.figure(figsize=(6,6))
plt.subplot(2,2,1)
plt.imshow(U_ @ S_ @ V_, cmap='Greys_r')
plt.title("Full Rank ({0})".format(max(orig_img.shape[0], orig_img.shape[1])))
plt.axis('off')

ranks = [2,10,20]
for i, k in enumerate(ranks):
   plt.subplot(2,2,i+2)
   plt.imshow(U_[:,:k] @ S_[:k,:] @ V_, cmap='Greys_r')
   plt.title("Rank {0}".format(k))
   plt.axis('off')

plt.tight_layout()
plt.show()
```

6. (Murphy 12.5 - Deriving the Residual Error for PCA) It may be helpful to reference section 12.2.2 of Murphy.

(a) Prove that

$$\left\| \mathbf{x}_i - \sum_{j=1}^{k} z_{ij}\mathbf{v}_j \right\|^2 = \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{v}_j.$$

Hint: first consider the case when $k = 2$. Use the fact that $\mathbf{v}_i^\top \mathbf{v}_j$ is 1 if $i = j$ and 0 otherwise. Recall that $z_{ij} = \mathbf{x}_i^\top \mathbf{v}_j$.

(b) Now show that

$$J_k = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{v}_j \right) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \lambda_j.$$

Hint: recall that $\mathbf{v}_j^\top \boldsymbol{\Sigma} \mathbf{v}_j = \lambda_j \mathbf{v}_j^\top \mathbf{v}_j = \lambda_j$.

(c) If $k = d$ there is no truncation, so $J_d = 0$. Use this to show that the error from only using $k < d$ terms is given by

$$J_k = \sum_{j=k+1}^{d} \lambda_j.$$

Hint: partition the sum $\sum_{j=1}^{d} \lambda_j$ into $\sum_{j=1}^{k} \lambda_j$ and $\sum_{j=k+1}^{d} \lambda_j$.

Answers to 6:

(a) Proof:

$$
\begin{aligned}
\|\mathbf{x}_i - \sum_{j=1}^{k} z_{ij}\mathbf{v}_j\|_2^2 &= \left(\mathbf{x}_i - \sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right)^\top \left(\mathbf{x}_i - \sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right) \quad \text{by definition of the L2 norm} \\
&= \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} z_{ij}\mathbf{v}_j^\top \mathbf{x}_i - \mathbf{x}_i^\top \sum_{j=1}^{k} z_{ij}\mathbf{v}_j + \left(\sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right)^\top \left(\sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right) \quad \text{expanding} \\
&= \mathbf{x}_i^\top \mathbf{x}_i - 2\sum_{j=1}^{k} z_{ij}\mathbf{v}_j^\top \mathbf{x}_i + \left(\sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right)^\top \left(\sum_{j=1}^{k} z_{ij}\mathbf{v}_j\right) \quad \text{combining the sums} \\
&= \mathbf{x}_i^\top \mathbf{x}_i - 2\sum_{j=1}^{k} z_{ij}\mathbf{v}_j^\top \mathbf{x}_i + \sum_{j=1}^{k} \mathbf{v}_j^\top z_{ij}^\top z_{ij}\mathbf{v}_j \quad \text{since } \mathbf{v}_i^\top \mathbf{v}_j = 1 \iff i = j \\
&= \mathbf{x}_i^\top \mathbf{x}_i - 2\sum_{j=1}^{k} z_{ij}\mathbf{v}_j^\top \mathbf{x}_i + \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{v}_j \quad \text{since } z_{ij}^\top z_{ij} = \mathbf{x}_i \mathbf{x}_i^\top \\
&= \mathbf{x}_i^\top \mathbf{x}_i - 2\sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i z_{ij} + \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{v}_j \quad \text{since } z_{ij} \text{ is a constant} \\
&= \mathbf{x}_i^\top \mathbf{x}_i - 2\sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{v}_j + \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{v}_j \quad \text{since } z_{ij} = \mathbf{x}_i^\top \mathbf{v}_j \\
&= \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{v}_j
\end{aligned}
$$

as desired.

(b) Proof:

$$
\begin{aligned}
J_k &= \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{v}_j\right) \quad \text{by definition} \\
&= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \frac{1}{n}\left(\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top\right) \mathbf{v}_j \quad \text{splitting the sums} \\
&= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \mathbf{v}_j^\top \Sigma \mathbf{v}_j \quad \text{by definition of } \Sigma \\
&= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \lambda_j \quad \text{by definition of } \lambda_j
\end{aligned}
$$

as desired.

(c) Recall that $J_d = 0$. So

$$\begin{aligned}
\sum_{j=1}^{d} \lambda_j &= \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i \quad \text{from part (a)} \\
J_k &= \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{k} \lambda_j \quad \text{from part (b)} \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i - \sum_{j=1}^{d} \lambda_j + \sum_{j=k+1}^{d} \lambda_j \quad \text{since } \sum_{j=1}^{d} \lambda_j = \sum_{j=1}^{k} \lambda_j + \sum_{j=k+1}^{d} \lambda_j \\
&= \sum_{j=k+1}^{d} \lambda_j
\end{aligned}$$

as desired.