HW2

HW2 is required to finish the following two tasks based on this incomplete Notebook file. The final submission should include the completed code snippets, obtained results, and the given answers. You can get the PDF submission by "File -> Download as -> HTML," then open your downloaded HTML file via a browser and print the webpage as a PDF file.

- **Q1:** Multivariate linear regression (60 pts)
  1. Given a dataset, implement a multivariate linear regression model and show the training/validation/testing MSE errors (20pts).
  2. Implement a Ridge regression based model on step 1 (10 pts)
  3. Tune the hyperparameter α of Ridge regression on the validation set. Plot training/validation MSE errors over the given range. (20 pts)
  4. Choose one hyperparameter from step 3. Given the same hyperparameter, compare your Ridge regression model and the Sklearn-Ridge regression on the testing set. (10 pts)

- **Q2:** K-means clustering (40 pts)
  1. Given a K-means clustering model, run it on the iris dataset 20 times. Report the average NMI value along with its std (Normalized Mutual Information (NMI) is a standard clustering evaluation method. Basically, a higher NMI value indicates better clustering performance. More details could be referred to usage here and this introduction). Save the best clustering result by NMI for the following steps. (10 pts)
  2. Repeat step 1 for the given spectral clustering and hierarchical clustering models. Compare the clustering performance of these three methods by averaged NMI$\pm$std. (10 pts)
  3. Plot the best and worst clustering results of K-means by NMI. (20 pts)

Q1: Multivariate linear regression

We start from a given dataset, digits, which has $N = 1797$ samples with $D = 64$ features. The digits dataset is generally used for a classification task. Here, however, we tailor it as a multivariate regression problem: we aim to predict the last 8 features upon the given 56 features, resulting in $\mathbf{X} \in \mathbb{R}^{1797\times56}$ and $\mathbf{Y} \in \mathbb{R}^{1797\times8}$. Linear/Ridge regression models learn $\mathbf{W} \in \mathbb{R}^{56\times8} : \mathbf{X} \to \mathbf{Y}$.

Hint: You may refer to the implementation details in "Camino -> Modules -> Notebook -> Linear Regression" for solving the Q1. Think about the difference between (single value) linear regression and multivariate linear regression.

In [53]:
```
%matplotlib inline

import numpy as np
from sklearn import linear_model
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

# We first split the dataset into training/vaidation/test
# load dataset
digits = load_digits()
# normalzied features to range [0,1]
data = digits.data / 16.
N, _ = data.shape
D = 56 # pre-defined feature dimension
X = data[:, :D]
Y = data[:, D:]
# random shuffle the data
np.random.seed(0)
ind = np.random.permutation(N)
# split the data as 0.6/0.2/0.2
n1, n2 = int(0.6*N), int(0.8*N)
X_trn, Y_trn = X[ind[:n1]], Y[ind[:n1]]
X_val, Y_val = X[ind[n1:n2]], Y[ind[n1:n2]]
X_tst, Y_tst = X[ind[n2:]], Y[ind[n2:]]
print(X_trn.shape)
print(Y_trn.shape)
print(ind)
```

```
(1078, 56)
(1078, 8)
[1081 1707  927 ... 1653  559  684]
```

In [54]:

```
def mse(y_hat, y):
    return ((y_hat-y)**2).sum()/len(y)
```

Q1-1: Implement a multivariate linear regression model (20 pts)

$\mathcal{L}_w = \frac{1}{2}\|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_{\mathrm{F}}^2$, $\mathbf{W}^* = \mathtt{argmin}_{\mathbf{W}}\mathcal{L}_w = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$

For a given matrix $\mathbf{A}$, $\|\mathbf{A}\|_{\mathrm{F}}$ is defined as $\|\mathbf{A}\|_{\mathrm{F}} = \sqrt{\sum_i \sum_j \mathbf{A}_{ij}^2}$. You may refer to Matrix norm for more details about the vector norm and matrix norm.

Note that, it is common to raise a "singular matrix error" when computing $(\mathbf{X}^T\mathbf{X})^{-1}$. One common trick is to compute $(\mathbf{X}^T\mathbf{X} + \epsilon\mathbf{I})^{-1}$ instead, where ϵ could be set as $1e-8$.

In [81]:

```
# TODO: get the predictions via your linear regression model

# w_l = ? # @ is equivilent to np.matmul -> matrix multiplication
# w_l = np.linalg.inv(np.transpose(X_trn) @ X_trn) @ np.transpose(X_trn) @ Y_trn
w_l = np.linalg.inv(np.transpose(X_trn) @ X_trn + 1e-8*np.identity(X_trn.shape[1
print(w_l.shape)

# y_trn_hat = ?
y_trn_hat = X_trn @ w_l
# y_val_hat = ?
y_val_hat = X_val @ w_l
# y_tst_hat = ?
y_tst_hat = X_tst @ w_l

# print the training/validation/testing MSE error by using the mse function
print('Training MSE:\t %.4f' % mse(y_trn_hat, Y_trn))
print('Validation MSE:\t %.4f' % mse(y_val_hat, Y_val))
print('Testing MSE:\t %.4f' % mse(y_tst_hat, Y_tst))
```

```
(56, 8)
Training MSE:    0.1477
Validation MSE:  0.1606
Testing MSE:     0.1661
```

Q1-2: Implement a multivariate Ridge regression model (10 pts)

For Ridge regression, we have $\mathcal{L}_w = \frac{1}{2}(\|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_F^2 + \alpha\|\mathbf{W}\|_F^2)$, $\mathbf{W}^* = \texttt{argmin}_{\mathbf{W}}\mathcal{L}_w = (\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$, where $\alpha > 0$ is a trade-off hyperparameter balancing regression error and ℓ_2 regularization.

In [82]:
```
# Todo: give the W solution for ridge regression

alpha = 0.1
# w_r = ?
w_r = np.linalg.inv(np.transpose(X_trn) @ X_trn + alpha*np.identity(X_trn.shape[
print(w_r.shape)

# y_trn_hat = ?
y_trn_hat = X_trn @ w_r
# y_val_hat = ?
y_val_hat = X_val @ w_r
# y_tst_hat = ?
y_tst_hat = X_tst @ w_r

# print the training/validation/testing MSE error by using the mse function
print('Training MSE:\t %.4f' % mse(y_trn_hat, Y_trn))
print('Validation MSE:\t %.4f' % mse(y_val_hat, Y_val))
print('Testing MSE:\t %.4f' % mse(y_tst_hat, Y_tst))
```

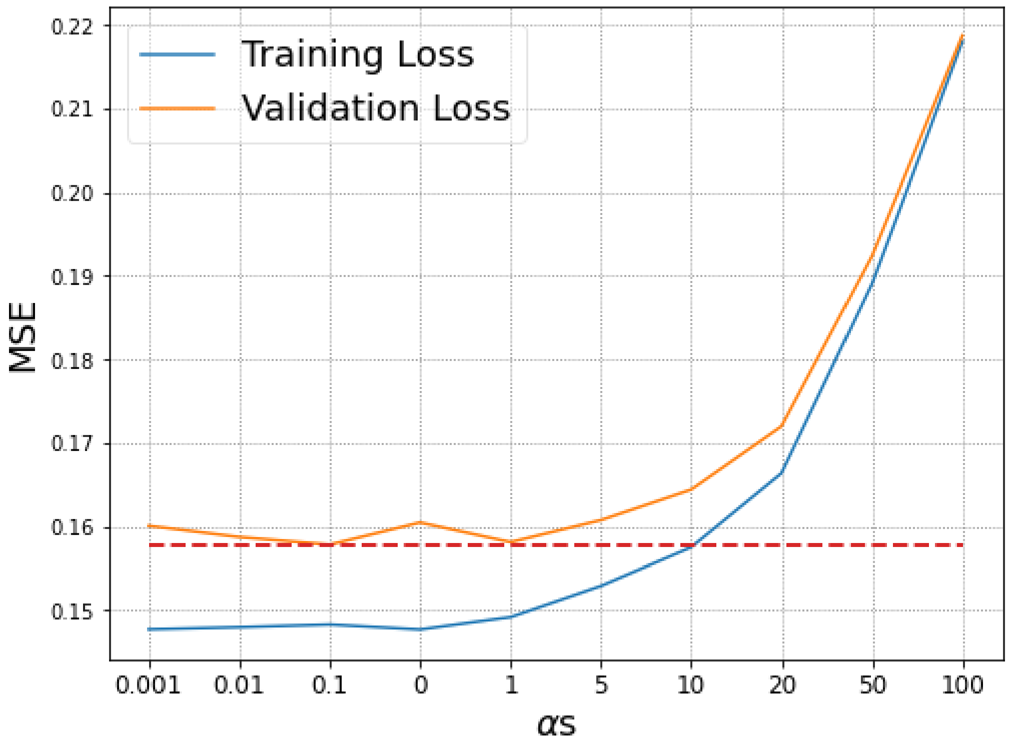
```
(56, 8)
Training MSE:    0.1483
Validation MSE:  0.1578
Testing MSE:     0.1646
```

Q1-3: Tune the hyperparameter α on the validation set (20 pts)

We may utilize the Sklearn implemntation for tuning the hyperparameter of a ridge regression model.

In [90]:
```
alphas = [0.001, 0.01, 0.1, 0, 1, 5, 10, 20, 50, 100]
loss_trns = []
loss_vals = []
for alpha in alphas:
    # instantiate a ridge regression model with a given alpha
    model = linear_model.Ridge(alpha=alpha, fit_intercept=False)
    # TODO: training the model and get the training and validation loss for each
    # train model?
    model.fit(X_trn, Y_trn)
    # loss_trns.append(?)
    loss_trns.append(mse(model.predict(X_trn), Y_trn))
    # loss_vals.append(?)
    loss_vals.append(mse(model.predict(X_val), Y_val))

fig, ax = plt.subplots(figsize=(8,6))
plt.grid(which='both',axis='both', color='grey',linestyle=':')
```

```
# TODO plot the training and validation loss below
# plt.plot(training MSE?)
plt.plot(loss_trns, label = 'Training Loss')
# plt.plot(validation MSE?)
plt.plot(loss_vals, label = 'Validation Loss')

# TODO how to plot the best validation loss?
# plt.plot(np.arange(len(alphas)), ?, color='tab:red', linestyle='--', linewidth
# For example, if you want to hilight one horizontal line.
plt.plot(np.arange(len(alphas)), np.ones(len(alphas))*np.min(loss_vals), color='

plt.ylabel('MSE', fontsize=18)
plt.xlabel(r'${\alpha}$s', fontsize=18)
plt.xticks(np.arange(len(alphas)), alphas, fontsize=12)
# TODO: add a legend
plt.legend(fancybox=True, framealpha=0.5, fontsize=18)
plt.show()
```



Q1-4: Compare your ridge regression model with Sklearn implmentation

In [95]:
```
# TODO: select alpha from Q1-3
# alpha = ?
# Given w_r from Q1-2

# y_r_trn = ?
# y_r_val = ?
# y_r_tst = ?
# print the training/validation/testing MSE error by using the mse function

alpha = 0.1
# w_r = ?
w_r = np.linalg.inv(np.transpose(X_trn) @ X_trn + alpha*np.identity(X_trn.shape[
print(w_r.shape)
```

```
# y_trn_hat = ?
y_trn_hat = X_trn @ w_r
# y_val_hat = ?
y_val_hat = X_val @ w_r
# y_tst_hat = ?
y_tst_hat = X_tst @ w_r

# print the training/validation/testing MSE error by using the mse function
print('My Model Summary:')
print('Training MSE:\t %.4f' % mse(y_trn_hat, Y_trn))
print('Validation MSE:\t %.4f' % mse(y_val_hat, Y_val))
print('Testing MSE:\t %.4f' % mse(y_tst_hat, Y_tst))

# sklearn mdoel with the same alpha
model = linear_model.Ridge(alpha=0.1, fit_intercept=False)
model.fit(X_trn, Y_trn)
# train the model
# print the training/validation/testing MSE error by using the mse function
print('Sklearn Model Summary:')
print('Training MSE:\t %.4f' % mse(model.predict(X_trn), Y_trn))
print('Validation MSE:\t %.4f' % mse(model.predict(X_val), Y_val))
print('Testing MSE:\t %.4f' % mse(model.predict(X_tst), Y_tst))
```

```
(56, 8)
My Model Summary:
Training MSE:    0.1483
Validation MSE:  0.1578
Testing MSE:     0.1646
Sklearn Model Summary:
Training MSE:    0.1483
Validation MSE:  0.1578
Testing MSE:     0.1646
```

Q2: K-means clustering

We will directly use the Sklearn libary for running K-means, spectral clustering, and hierarchical clustering. More details about the function usage could be referred to Sklearn cluster and this tutorial.

In [24]:

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering
from sklearn.metrics.cluster import normalized_mutual_info_score as nmi_score
import numpy as np

dataset = load_iris()
X = dataset.data
gnd = dataset.target
semantic_labels = dataset.target_names
print(semantic_labels)
print(gnd)
K = len(set(gnd)) # get the ground-truth cluster number
```

```
['setosa' 'versicolor' 'virginica']
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Q2-1: K-means (10 pts)

Given a K-means model, run it 20 times to report averaged NMI and std value. Save the clustering labels of each trial for the following steps.

In [50]:
```
km = KMeans(n_clusters=K, init='random', n_init=1)
idx, nmis = [], []
for i in range(20):
    # run K-means here
    km.fit(X)
    # idx.append(the obtained clustering labels)
    idx.append(km.labels_)
    # nmi.append(nmi_score(gnd, clustering label))
    nmis.append(nmi_score(gnd, km.labels_))

#     pass # delete this statement once you complete this snippet.
# report mean and std values of nmis
print('Mean of nmis using k-means is: ', np.mean(nmis))
print('Std of nmis using k-means is: ', np.std(nmis))
```

```
Mean of nmis using k-means is:  0.6858669044421231
Std of nmis using k-means is:  0.08050069843356983
```

Q2-2: Compare K-means with SC and AC (10 pts)

Repeat Q2-1 with the following two given models.

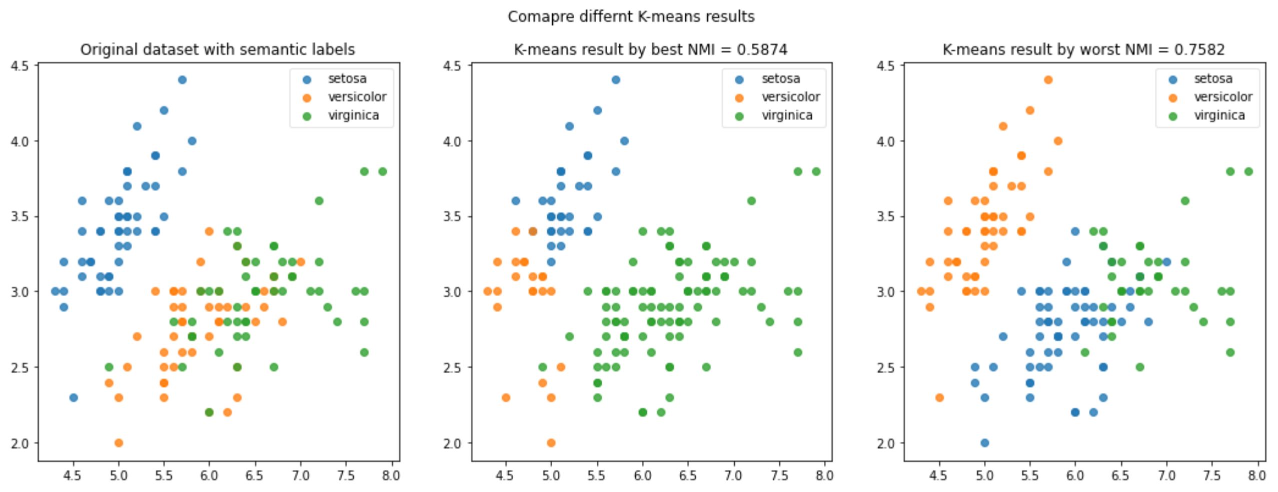
In [51]:
```
sc = SpectralClustering(n_clusters=K, n_init=1) # spectral clustering model
ac = AgglomerativeClustering(n_clusters=K) # hierarchical clustering model
# TODO: repeat Q2-1 and compare km with sc and ac
# print the comparion results
idx_SC, nmis_SC = [], []
idx_AC, nmis_AC = [], []

for i in range(20):
    # run SC and AC here
    sc.fit(X)
    idx_SC.append(sc.labels_)
    nmis_SC.append(nmi_score(gnd, sc.labels_))
    ac.fit(X)
    idx_AC.append(ac.labels_)
    nmis_AC.append(nmi_score(gnd, ac.labels_))

#     pass # delete this statement once you complete this snippet.
# report mean and std values of nmis
print('Mean and std of nmis using k-means is: \t', np.mean(nmis),'\t', np.std(nm
print('Mean and std of nmis using SC is: \t', np.mean(nmis_SC), '\t',np.std(nmis
print('Mean and std of nmis using AC is: \t', np.mean(nmis_AC), '\t',np.std(nmis
```

```
Mean and std of nmis using k-means is:   0.6858669044421231      0.0805006984335
6983
Mean and std of nmis using SC is:        0.7979885217013318      1.1102230246251
565e-16
Mean and std of nmis using AC is:        0.770083661648787       0.0
```

Q2-3: Plot K-means clustering results (20 pts)

In [52]:
```
import seaborn as sns
# sns.set_theme()
```

```
clustering_labels = ['Cluster 1', 'Cluster 2', 'Cluster 3']

# TODO:

# 1) find the best/worst clustering labels by NMI, termed as bst_idx and wst_idx
bst_idx = idx[np.argmin(nmis)]
wst_idx = idx[np.argmax(nmis)]

# 2) scatter plot the data samples upon the clustering labels, bst_idx and wst_i
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18,6))
fig.suptitle('Comapre differnt K-means results')
colors = ['tab:blue', 'tab:orange', 'tab:green']
ax1.set_title('Original dataset with semantic labels')
# fill out the NMI values in the title
ax2.set_title('K-means result by best NMI = ' + str(np.round(min(nmis),4)))
ax3.set_title('K-means result by worst NMI = '  + str(np.round(max(nmis),4)))

for color, i in zip(colors, [0, 1, 2]):
    ax1.scatter(X[gnd == i, 0], X[gnd == i, 1], color=color, alpha=.8, label=sem
    # ax2.scatter(?) plot with bst_idx and clsutering_labels
    ax2.scatter(X[bst_idx == i, 0], X[bst_idx == i, 1], color=color, alpha=.8, l
    # ax3.scatter(?) plot with wst_idx and clsutering_labels
    ax3.scatter(X[wst_idx == i, 0], X[wst_idx == i, 1], color=color, alpha=.8, l

ax1.legend(fancybox=True, framealpha=0.5)

# TODO: add legends for ax2 and ax3
ax2.legend(fancybox=True, framealpha=0.5)
ax3.legend(fancybox=True, framealpha=0.5)
plt.show()
```



Answer the following questions:

- Can we use the semantic_labels for the clustering results plot? Why?
- Why does the same clustering label (e.g., clustering 1) have different groups of samples between two K-means results?

Please directly answer the above question with Markdown inside this notebook file.

Answer:

- No, we cannot use the semantic_labels for clustering results plot. Because clustering is unsupervised learning. So it don't have semantic_labels
- Because it is unsupervised learning. So it don't have the label. So we could just randomly generate numbers.