Summer 2022 CS4641/CS7641 A Homework 3

Instructor: Dr. Mahdi Roozbahani

Deadline: Wednesday, July 13, 11:59 pm AOE

- No unapproved extension of the deadline is allowed. Late submission will lead to 0 credit.
- Discussion is encouraged on Ed as part of the Q/A. However, all assignments should be done individually.
- Plagiarism is a **serious offense**. You are responsible for completing your own work. You are not allowed to copy and paste, or paraphrase, or submit materials created or published by others, as if you created the materials. All materials submitted must be your own.
- All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures. If we observe any (even small) similarities/plagiarisms detected by Gradescope or our TAs, **WE WILL DIRECTLY REPORT ALL CASES TO OSI**, which may, unfortunately, lead to a very harsh outcome. **Consequences can be severe, e.g., academic probation or dismissal, grade penalties, a 0 grade for assignments concerned, and prohibition from withdrawing from the class.**

Instructions for the assignment

- This assignment consists of both programming and theory questions.
- Unless a theory question explicitly states that no work is required to be shown, you must provide an explanation, justification, or calculation for your answer.
- To switch between cell for code and for markdown, see the menu -> Cell -> Cell Type
- You can directly type Latex equations into markdown cells.
- If a question requires a picture, you could use this syntax `<img src="" style="width: 300px;"/>` to include them within your ipython notebook.
- Your write up must be submitted in PDF form. You may use either Latex, markdown, or any word processing software. We will **NOT** accept handwritten work. Make sure that your work is formatted correctly, for example submit $\sum_{i=0} x_i$ instead of \text{sum_{i=0} x_i}
- When submitting the non-programming part of your assignment, you must correctly map pages of your PDF to each question/subquestion to reflect where they appear. **Improperly mapped questions may not be graded correctly and/or will result in point deductions for the error.**

- All assignments should be done individually, and each student must write up and submit their own answers.
- **Graduate Students**: You are required to complete any sections marked as Bonus for Undergrads

Using the autograder

- Grads will find three assignments on Gradescope that correspond to HW3: "Assignment 3 Programming", "Assignment 3 - Non-programming" and "Assignment 3 Programming - Bonus for all". Undergrads will find an additional assignment called "Assignment 3 Programming - Bonus for Undergrads".
- You will submit your code for the autograder in the Programming sections. Please refer to the Deliverables and Point Distribution section for what parts are considered required, bonus for undergrads, and bonus for all.

- We provided you different .py files and we added libraries in those files please DO NOT remove those lines and add your code after those lines. Note that these are the only allowed libraries that you can use for the homework.

- You are allowed to make as many submissions until the deadline as you like. Additionally, note that the autograder tests each function separately, therefore it can serve as a useful tool to help you debug your code if you are not sure of what part of your implementation might have an issue.

- **For the "Assignment 3 - Non-programming" part, you will download your Jupyter Notebook as HTML and submit it as a PDF on Gradescope. To download the notebook as HTML, click on "File" on the top left corner of this page and select "Download as > HTML". Then, open the HTML file and print to PDF.** Please refer to the **Deliverables and Point Distribution** section for an outline of the non-programming questions.

- **When submitting to Gradescope, please make sure to mark the page(s) corresponding to each problem/sub-problem. The pages in the PDF should be of size 8.5" x 11", otherwise there may be a deduction in points for extra long sheets.**

Using the local tests

- For some of the programming questions we have included a local test using a small toy dataset to aid in debugging. The local test sample data and outputs are stored in .py files in the **local_tests_folder**
- There are no points associated with passing or failing the local tests, you must still pass the autograder to get points.
- **It is possible to fail the local test and pass the autograder** since the autograder has a certain allowed error tolerance while the local test allowed error may be smaller. Likewise, passing the local tests does not guarantee passing the autograder.

- **You do not need to pass both local and autograder tests to get points, passing the Gradescope autograder is sufficient for credit.**
- It might be helpful to comment out the tests for functions that have not been completed yet.
- It is recommended to test the functions as it gets completed instead of completing the whole class and then testing. This may help in isolating errors. Do not solely rely on the local tests, continue to test on the autograder regularly as well.

Deliverables and Points Distribution

Q1: Image Compression [30pts]

Deliverables: imgcompression.py and printed results

- **1.1 Image Compression** [20 pts] - *programming*
  - svd [5pts]
  - rebuild_svd [5pts]
  - compression_ratio [5pts]
  - recovered_variance_proportion [5pts]
- **1.2 Black and White** [5 pts] *non-programming*
- **1.3 Color Image** [5 pts] *non-programming*

Q2: Understanding PCA [20pts]

Deliverables: pca.py and written portion

- **2.1 PCA Implementation** [10 pts] - *programming*
  - fit [5pts]
  - transform [2pts]
  - transform_rv [3pts]
- **2.2 Visualize** [5 pts] *non-programming*
- **2.3 PCA Reduced Mask Dataset Analysis** [5 pts] *non-programming*

Q3: Regression and Regularization [50pts + 20pts Bonus for Undergrads + 10pts Bonus for All]

Deliverables: regression.py and Written portion

- **3.1 Regression and Regularization Implementations** [30 pts + 20 pts Bonus for Undergrad] - *programming*
  - RMSE [5pts]
  - Construct Poly Features 1D [2pts]
  - Construct Poly Features 2D [3pts]
  - Prediction [5pts]
  - Linear Fit Closed Form [5pts]
  - Ridge Fit Closed Form [5pts]
  - Cross Validation [5pts]
  - Linear Gradient Descent [5pts] **Bonus for Undergrad**
  - Linear Stochastic Gradient Descent [5pts] **Bonus for Undergrad**
  - Ridge Gradient Descent [5pts] **Bonus for Undergrad**
  - Ridge Stochastic Gradient Descent [5pts] **Bonus for Undergrad**
- **3.2 About RMSE** [3 pts] *non-programming*
- **3.3 Testing: General Functions and Linear Regression** [5 pts] *non-programming*
- **3.4 Testing: Ridge Regression** [5 pts] *non-programming*
- **3.5 Cross Validation** [7 pts] *non-programming*
- **3.6 Noisy Input Samples in Linear Regression** [10 pts] *non-programming* **BONUS FOR ALL**

Q4: Naive Bayes Classification [25pts]

Deliverables: nb.py and Written portion

- **4.1 Llama Breed Problem** [5 pts] *non-programming*
- **4.2 News Data Sentiment Classification** [15 pts] - *programming*
  - priors_prob [6pts]
  - likelihood_ratio [6pts]
  - analyze_star_rating [3pts]
- **4.3 Accuracy result analysis** [5 pts] *non-programming*

Q5: Noise in PCA and Linear Regression [15pts Bonus for All]

Deliverables: Written portion

- **5.1 Slope Functions** [5 pts] *non-programming*
- **5.2 Error in Y and Error in X and Y** [5 pts] *non-programming*
- **5.3 Analysis** [5 pts] *non-programming*

Q6: Feature Reduction.py [25pts Bonus for All]

Deliverables: feature_reduction.py and Written portion

- **6.1 Feature Reduction** [18 pts] - *programming*
  - forward_selection [9pts]
  - backward_elimination [9pts]
- **6.2 Feature Selection - Discussion** [7 pts] *non-programming*

0 Set up

This notebook is tested under python 3. . , and the corresponding packages can be downloaded from miniconda. You may also want to get yourself familiar with several packages:

- jupyter notebook
- numpy
- matplotlib
- sklearn
- Axes3D

There is also a VS Code and Anaconda Setup Tutorial on Ed under the "Links" category

Please implement the functions that have `raise NotImplementedError` , and after you finish the coding, please delete or comment out `raise NotImplementedError` .

Library imports

In [1]:
```
###############################
### DO NOT CHANGE THIS CELL ###
###############################
# This is cell which sets up some of the modules you might need
# Please do not change the cell or import any additional packages.

import numpy as np
import pandas as pd
import json
import math
import matplotlib
```

```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.feature_extraction import text
from sklearn.datasets import load_boston, load_diabetes, load_digits, load_breas
from sklearn.linear_model import Ridge, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, accuracy_score
import warnings
import sys
import re
import gzip
from tqdm.notebook import tqdm

print('Version information')

print('python: {}'.format(sys.version))
print('matplotlib: {}'.format(matplotlib.__version__))
print('numpy: {}'.format(np.__version__))

warnings.filterwarnings('ignore')

%matplotlib inline
%load_ext autoreload
%autoreload 2

STUDENT_VERSION = 1
EO_TEXT, EO_FONT, EO_COLOR = 'TA VERSION', 'Arial', 'gray',
EO_ALPHA, EO_SIZE, EO_ROT = 0.5, 90, 40
```

```
Version information
python: 3.8.5 (default, Sep  4 2020, 02:22:02)
[Clang 10.0.0 ]
matplotlib: 3.3.2
numpy: 1.23.0
```

Q1: Image Compression [30 pts]

Load images data and plot

In [2]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
# load Image
image = plt.imread("./data/HW3_image_compression.jpg")/255
#plot image
fig = plt.figure(figsize=(10,10))
if not STUDENT_VERSION:
    fig.text(0.5, 0.5, EO_TEXT, transform=fig.transFigure,
        fontsize=EO_SIZE, color=EO_COLOR, alpha=EO_ALPHA, fontname=EO_FONT,
        ha='center', va='center', rotation=EO_ROT)
plt.imshow(image)
```

Out[2]:
```
<matplotlib.image.AxesImage at 0x7fd87ea126d0>
```

In [3]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

fig = plt.figure(figsize=(10, 10))

if not STUDENT_VERSION:
    fig.text(0.5, 0.5, EO_TEXT, transform=fig.transFigure,
        fontsize=EO_SIZE, color=EO_COLOR, alpha=EO_ALPHA, fontname=EO_FONT,
        ha='center', va='center', rotation=EO_ROT)
# plot several images
plt.imshow(rgb2gray(image), cmap=plt.cm.bone)
```

Out[3]:
```
<matplotlib.image.AxesImage at 0x7fd8906cde50>
```

1.1 Image compression [20pts] **[P]**

SVD is a dimensionality reduction technique that allows us to compress images by throwing away the least important information.

Higher singular values capture greater variance and, thus, capture greater information from the corresponding singular vector. To perform image compression, apply SVD on each matrix and get rid of the small singular values to compress the image. The loss of information through this process is negligible, and the difference between the images can be hardly spotted.

For example, the proportion of variance captured by the first component is

$$\frac{\sigma_1^2}{\sum_{i=1}^{n} \sigma_i^2}$$

where σ_i is the i^{th} singular value.

In the **imgcompression.py** file, complete the following functions:

- **svd**: You may use `np.linalg.svd` in this function, and although the function defaults this parameter to true, you may explicitly set `full_matrices=True` using the optional `full_matrices` parameter. Hint 2 may be useful.
- **rebuild_svd**
- **compression_ratio**: Hint 1 may be useful
- **recovered_variance_proportion**: Hint 1 may be useful

Hint 1: http://timbaumann.info/svd-image-compression-demo/ is a useful article on image compression and compression ratio. You may find this article useful for implementing the functions compression_ratio and recovered_variance_proportion

Hint 2: If you have never used `np.linalg.svd`, it might be helpful to read Numpy's SVD documentation and note the particularities of the V matrix and that it is returned already transposed.

Hint 3: The shape of S resulting from SVD may change depending on if N>D, N<D, or N=D. Therefore, when checking the shape of S, note that min(N,D) means the value should be equal to whichever is lower between N and D.

1.1.1 Local Tests for Image Compression Black and White Case [No Points]

You may test your implementation of the functions contained in **imgcompression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

In [6]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################

from imgcompression import ImgCompression
from local_tests_folder.ic_test import IC_Test

test_ic = IC_Test()
imc = ImgCompression()

print('Local Tests for Black and White (Grayscale) Case \n')
# Local test for svd
U, S, V = imc.svd(test_ic.bw_image)
svdg_test = np.allclose(U, test_ic.Ug) and np.allclose(S, test_ic.Sg) and np.all
print('Your svd works within the expected range:', svdg_test)

# Local test for rebuild_svd
Xrebuild_g = imc.rebuild_svd(test_ic.Ug, test_ic.Sg, test_ic.Vg, 2)
rsvdg_test = np.allclose(Xrebuild_g, test_ic.Xrebuild_g)
print('Your rebuild_svd works within the expected range:', rsvdg_test)

# Local test for compression ratio
crg = imc.compression_ratio(test_ic.bw_image, 2)
crg_test = np.allclose(crg, test_ic.cr_g)
print('Your compression_ratio works within the expected range:', crg_test )

# Local test for recovered variance proportion
rvpg = imc.recovered_variance_proportion(test_ic.Sg, 2)
rvpg_test = np.allclose(rvpg, test_ic.rvp_g)
print('Your recovered_variance_proportion works within the expected range:', rvp
```

```
Local Tests for Black and White (Grayscale) Case

Your svd works within the expected range: True
Your rebuild_svd works within the expected range: True
Your compression_ratio works within the expected range: True
Your recovered_variance_proportion works within the expected range: True
```

1.1.2 Local Tests for Image Compression Color Case [No Points]

You may test your implementation of the functions contained in **imgcompression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

In [7]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from imgcompression import ImgCompression
from local_tests_folder.ic_test import IC_Test

test_ic = IC_Test()
imc = ImgCompression()

print('Local Tests for Color Case \n')
# Local test for svd
U, S, V = imc.svd(test_ic.color_image)
svdc_test = np.allclose(U, test_ic.Uc) and np.allclose(S, test_ic.Sc) and np.all
print('Your svd works within the expected range:', svdc_test)

# Local test for rebuild_svd
Xrebuild_c = imc.rebuild_svd(test_ic.Uc, test_ic.Sc, test_ic.Vc, 2)
rsvdc_test = np.allclose(Xrebuild_c, test_ic.Xrebuild_c)
print('Your rebuild_svd works within the expected range:', rsvdc_test)

# Local test for compression ratio
crc = imc.compression_ratio(test_ic.color_image, 2)
crc_test = np.allclose(crc, test_ic.cr_c)
print('Your compression_ratio works within the expected range:', crc_test)

# Local test for recovered variance proportion
rvpc = imc.recovered_variance_proportion(test_ic.Sc, 2)
rvpc_test = np.allclose(rvpc, test_ic.rvp_c)
print('Your recovered_variance_proportion works within the expected range:', rvp
```

```
Local Tests for Color Case

Your svd works within the expected range: True
Your rebuild_svd works within the expected range: True
Your compression_ratio works within the expected range: True
Your recovered_variance_proportion works within the expected range: True
```

1.2 Black and white [5 pts] **[W]**

This question will use your implementation of the functions from Q1.1 to generate a set of images compressed to different degrees. You can simply run the below cell without making any changes to it, assuming you have implemented the functions in Q1.1.

Make sure these images are displayed when submitting the PDF version of the Juypter notebook as part of the non-programming submission of this assignment.

In [8]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
from imgcompression import ImgCompression
```

```
imcompression = ImgCompression()
bw_image = rgb2gray(image)
U, S, V = imcompression.svd(bw_image)
component_num = [1,2,5,10,20,40,80,160,256]

fig = plt.figure(figsize=(18, 18))

# plot several images
i=0
for k in component_num:
    img_rebuild = imcompression.rebuild_svd(U, S, V, k)
    c = np.around(imcompression.compression_ratio(bw_image, k), 4)
    r = np.around(imcompression.recovered_variance_proportion(S, k), 3)
    ax = fig.add_subplot(3, 3, i + 1, xticks=[], yticks=[])
    ax.imshow(img_rebuild, cmap=plt.cm.bone)
    ax.set_title(f"{k} Components")
    if not STUDENT_VERSION:
        ax.text(0.5, 0.5, EO_TEXT, transform=ax.transAxes,
            fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA, fontname=EO_FONT
            ha='center', va='center', rotation=EO_ROT)
    ax.set_xlabel(f"Compression: {c},\nRecovered Variance: {r}")
    i = i+1
```

1.3 Color image [5 pts] **[W]**

This section will use your implementation of the functions from Q1.1 to generate a set of images compressed to different degrees. You can simply run the below cell without making any changes to it, assuming you have implemented the functions in Q1.1.

Make sure these images are displayed when submitting the PDF version of the Juypter notebook as part of the non-programming submission of this assignment.

Note: You might get warning "Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)." This warning is acceptable since some of the pixels may go above 1.0 while rebuilding. You should see similar images to original even with such clipping.

Hint 1: Make sure your implementation of `recovered_variance_proportion` returns an array of 3 floats for a color image.
Hint 2: Try performing SVD on the individual color channels and then stack the individual channel U, S, V matrices.
Hint 3: You may need separate implementations for a color or grayscale image in the same function.

In [9]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
from imgcompression import ImgCompression

imcompression = ImgCompression()
U, S, V = imcompression.svd(image)

# component_num = [1,2,5,10,20,40,80,160,256]
component_num = [1,2,5,10,20,40,80,160,256]

fig = plt.figure(figsize=(18, 18))

# plot several images
i=0
for k in component_num:
    img_rebuild = np.clip(imcompression.rebuild_svd(U, S, V, k),0,1)
    c = np.around(imcompression.compression_ratio(image, k), 4)
    r = np.around(imcompression.recovered_variance_proportion(S, k), 3)
    ax = fig.add_subplot(3, 3, i + 1, xticks=[], yticks=[])
    ax.imshow(img_rebuild)
    ax.set_title(f"{k} Components")
    if not STUDENT_VERSION:
        ax.text(0.5, 0.5, EO_TEXT, transform=ax.transAxes,
            fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA, fontname=EO_FONT
            ha='center', va='center', rotation=EO_ROT)
    ax.set_xlabel(f"Compression: {np.around(c,4)},\nRecovered Variance:  R: {r[0
    i = i+1
```

Q2: Understanding PCA [20 pts]

2.1 Implementation [10 pts] **[P]**

Principal Component Analysis (PCA) is another dimensionality reduction technique that reduces dimensions by eliminating small variance eigenvalues and their vectors. With PCA, we center the data first by subtracting the mean. Each singular value tells us how much of the variance of a matrix (e.g. image) is captured in each component. In this problem, we will investigate how PCA can be used to improve features for regression and classification tasks and how the data itself affects the behavior of PCA.

Implement PCA. In the **pca.py** file, complete the following functions:

- **fit**: You may use `np.linalg.svd`. Set `full_matrices=False`. Hint 1 may be useful.
- **transform**
- **transform_rv**: You may find `np.cumsum` helpful for this function.

Assume a dataset is composed of N datapoints, each of which has D features with D < N. The dimension of our data would be D. However, it is possible that many of these dimensions contain redundant information. Each feature explains part of the variance in our dataset, and some features may explain more variance than others.

Hint 1: Make sure you remember to first center your data by subtracting the mean.

2.1.1 Local Tests for PCA [No Points]

You may test your implementation of the functions contained in **pca.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

In [20]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from pca import PCA
from local_tests_folder.pca_test import PCA_Test

test_pca = PCA_Test()
pca = PCA()

print('Local Tests PCA \n')

# Local test for fit
pca.fit(test_pca.data)
U, S, V = pca.U, pca.S, pca.V
fit_test = np.allclose(U, test_pca.U) and np.allclose(S, test_pca.S) and np.allc
print('Your fit works within the expected range:', fit_test)

# Local test for transform
X_new = pca.transform(test_pca.data, 2)
transform_test = np.allclose(X_new, test_pca.X_new)
print('Your transform works within the expected range:', transform_test)

# Local test for transform_rv
X_new_rv = pca.transform_rv(test_pca.data, 0.7)
transform_rv_test = np.allclose(X_new_rv, test_pca.X_new_rv)
print('Your transform_rv works within the expected range:', transform_rv_test)
```

```
Local Tests PCA

Your fit works within the expected range: True
Your transform works within the expected range: True
Your transform_rv works within the expected range: True
```

2.2 Visualize [5 pts] **[W]**

PCA is used to transform multivariate data tables into smaller sets so as to observe the hidden trends and variations in the data. It can also be used as a feature extractor for images. Here you will visualize two datasets using PCA, first is the iris dataset and then a dataset of masked and unmasked images.

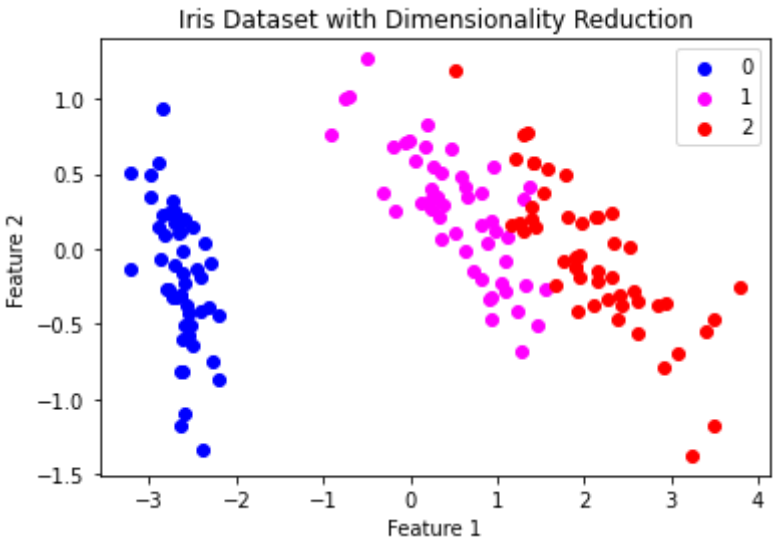
In the **pca.py**, complete the following function:

- **visualize**: Use your implementation of PCA and reduce the datasets such that they contain only two features. Using Matplotlib's Pyplot, create 2-D scatter plots of the data points using these features. Make sure to differentiate the data points according to their true labels using color.

The datasets have already been loaded for you in the subsequent cells.

Iris Dataset

In [21]:
```
###############################
### DO NOT CHANGE THIS CELL ###
###############################

# Use PCA for visualization of iris dataset

from pca import PCA

iris_data = load_iris(return_X_y=True)

X = iris_data[0]
y = iris_data[1]

fig = plt.figure()
plt.title('Iris Dataset with Dimensionality Reduction')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
PCA().visualize(X,y,fig)
```

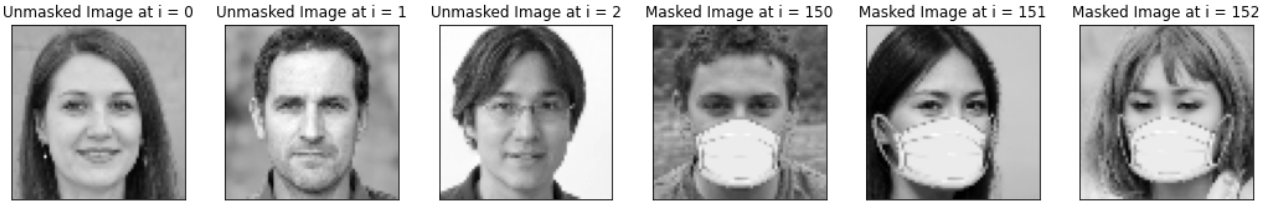


Facemask Dataset

The masked and unmasked dataset is made up of grayscale images of human faces facing forward. Half of these images are faces that are completely unmasked, and the remaining images show half of the face covered with an artificially generated face mask. The images have already been preprocessed, they are also reduced to a small size of 64x64 pixels and then reshaped into a feature vector of 4096 pixels. Below is a sample of some of the images in the dataset.

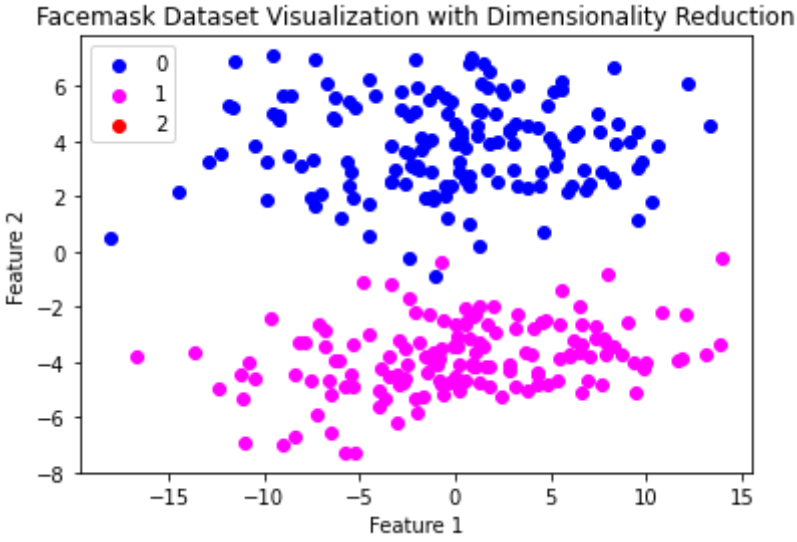
In [22]:
```
###############################
```

```
### DO NOT CHANGE THIS CELL ###
##############################

X = np.load('./data/smallflat_64.npy')
y = np.load('./data/masked_labels.npy').astype('int')
i = 0
fig = plt.figure(figsize=(18, 18))
for idx in [0,1,2,150,151,152]:
    ax = fig.add_subplot(6, 6, i + 1, xticks=[], yticks=[])
    ax.imshow(X[idx].reshape(64, 64), cmap = 'gray')
    m_status = 'Unmasked' if idx < 150 else 'Masked'
    ax.set_title(f"{m_status} Image at i = {idx}")
    i += 1
```



In [23]:

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
# Use PCA for visualization of masked and unmasked images

X = np.load('./data/smallflat_64.npy')
y = np.load('./data/masked_labels.npy')

fig = plt.figure()
plt.title('Facemask Dataset Visualization with Dimensionality Reduction')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
PCA().visualize(X,y, fig)
print('*In this plot, the 0 points are unmasked images and the 1 points are mask
```




```
*In this plot, the 0 points are unmasked images and the 1 points are masked imag
es.
```

What do you think of this 2 dimensional plot, knowing that the original dataset was originally a set of flattened image vectors that had 4096 pixels/features? No written answer necessary.

Now you will use PCA on an actual real-world dataset. We will use your implementation of PCA function to reduce the dataset with 99% retained variance and use it to obtain the reduced features. On the reduced dataset, we will use logistic and linear regression to compare results between PCA and non-PCA datasets. Run the following cells to see how PCA works on regression and classification tasks.

In [24]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
#load the dataset
digits = load_digits()

X = digits.data
y = digits.target

iris_data = load_iris(return_X_y=True)

X = iris_data[0]
y = iris_data[1]

print("data shape before PCA ",X.shape)

pca = PCA()
pca.fit(X)

X_pca = pca.transform_rv(X)

print("data shape with PCA ",X_pca.shape)
```

```
data shape before PCA  (150, 4)
data shape with PCA  (150, 3)
```

In [25]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
# Train, test splits
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3,
                                                    stratify=y,
                                                    random_state=42)

# Use logistic regression to predict classes for test set
clf = LogisticRegression()
clf.fit(X_train, y_train)
preds = clf.predict_proba(X_test)
print('Accuracy before PCA: {:.5f}'.format(accuracy_score(y_test,
                                                          preds.argmax(axis=1))))
```

```
Accuracy before PCA: 0.93333
```

In [26]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
# Train, test splits
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=.3,
                                                    stratify=y,
                                                    random_state=42)

# Use logistic regression to predict classes for test set
clf = LogisticRegression()
```

```
clf.fit(X_train, y_train)
preds = clf.predict_proba(X_test)
print('Accuracy after PCA: {:.5f}'.format(accuracy_score(y_test,
                                                 preds.argmax(axis=1))))
```

```
Accuracy after PCA: 0.95556
```

In [27]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
def apply_regression(X_train, y_train, X_test):
    ridge = Ridge()
    weight = ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_test)

    return y_pred
```

In [28]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
#load the dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

print(X.shape, y.shape)

pca = PCA()
pca.fit(X)

X_pca = pca.transform_rv(X, retained_variance = 0.9)
print("data shape with PCA ",X_pca.shape)
```

```
(442, 10) (442,)
data shape with PCA  (442, 7)
```

In [29]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
# Train, test splits
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_s

#Ridge regression without PCA
y_pred = apply_regression(X_train, y_train, X_test)

# calculate RMSE
rmse_score = np.sqrt(mean_squared_error(y_pred, y_test))
print('RMSE score using Ridge Regression before PCA: {:.5}'.format(rmse_score))
```

```
RMSE score using Ridge Regression before PCA: 55.794
```

In [30]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
#Ridge regression with PCA
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=.3, rand

#use Ridge Regression for getting predicted labels
y_pred = apply_regression(X_train,y_train,X_test)

#calculate RMSE
```

```
rmse_score = np.sqrt(mean_squared_error(y_pred, y_test))
print('RMSE score using Ridge Regression after PCA: {:.5}'.format(rmse_score))
```

```
RMSE score using Ridge Regression after PCA: 55.725
```

2.3 Reduced Masked Dataset Analysis [5 pts] **[W]**

1. If the face mask dataset that has been reduced to 2 features was fed into a classifier, do you think the classifier would produce high accuracy or low accuracy? Why? (One or two sentences will suffice for this question.) **(3 pts)**

   The classifier would produce low accuracy. Such a large reduction in features would lead to important features being ignored, leading to the classification to be inaccurate.

2. Assuming an equal rate of accuracy, what do you think is the main advantage in feeding a classifier a dataset with 2 features vs a dataset with 4096 features? (One sentence will suffice for this question.) **(2 pts)**

   Smaller number of features would lead to better(faster) computational performance.

3 Polynomial regression and regularization [50pts + 20pts Bonus for Undergrads + 10pts Bonus for All] **[P]** | **[W]**

3.1 Regression and regularization implementations [30 pts + 20 pts bonus for CS 4641] **[P]**

We have three methods to fit linear and ridge regression models: 1) closed form solution; 2) gradient descent (GD); 3) stochastic gradient descent (SGD). Some of the functions are bonus, see the below function list on what is required to be implemented for graduate and undergraduate students. We use the term weight in the following code. Weights and parameters (θ) have the same meaning here. We used parameters (θ) in the lecture slides.

In the **regression.py** file, complete the Regression class by implementing the functions:

- **rmse**
- **construct_polynomial_feats**
- **predict**
- **linear_fit_closed**: You should use `np.linalg.pinv` in this function
- **linear_fit_GD** (bonus for undergrad, **required for grad**)
- **linear_fit_SGD** (bonus for undergrad, **required for grad**)
- **ridge_fit_closed**: You should adjust your `I` matrix to handle the bias term differently than the rest of the terms
- **ridge_fit_GD** (bonus for undergrad, **required for grad**)
- **ridge_fit_SGD** (bonus for undergrad, **required for grad**)
- **ridge_cross_validation**: Use `ridge_fit_closed` for this function

The points for each function is in the **Deliverables and Points Distribution** section.

3.1.1 Local Tests for Helper Regression Functions [No Points]

You may test your implementation of the functions contained in **regression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

In [71]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from regression import Regression
from local_tests_folder.regression_test import Regression_Test

test_reg =  Regression_Test()
reg = Regression()

print('Local Tests for Helper Regression Functions\n')
rmse_test = np.allclose(reg.rmse(test_reg.predict, test_reg.y_all), test_reg.rms
print('Your rmse works within the expected range:', rmse_test)

cpf_test = np.allclose(reg.construct_polynomial_feats(test_reg.x_all, 2), test_r
print('Your construct_poly_feats works within the expected range:', cpf_test)

predict_test = np.allclose(reg.predict(test_reg.x_all_feat, test_reg.true_weight
print('Your predict works within the expected range:', predict_test)
```

```
Local Tests for Helper Regression Functions

Your rmse works within the expected range: True
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-71-f94da31253fe> in <module>
     13 print('Your rmse works within the expected range:', rmse_test)
     14
---> 15 cpf_test = np.allclose(reg.construct_polynomial_feats(test_reg.x_all, 2)
, test_reg.construct_poly)
     16 print('Your construct_poly_feats works within the expected range:', cpf_
test)
     17

~/repos/CS-4641/HW3/hw3_code/regression.py in construct_polynomial_feats(self,
 x, degree)
     59         #print(feat[:,1].shape)
     60         #print(x.shape)
---> 61         feat[:, 1] = x
     62         for i in range(x.shape[0]):
     63             for j in range(1, degree + 1):

ValueError: could not broadcast input array from shape (9,2) into shape (9,)
```

3.1.2 Local Tests for Linear Regression Functions [No Points]

You may test your implementation of the functions contained in **regression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

In [129…
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
```

```
from regression import Regression
from local_tests_folder.regression_test import Regression_Test

test_reg =  Regression_Test()
reg = Regression()

print('Local Tests for Linear Regression\n')
linear_closed_test = np.allclose(reg.linear_fit_closed(test_reg.x_all_feat, test
print('Your linear_fit_closed works within the expected range:', linear_closed_t

linear_GD, linear_GD_loss = reg.linear_fit_GD(test_reg.x_all_feat, test_reg.y_al
lgd_test = np.allclose(linear_GD, test_reg.linear_GD)
lgd_loss_test = np.allclose(linear_GD_loss, test_reg.linear_GD_loss)
print('\nYour linear_fit_GD weights are within the expected range:', lgd_test)
print('Your linear_fit_GD loss per epoch are within the expected range:', lgd_lo

linear_SGD, linear_SGD_loss = reg.linear_fit_SGD(test_reg.x_all_feat, test_reg.y
lsgd_test = np.allclose(linear_SGD, test_reg.linear_SGD)
lsgd_loss_test = np.allclose(linear_SGD_loss, test_reg.linear_SGD_loss)
print('\nYour linear_fit_SGD weights are within the expected range:', lsgd_test)
print('Your linear_fit_SGD loss per epoch are within the expected range:', lsgd_
```

```
Local Tests for Linear Regression

Your linear_fit_closed works within the expected range: True
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-129-892aa526994c> in <module>
     15 linear_GD, linear_GD_loss = reg.linear_fit_GD(test_reg.x_all_feat, test_
reg.y_all)
     16 lgd_test = np.allclose(linear_GD, test_reg.linear_GD)
---> 17 lgd_loss_test = np.allclose(linear_GD_loss, test_reg.linear_GD_loss)
     18 print('\nYour linear_fit_GD weights are within the expected range:', lgd
_test)
     19 print('Your linear_fit_GD loss per epoch are within the expected range:'
, lgd_loss_test)

~/opt/anaconda3/lib/python3.8/site-packages/numpy/core/overrides.py in allclose
(*args, **kwargs)

~/opt/anaconda3/lib/python3.8/site-packages/numpy/core/numeric.py in allclose(a,
b, rtol, atol, equal_nan)
   2263
   2264     """
-> 2265     res = all(isclose(a, b, rtol=rtol, atol=atol, equal_nan=equal_nan))
   2266     return bool(res)
   2267

~/opt/anaconda3/lib/python3.8/site-packages/numpy/core/overrides.py in isclose(*
args, **kwargs)

~/opt/anaconda3/lib/python3.8/site-packages/numpy/core/numeric.py in isclose(a,
 b, rtol, atol, equal_nan)
   2373     yfin = isfinite(y)
   2374     if all(xfin) and all(yfin):
-> 2375         return within_tol(x, y, atol, rtol)
   2376     else:
   2377         finite = xfin & yfin

~/opt/anaconda3/lib/python3.8/site-packages/numpy/core/numeric.py in within_tol
(x, y, atol, rtol)
   2354     def within_tol(x, y, atol, rtol):
```

```
   2355         with errstate(invalid='ignore'):
-> 2356             return less_equal(abs(x-y), atol + rtol * abs(y))
   2357
   2358     x = asanyarray(a)

ValueError: operands could not be broadcast together with shapes (0,) (5,)
```

3.1.3 Local Tests for Ridge Regression Functions [No Points]

You may test your implementation of the functions contained in **regression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

In [128…

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from regression import Regression
from local_tests_folder.regression_test import Regression_Test

test_reg =  Regression_Test()
reg = Regression()

print('Local Tests for Ridge Regression & Cross Validation\n')
ridge_closed_test = np.allclose(reg.ridge_fit_closed(test_reg.x_all_feat, test_r
print('Your ridge_fit_closed works within the expected range:', ridge_closed_tes

ridge_GD, ridge_GD_loss = reg.ridge_fit_GD(test_reg.x_all_feat, test_reg.y_all,
rgd_test = np.allclose(ridge_GD, test_reg.ridge_GD)
rgd_loss_test = np.allclose(ridge_GD_loss, test_reg.ridge_GD_loss)
print('\nYour ridge_fit_GD weights are within the expected range:', rgd_test)
print('Your ridge_fit_GD loss per epoch are within the expected range:', rgd_los

ridge_SGD, ridge_SGD_loss = reg.ridge_fit_SGD(test_reg.x_all_feat, test_reg.y_al
rsgd_test = np.allclose(ridge_SGD, test_reg.ridge_SGD)
rsgd_loss_test = np.allclose(ridge_SGD_loss, test_reg.ridge_SGD_loss)
print('\nYour ridge_fit_SGD weights are within the expected range:', rsgd_test)
print('Your ridge_fit_SGD loss per epoch are within the expected range:', rsgd_l

cross_val_test = np.allclose(reg.ridge_cross_validation(test_reg.x_all_feat, tes
print('\nYour ridge_cross_validation works within the expected range:', cross_va
```

```
Local Tests for Ridge Regression & Cross Validation

Your ridge_fit_closed works within the expected range: True
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-128-b1eb696f17ea> in <module>
     15 ridge_GD, ridge_GD_loss = reg.ridge_fit_GD(test_reg.x_all_feat, test_reg
.y_all, 10, 5)
     16 rgd_test = np.allclose(ridge_GD, test_reg.ridge_GD)
---> 17 rgd_loss_test = np.allclose(ridge_GD_loss, test_reg.ridge_GD_loss)
     18 print('\nYour ridge_fit_GD weights are within the expected range:', rgd_
test)
     19 print('Your ridge_fit_GD loss per epoch are within the expected range:',
rgd_loss_test)

~/opt/anaconda3/lib/python3.8/site-packages/numpy/core/overrides.py in allclose
(*args, **kwargs)

~/opt/anaconda3/lib/python3.8/site-packages/numpy/core/numeric.py in allclose(a,
```

```
b, rtol, atol, equal_nan)
   2263
   2264     """
-> 2265     res = all(isclose(a, b, rtol=rtol, atol=atol, equal_nan=equal_nan))
   2266     return bool(res)
   2267

~/opt/anaconda3/lib/python3.8/site-packages/numpy/core/overrides.py in isclose(*
args, **kwargs)

~/opt/anaconda3/lib/python3.8/site-packages/numpy/core/numeric.py in isclose(a,
 b, rtol, atol, equal_nan)
   2373     yfin = isfinite(y)
   2374     if all(xfin) and all(yfin):
-> 2375         return within_tol(x, y, atol, rtol)
   2376     else:
   2377         finite = xfin & yfin

~/opt/anaconda3/lib/python3.8/site-packages/numpy/core/numeric.py in within_tol
(x, y, atol, rtol)
   2354     def within_tol(x, y, atol, rtol):
   2355         with errstate(invalid='ignore'):
-> 2356             return less_equal(abs(x-y), atol + rtol * abs(y))
   2357
   2358     x = asanyarray(a)

ValueError: operands could not be broadcast together with shapes (0,) (5,)
```

3.2 About RMSE [3 pts] **[W]**

What is a good RMSE value? If we normalize our labels such that the outputs of our regression model can only be between 0 and 1, what does it mean when normalized RMSE = 1? Please provide an example with your explanation.

Answer

A lower RMSE value is often desired. However, it is difficult to evaluate RMSE since datasets could have a large rage of values. With the same RMSE value of 10, if our dataset's value range is from 0 to 1,000,000, that would mean our model is performing really good. However, if our dataset's value range is from 0 to 20, then our model performance is terrible. Therefore, normalization is needed to conclude a better evaluation of our model.

A RMSE value of 1 means that our model's prediction has a difference to the actual data by 1 after normalization, which means that the predictions are as bad as they can be. Therefore, Normalized RMSE = 1 means that we have the worst model possible.

3.3 Testing: General Functions and Linear Regression [5 pts] **[W]**

In this section. we will test the performance of the linear regression. As long as your test RMSE score is close to the TA's answer (TA's answer ± 0.5), you can get full points. Let's first construct a dataset for polynomial regression.

In this case, we construct the polynomial features up to degree 5. Each data sample consists of two features $[a, b]$. We compute the polynomial features of both a and b in order to yield the vectors $[1, a, a^2, a^3, \ldots, a^{degree}]$ and $[1, b, b^2, b^3, \ldots, b^{degree}]$. We train our model with the

cartesian product of these polynomial features. The cartesian product generates a new feature vector consisting of all polynomial combinations of the features with degree less than or equal to the specified degree.

For example, if degree = 2, we will have the polynomial features $[1, a, a^2]$ and $[1, b, b^2]$ for the datapoint $[a, b]$. The cartesian product of these two vectors will be $[1, a, b, ab, a^2, b^2]$. We do not generate a^3 and b^3 since their degree is greater than 2 (specified degree).

In [141…
```
from regression import Regression
```

In [142…
```
################################
### DO NOT CHANGE THIS CELL ###
################################
POLY_DEGREE = 7
N_SAMPLES = 1200

rng = np.random.RandomState(seed=10)

# Simulating a regression dataset with polynomial features.
true_weight = rng.rand(POLY_DEGREE ** 2 + 2, 1)
x_feature1 = np.linspace(-5, 5, N_SAMPLES)
x_feature2 = np.linspace(-3, 3, N_SAMPLES)
x_all = np.stack((x_feature1, x_feature2), axis=1)

reg = Regression()
x_all_feat = reg.construct_polynomial_feats(x_all, POLY_DEGREE)
x_cart_flat = []
for i in range(x_all_feat.shape[0]):
    point = x_all_feat[i]
    x1 = point[:,0]
    x2 = point[:,1]
    x1_end = x1[-1]
    x2_end = x2[-1]
    x1 = x1[:-1]
    x2 = x2[:-1]
    x3 = np.asarray([[m*n for m in x1] for n in x2])

    x3_flat = list(np.reshape(x3, (x3.shape[0] ** 2)))
    x3_flat.append(x1_end)
    x3_flat.append(x2_end)
    x3_flat = np.asarray(x3_flat)
    x_cart_flat.append(x3_flat)

x_cart_flat = np.asarray(x_cart_flat)
x_cart_flat = (x_cart_flat - np.mean(x_cart_flat)) / np.std(x_cart_flat)  # Norm
x_all_feat = np.copy(x_cart_flat)

# We must add noise to data, else the data will look unrealistically perfect.
y_noise = rng.randn(x_all_feat.shape[0], 1)
y_all = np.dot(x_cart_flat, true_weight) + y_noise
print("x_all: ", x_all.shape[0], " (rows/samples) ", x_all.shape[1], " (columns/
print("y_all: ", y_all.shape[0], " (rows/samples) ", y_all.shape[1], " (columns/
```

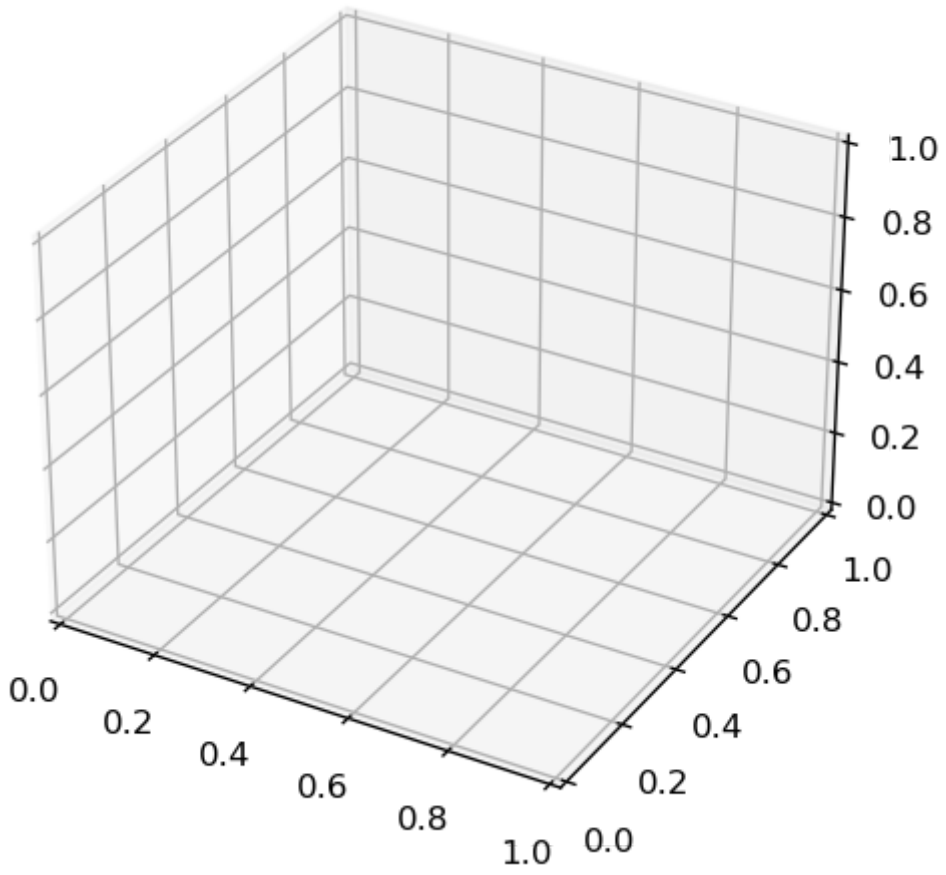
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-142-93cfbd779c33> in <module>
     14
     15 reg = Regression()
```

```
---> 16 x_all_feat = reg.construct_polynomial_feats(x_all, POLY_DEGREE)
     17 x_cart_flat = []
     18 for i in range(x_all_feat.shape[0]):

~/repos/CS-4641/HW3/hw3_code/regression.py in construct_polynomial_feats(self,
 x, degree)
     57         """
     58         feat = np.ones((x.shape[0], degree + 1))
---> 59         feat[:, 1] = x
     60         for i in range(x.shape[0]):
     61             for j in range(1, degree + 1):

ValueError: could not broadcast input array from shape (1200,2) into shape (120
0,)
```

In [106…

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
fig = plt.figure(figsize=(8,5), dpi=120)
ax = fig.add_subplot(111, projection='3d')

p = np.reshape(np.dot(x_cart_flat, true_weight), (N_SAMPLES,))
ax.scatter(x_all[:,0], x_all[:,1], y_all, label='Datapoints', s=4, alpha=0.2)
ax.plot(x_all[:,0], x_all[:,1], p, label='Line of Best Fit', c="red", linewidth=
ax.set_xlabel("feature 1")
ax.set_ylabel("feature 2")
ax.set_zlabel("y")

if not STUDENT_VERSION:
    ax.text2D(0.5, 0.5, EO_TEXT, transform=ax.transAxes,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.4, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)

ax.legend()
ax.text2D(0.05, 0.95, "All Simulated Datapoints", transform=ax.transAxes)
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-106-e7abd8b79aa5> in <module>
      5 ax = fig.add_subplot(111, projection='3d')
      6
----> 7 p = np.reshape(np.dot(x_cart_flat, true_weight), (N_SAMPLES,))
      8 ax.scatter(x_all[:,0], x_all[:,1], y_all, label='Datapoints', s=4, alpha
=0.2)
      9 ax.plot(x_all[:,0], x_all[:,1], p, label='Line of Best Fit', c="red", li
newidth=2)

NameError: name 'x_cart_flat' is not defined
```



In the figure above, the red curve is the true fuction we want to learn, while the blue dots are the noisy data points. The data points are generated by $Y = X\theta + \epsilon$, where $\epsilon_i \sim N(0, 1)$ are i.i.d. generated noise.

Now let's split the data into two parts, the training set and testing set. The yellow dots are for training, while the black dots are for testing.

In [107…

```
################################
### DO NOT CHANGE THIS CELL ###
################################
PERCENT_TRAIN = 0.8

all_indices = rng.permutation(N_SAMPLES)  # Random indicies
train_indices = all_indices[:round(N_SAMPLES * PERCENT_TRAIN)]  # 80% Training
test_indices = all_indices[round(N_SAMPLES * PERCENT_TRAIN):]  # 20% Testing

xtrain = x_all[train_indices]
ytrain = y_all[train_indices]
xtest = x_all[test_indices]
ytest = y_all[test_indices]

# -- Plotting Code --
fig = plt.figure(figsize=(8,5), dpi=120)
ax = fig.add_subplot(111, projection='3d')

ax.scatter(xtrain[:,0], xtrain[:,1], ytrain, label='Training', c='y',s=4)
ax.scatter(xtest[:,0], xtest[:,1], ytest, label='Testing', c='black',s=4)
```

```
ax.set_xlabel("feature 1")
ax.set_ylabel("feature 2")
ax.set_zlabel("y")

if not STUDENT_VERSION:
    ax.text2D(0.5, 0.5, EO_TEXT, transform=ax.transAxes,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.4, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)

ax.legend(loc = 'upper right')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-107-5e6e1f6c9749> in <module>
      9
     10 xtrain = x_all[train_indices]
---> 11 ytrain = y_all[train_indices]
     12 xtest = x_all[test_indices]
     13 ytest = y_all[test_indices]

NameError: name 'y_all' is not defined
```

Now let us train our model using the training set and see how our model performs on the testing set. Observe the red line, which is our model's learned function.

In [108…

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for both Grad and Undergrad

weight = reg.linear_fit_closed(x_all_feat[train_indices], y_all[train_indices])
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all[test_indices])
print('Linear (closed) RMSE: %.4f' % test_rmse)

# -- Plotting Code --
fig = plt.figure(figsize=(8,5), dpi=120)
ax = fig.add_subplot(111, projection='3d')

y_pred = reg.predict(x_all_feat, weight)
y_pred = np.reshape(y_pred, (y_pred.size,))
ax.plot(x_all[:,0], x_all[:,1], y_pred, label='Trendline', color='r', lw=2, zord

ax.scatter(xtrain[:,0], xtrain[:,1], ytrain, label='Training', c='y',s=4)
ax.scatter(xtest[:,0], xtest[:,1], ytest, label='Testing', c='black',s=4)
ax.set_xlabel("feature 1")
ax.set_ylabel("feature 2")
ax.set_zlabel("y")

if not STUDENT_VERSION:
    ax.text2D(0.5, 0.5, EO_TEXT, transform=ax.transAxes,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.4, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)

ax.text2D(0.05, 0.95, "Linear (Closed)", transform=ax.transAxes)
ax.legend(loc = 'upper right')
plt.show()
```

```
---------------------------------------------------------------------------
```

```
NameError                                 Traceback (most recent call last)
<ipython-input-108-5421c8a31836> in <module>
      5 # Required for both Grad and Undergrad
      6
----> 7 weight = reg.linear_fit_closed(x_all_feat[train_indices], y_all[train_in
dices])
      8 y_test_pred = reg.predict(x_all_feat[test_indices], weight)
      9 test_rmse = reg.rmse(y_test_pred, y_all[test_indices])

NameError: name 'x_all_feat' is not defined
```

HINT: If your RMSE is off, make sure to follow the instruction given for `linear_fit_closed` in the list of functions to implement above.

Now let's use our linear gradient descent function with the same setup. Observe that the trendline is now less optimal, and our RMSE decreased. Do not be alarmed.

In [109…

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for Grad Only
# This cell may take more than 1 minute

weight, _ = reg.linear_fit_GD(x_all_feat[train_indices],
                              y_all[train_indices],
                              epochs=50000,
                              learning_rate=1e-8)
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all[test_indices])
print('Linear (GD) RMSE: %.4f' % test_rmse)

# -- Plotting Code --
fig = plt.figure(figsize=(8,5), dpi=120)
ax = fig.add_subplot(111, projection='3d')

y_pred = reg.predict(x_all_feat, weight)
y_pred = np.reshape(y_pred, (y_pred.size,))
ax.plot(x_all[:,0], x_all[:,1], y_pred, label='Trendline', color='r', lw=2, zord

ax.scatter(xtrain[:,0], xtrain[:,1], ytrain, label='Training', c='y',s=4)
ax.scatter(xtest[:,0], xtest[:,1], ytest, label='Testing', c='black',s=4)
ax.set_xlabel("feature 1")
ax.set_ylabel("feature 2")
ax.set_zlabel("y")

if not STUDENT_VERSION:
    ax.text2D(0.5, 0.5, EO_TEXT, transform=ax.transAxes,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.4, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)

ax.text2D(0.05, 0.95, "Linear (GD)", transform=ax.transAxes)
ax.legend(loc = 'upper right')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-109-fab8686a13c6> in <module>
      6 # This cell may take more than 1 minute
      7
```

```
----> 8 weight, _ = reg.linear_fit_GD(x_all_feat[train_indices],
      9                               y_all[train_indices],
     10                               epochs=50000,

NameError: name 'x_all_feat' is not defined
```

We must tune our epochs and learning_rate. As we tune these parameters our trendline will approach the trendline generated by the linear closed form solution. Observe how we slowly tune (increase) the epochs and learning_rate below to create a better model.

Note that the closed form solution will always give the most optimal/overfit results. We cannot outperform the closed form solution with GD. We can only approach closed forms level of optimality/overfitness. We leave the reasoning behind this as an exercise to the reader.

In [118…

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for Grad Only
# This cell may take more than 1 minute

learning_rates = [1e-8, 1e-6, 1e-4]
weights = np.zeros((3, POLY_DEGREE ** 2 + 2))

for ii in range(len(learning_rates)):
    weights[ii,:] = reg.linear_fit_GD(x_all_feat[train_indices],
                                      y_all[train_indices],
                                      epochs=50000,
                                      learning_rate=learning_rates[ii])[0].ravel
    y_test_pred = reg.predict(x_all_feat[test_indices],
                              weights[ii, :].reshape((POLY_DEGREE ** 2 + 2, 1)))
    test_rmse = reg.rmse(y_test_pred, y_all[test_indices])
    print('Linear (GD) RMSE: %.4f (learning_rate=%s)' % (test_rmse, learning_rat

# -- Plotting Code --
fig = plt.figure(figsize=(8,5), dpi=120)
ax = fig.add_subplot(111, projection='3d')

colors = ['g', 'orange', 'r']
for ii in range(len(learning_rates)):
    y_pred = reg.predict(x_all_feat, weights[ii])
    y_pred = np.reshape(y_pred, (y_pred.size,))
    ax.plot(x_all[:,0], x_all[:,1], y_pred,
            label='Trendline LR=' + str(learning_rates[ii]),
            color=colors[ii], lw=2, zorder=5)

ax.scatter(xtrain[:,0], xtrain[:,1], ytrain, label='Training', c='y',s=4)
ax.scatter(xtest[:,0], xtest[:,1], ytest, label='Testing', c='black',s=4)
ax.set_xlabel("feature 1")
ax.set_ylabel("feature 2")
ax.set_zlabel("y")

if not STUDENT_VERSION:
    ax.text2D(0.5, 0.5, EO_TEXT, transform=ax.transAxes,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.4, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)

ax.text2D(0.05, 0.95, "Tunning Linear (GD)", transform=ax.transAxes)
```

```
ax.legend(loc = 'upper right')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-118-4194e2c4bca6> in <module>
     10
     11 for ii in range(len(learning_rates)):
---> 12     weights[ii,:] = reg.linear_fit_GD(x_all_feat[train_indices],
     13                                       y_all[train_indices],
     14                                       epochs=50000,

NameError: name 'x_all_feat' is not defined
```

And what if we just use the first 10 data points to train?

In [117…

```
################################
### DO NOT CHANGE THIS CELL ###
################################
rng = np.random.RandomState(seed=5)
y_all_noisy = np.dot(x_cart_flat, np.zeros((POLY_DEGREE ** 2 + 2, 1))) + rng.ran
sub_train = train_indices[10:20]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-117-2269c32e3750> in <module>
      3 ################################
      4 rng = np.random.RandomState(seed=5)
----> 5 y_all_noisy = np.dot(x_cart_flat, np.zeros((POLY_DEGREE ** 2 + 2, 1))) +
rng.randn(x_all_feat.shape[0], 1)
      6 sub_train = train_indices[10:20]

NameError: name 'x_cart_flat' is not defined
```

In [116…

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for both Grad and Undergrad

weight = reg.linear_fit_closed(x_all_feat[sub_train], y_all_noisy[sub_train])
y_pred = reg.predict(x_all_feat, weight)
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all_noisy[test_indices])
print('Linear (closed) 10 Samples RMSE: %.4f' % test_rmse)

# -- Plotting Code --
fig = plt.figure(figsize=(8,5), dpi=120)
ax = fig.add_subplot(111, projection='3d')

x1 = x_all[:,0]
x2 = x_all[:,1]
y_pred = np.reshape(y_pred, (N_SAMPLES,))
ax.plot(x1, x2, y_pred, color='b', lw=4)

x3 = x_all[sub_train,0]
x4 = x_all[sub_train,1]
ax.scatter(x3, x4, y_all_noisy[sub_train], s=100, c='r', marker='x')

y_test_pred = reg.predict(x_all_feat[test_indices], weight)
ax.set_xlabel("feature 1")
ax.set_ylabel("feature 2")
```

```
ax.set_zlabel("y")
ax.set_zlim([None, 8])

if not STUDENT_VERSION:
    ax.text2D(0.5, 0.5, EO_TEXT, transform=ax.transAxes,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.4, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)

ax.text2D(0.05, 0.95, "Linear Regression (Closed)", transform=ax.transAxes)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-116-238843a81aad> in <module>
      5 # Required for both Grad and Undergrad
      6
----> 7 weight = reg.linear_fit_closed(x_all_feat[sub_train], y_all_noisy[sub_tr
ain])
      8 y_pred = reg.predict(x_all_feat, weight)
      9 y_test_pred = reg.predict(x_all_feat[test_indices], weight)

NameError: name 'x_all_feat' is not defined
```

Did you see a worse performance? Let's take a closer look at what we have learned.

3.4 Testing: Testing ridge regression [5 pts] **[W]**

Now let's try ridge regression. Like before, undergraduate students need to implement the closed form, and graduate students need to implement all three methods. We will call the prediction function from linear regression part. As long as your test RMSE score is close to the TA's answer (TA's answer ± 0.5), you can get full points.

Again, let's see what we have learned. **You only need to run the cell corresponding to your specific implementation.**

In [134…

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for both Grad and Undergrad

weight = reg.ridge_fit_closed(x_all_feat[sub_train],
                              y_all_noisy[sub_train],
                              c_lambda=10)
y_pred = reg.predict(x_all_feat, weight)
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all_noisy[test_indices])
print('Ridge Regression (closed) RMSE: %.4f' % test_rmse)

# -- Plotting Code --
fig = plt.figure(figsize=(8,5), dpi=120)
ax = fig.add_subplot(111, projection='3d')

x1 = x_all[:,0]
x2 = x_all[:,1]
y_pred = np.reshape(y_pred, (N_SAMPLES,))
ax.plot(x1, x2, y_pred, color='b', lw=4)

x3 = x_all[sub_train,0]
x4 = x_all[sub_train,1]
```

```
ax.scatter(x3, x4, y_all_noisy[sub_train], s=100, c='r', marker='x')

y_test_pred = reg.predict(x_all_feat[test_indices], weight)
ax.set_xlabel("feature 1")
ax.set_ylabel("feature 2")
ax.set_zlabel("y")

if not STUDENT_VERSION:
    ax.text2D(0.5, 0.5, EO_TEXT, transform=ax.transAxes,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.4, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)
ax.set_zlim([None, 8])
ax.text2D(0.05, 0.95, "Ridge Regression (Closed)", transform=ax.transAxes)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-134-8c9cfc0bc944> in <module>
      5 # Required for both Grad and Undergrad
      6
----> 7 weight = reg.ridge_fit_closed(x_all_feat[sub_train],
      8                                y_all_noisy[sub_train],
      9                                c_lambda=10)

NameError: name 'x_all_feat' is not defined
```

HINT: Make sure to follow the instruction given for `ridge_fit_closed` in the list of functions to implement above.

In [140…

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for Grad Only

weight, _ = reg.ridge_fit_GD(x_all_feat[sub_train],
                             y_all_noisy[sub_train],
                             c_lambda=10, learning_rate=1e-5)
y_pred = reg.predict(x_all_feat, weight)
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all_noisy[test_indices])
print('Ridge Regression (GD) RMSE: %.4f' % test_rmse)

# -- Plotting Code --
fig = plt.figure(figsize=(8,5), dpi=120)
ax = fig.add_subplot(111, projection='3d')

x1 = x_all[:,0]
x2 = x_all[:,1]
y_pred = np.reshape(y_pred, (N_SAMPLES,))
ax.plot(x1, x2, y_pred, color='b', lw=4)

x3 = x_all[sub_train,0]
x4 = x_all[sub_train,1]
ax.scatter(x3, x4, y_all_noisy[sub_train], s=100, c='r', marker='x')

y_test_pred = reg.predict(x_all_feat[test_indices], weight)
ax.set_xlabel("feature 1")
ax.set_ylabel("feature 2")
ax.set_zlabel("y")
ax.set_zlim([None, 8])
```

```
if not STUDENT_VERSION:
    ax.text2D(0.5, 0.5, EO_TEXT, transform=ax.transAxes,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.4, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)

ax.text2D(0.05, 0.95, "Ridge Regression (GD)", transform=ax.transAxes)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-140-405c5f820b2d> in <module>
      5 # Required for Grad Only
      6
----> 7 weight, _ = reg.ridge_fit_GD(x_all_feat[sub_train],
      8                             y_all_noisy[sub_train],
      9                             c_lambda=10, learning_rate=1e-5)

NameError: name 'x_all_feat' is not defined
```

In [138…

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for Grad Only

weight, _ = reg.ridge_fit_SGD(x_all_feat[sub_train],
                           y_all_noisy[sub_train],
                           c_lambda=10,
                           learning_rate=1e-5)
y_pred = reg.predict(x_all_feat, weight)
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all_noisy[test_indices])
print('Ridge Regression (SGD) RMSE: %.4f' % test_rmse)


# -- Plotting Code --
fig = plt.figure(figsize=(8,5), dpi=120)
ax = fig.add_subplot(111, projection='3d')

x1 = x_all[:,0]
x2 = x_all[:,1]
y_pred = np.reshape(y_pred, (N_SAMPLES,))
ax.plot(x1, x2, y_pred, color='b', lw=4)

x3 = x_all[sub_train,0]
x4 = x_all[sub_train,1]
ax.scatter(x3, x4, y_all_noisy[sub_train], s=100, c='r', marker='x')

y_test_pred = reg.predict(x_all_feat[test_indices], weight)
ax.set_xlabel("feature 1")
ax.set_ylabel("feature 2")
ax.set_zlabel("y")
ax.set_zlim([None, 8])

if not STUDENT_VERSION:
    ax.text2D(0.5, 0.5, EO_TEXT, transform=ax.transAxes,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.4, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)

ax.text2D(0.05, 0.95, "Ridge Regression (SGD)", transform=ax.transAxes)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-138-9b9468c41c88> in <module>
      5 # Required for Grad Only
      6
----> 7 weight, _ = reg.ridge_fit_SGD(x_all_feat[sub_train],
      8                              y_all_noisy[sub_train],
      9                              c_lambda=10,

NameError: name 'x_all_feat' is not defined
```

3.5 Cross validation [7 pts] **[W]**

Let's use Cross Validation to find the best value for `c_lambda` in ridge regression.

In [139…

```
################################
### DO NOT CHANGE THIS CELL ###
################################
# We provided 6 possible values for lambda, and you will use them in cross valid
# For cross validation, use 10-fold method and only use it for your training dat
# For the training data, split them in 10 folds which means that use 10 percent
# At the end for each lambda, you have calculated 10 rmse and get the mean value
# That's it. Pick up the lambda with the lowest mean value of rmse.
#
# ****HINTS****
#  1. np.concatenate is your friend
#  2. Make sure to follow the instruction given for ridge_fit_closed in the list
#

best_lambda = None
best_error = None
kfold = 10
lambda_list = [0.0001, 0.001, 0.1, 1, 5, 10, 50, 100, 1000, 10000]

for lm in lambda_list:
    err = reg.ridge_cross_validation(x_all_feat[train_indices], y_all[train_indi
    print('Lambda: %.4f' % lm, 'RMSE: %.6f'% err)
    if best_error is None or err < best_error:
        best_error = err
        best_lambda = lm

print('Best Lambda: %.4f' % best_lambda)
weight = reg.ridge_fit_closed(x_all_feat[train_indices], y_all_noisy[train_indic
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all_noisy[test_indices])
print('Best Test RMSE: %.4f' % test_rmse)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-139-eea76f3059bb> in <module>
     19
     20 for lm in lambda_list:
---> 21     err = reg.ridge_cross_validation(x_all_feat[train_indices], y_all[tr
ain_indices], kfold, lm)
     22     print('Lambda: %.4f' % lm, 'RMSE: %.6f'% err)
     23     if best_error is None or err < best_error:

NameError: name 'x_all_feat' is not defined
```

3.6 Noisy Input Samples in Linear Regression [10 pts Bonus for All]

[W]

Consider a linear model of the form:

$$y(x_n, \theta) = \theta_0 + \sum_{d=1}^{D} \theta_d x_{nd}$$

where $x_n = (x_{n1}, \ldots, x_{nD}) \in \mathbb{R}^D$ and weights $\theta = (\theta_0, \ldots, \theta_D) \in \mathbb{R}^D$. Given the the D-dimension input sample set $x = \{x_1, \ldots, x_n\}$ with corresponding target value $y = \{y_1, \ldots, y_n\}$, the sum-of-squares error function is:

$$E_D(\theta) = \frac{1}{2} \sum_{n=1}^{N} \left[y(x_n, \theta) - y_n \right]^2$$

Now, suppose that Gaussian noise $\epsilon_n \in \mathbb{R}^D$ is added independently to each of the input sample x_n to generate a new sample set $x' = \{x_1 + \epsilon_1, \ldots, x_n + \epsilon_n\}$. Here, ϵ_{ni} (an entry of ϵ_n) has zero mean and variance σ^2. For each sample x_n, let $x'_n = (x_{n1} + \epsilon_{n1}, \ldots, x_{nD} + \epsilon_{nd})$, where n and d is independent across both n and d indices.

1. (3pts) Show that $y(x'_n, \theta) = y(x_n, \theta) + \sum_{d=1}^{D} \theta_d \epsilon_{nd}$
2. (7pts) Assume the sum-of-squares error function of the noise sample set $x' = \{x_1 + \epsilon_1, \ldots, x_n + \epsilon_n\}$ is $E_D(\theta)'$. Prove the expectation of $E_D(\theta)'$ is equivalent to the sum-of-squares error $E_D(\theta)$ for noise-free input samples with the addition of a weight-decay regularization term (e.g. L_2 norm) , in which the bias parameter θ_0 is omitted from the regularizer. In other words, show that

$$E[E_D(\theta)'] = E_D(\theta) + \text{Regularizer}.$$

N.B. You should be incorporating your solution from the first part of this problem into the given sum of squares equation for the second part.

Hint:

- During the class, we have discussed how to solve for the weight θ for ridge regression, the function looks like this:

  $$E(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left[y(x_i, \theta) - y_i \right]^2 + \frac{\lambda}{N} \sum_{i=1}^{d} ||\theta_i||^2$$

  where the first term is the sum-of-squares error and the second term is the regularization term. N is the number of samples. In this question, we use another form of the ridge regression, which is:

  $$E(\theta) = \frac{1}{2} \sum_{i=1}^{N} \left[y(x_i, \theta) - y_i \right]^2 + \frac{\lambda}{2} \sum_{i=1}^{d} ||\theta_i||^2$$

- For the Gaussian noise ϵ_n, we have $E[\epsilon_n] = 0$

- Assume the noise $\epsilon = (\epsilon_1, \ldots, \epsilon_n)$ are **independent** to each other, we have

$$E[\epsilon_n \epsilon_m] = \begin{cases} \sigma^2 & m = n \\ 0 & m \neq n \end{cases}$$

Answer

...

Q4: Naive Bayes Classification [25pts] **[P]** | **[W]**

In Bayesian classification, we're interested in finding the probability of a label given some observed feature vector $x = [x_1, .., x_d]$, which we can write as $P(y \mid x_1, .., x_d)$. Bayes's theorem tells us how to express this in terms of quantities we can compute more directly:

$$P(y \mid x_1, .., x_d) = \frac{P(x_1, .., x_d \mid y)P(y)}{P(x_1, .., x_d)} \tag{1}$$

The main assumption in Naive Bayes is that, given the label, the observed features are conditionally independent i.e.

$$P(x_1, .., x_d \mid y) = P(x_1 \mid y) \times .. \times P(x_d \mid y)$$

Therefore, we can rewrite Bayes rule as

$$P(y \mid x_1, .., x_d) = \frac{P(x_1 \mid y) \times .. \times P(x_d \mid y)P(y)}{P(x_1, .., x_d)} \tag{2}$$

Training Naive Bayes

One way to train a Naive Bayes classifier is done using frequentist approach to calculate probability, which is simply going over the training data and calculating the frequency of different observations in the training set given different labels. For example,

$$P(x_1 = i \mid y = j) = \frac{P(x_1 = i, y = j)}{P(y = j)} = \frac{\text{Number of times in training data } x_1 = i \text{ and}}{\text{Total number of times in training data } y =}$$

Testing Naive Bayes

During the testing phase, we try to estimate the probability of a label given an observed feature vector. We combine the probabilities computed from training data to estimate the probability of a given label. For example, if we are trying to decide between two labels y_1 and y_2, then we compute the ratio of the posterior probabilities for each label:

$$\frac{P(y_1 \mid x_1, .., x_d)}{P(y_2 \mid x_1, .., x_d)} = \frac{P(x_1, .., x_d \mid y_1)}{P(x_1, .., x_d \mid y_2)} \frac{P(y_1)}{P(y_2)} = \frac{P(x_1 \mid y_1) \times .. \times P(x_d \mid y_1)P(y_1)}{P(x_1 \mid y_2) \times .. \times P(x_d \mid y_2)P(y_2)} \tag{4}$$

All we need now is to compute $P(x_1|y_i), .., P(x_d \mid y_i)$ and $P(y_i)$ for each label by plugging in the numbers we got during training. The label with the higher posterior probabilities is the one

that is selected.

4.1 Llama Breed Problem [5pts] [W]

Above are images of two different breeds of llamas – the Suri llama and the Wooly llama. The difference between these two breeds is subtle, as these two breeds are often mixed up. However the Suri Llama is vastly more valuable than the Wooly llama. You devise a way to determine with some confidence, which llama is which – without the need for expensive genetic testing.

You look at four key features of the llama: {curly hair, over 14 inch tail, over 400 pounds, extremely shy}.

You only have 6 randomly chosen llamas to work with, and their breed as the ground truth. You record the data as vectors with the entry 1 if true and 0 if false. For example a llama with vector {1,1,0,1} would have curly hair, a tail over 14 inches, be **less** than 400 pounds, and be extremely shy.

The **Suri Llamas** yield the following data: {1,0,0,0}, {1,0,0,1}, {1,1,1,1}, {0,0,0,1}

The **Wooly Llamas** yield the following data: {0,1,1,0}, {1,1,1,0}

Now is the time to test your method!

You see a new llama you are interested in that has curly hair, a tail over 14 inches, is over 400 pounds, and is **not** shy.

Using Naive Bayes, **is this new llama a Suri or Wooly Llama?**

Answer According to the description, our feature vector is $\{1,1,1,0\}$

We set y_1 to be the event that the llama is a Suri Llama, and y_2 to be the event that the iiama is a Wooly llama. \\ $P(x_1|y_1) = \frac{3}{4}, P(x_2|y_1) = \frac{1}{4}, P(x_3|y_1) = \frac{1}{4}, P(x_4|y_1) = \frac{1}{4}$ \\ $P(x_1|y_2) = \frac{1}{2}, P(x_2|y_2) = \frac{2}{2} = 1, P(x_2|y_2)\frac{2}{2} = 1, P(x_2|y_2) = \frac{2}{2} = 1$ \

$$\frac{P(y_1 \mid x_1, .., x_d)}{P(y_2 \mid x_1, .., x_d)} = \frac{\frac{3}{4}\frac{1}{4}\frac{1}{4}\frac{1}{4}\frac{4}{6}}{\frac{1}{2} \cdot 1 \cdot 1 \cdot 1 \cdot \frac{2}{6}} < 1 \tag{5}$$

Therefore, the new llama is a Wooly Llama.

4.2 News Data Sentiment Classification [15pts] **[P]**

This dataset contains the sentiments for financial news headlines from the perspective of a retail investor. The sentiment of news has 3 classes, negative (class label = 0), neutral (class label = 1) and positive (class label = 2). There are 4846 news in total with 9 duplicates. We remove those duplicates to achieve 4837 unique news values and then randomly split the 4837

news into training set and evaluation set with 8:2 ratio. We use the training set to fit the Naive Bayes model and use the evaluation set to evaluate the accuracy of our model.

The code which is provided loads the documents and builds a "bag of words" representation of each document. Your task is to complete the missing portions of the code and to determine whether a news is negative, neutral or positive. (Hint: Label 0 denotes the news is negative, label 1 denotes the news is neutral and 2 means the news is positive. Our job here is to determine whether a news is negative, neutral or positive using Naive Bayes).

In the **nb.py** file, complete the following functions:

- **priors_prob**: calculates the ratio of class probabilities of 0-negative, 1-neutral, 2-positive. We do this based on word counts rather than document counts.
- **likelihood_ratio**: calculates the ratio of word probablities given the label of whether the news was 0-negative, 1-neutral, 2-positive.
- **analyze_sentiment**: takes in the likelihood ratio, priors probabilities for each class and a number of test news headlines in Bag-of-Words representation, and analyzes the sentiment for each news.

For example, if we have a matrix like: (the first column denotes the class label, the entries in the remaining columns denote the number of occurrences for each word). We have two more columns for words. The first word is "happy" and the second word is "useless"

$$\begin{bmatrix} label & happy & useless \\ 0(negative) & 1 & 4 \\ 0 & 0 & 6 \\ 1(neutral) & 3 & 2 \\ 2(positive) & 3 & 1 \\ 2 & 4 & 0 \end{bmatrix}$$

Then we have

$$prior(negative) = \frac{1+4+0+6}{1+4+0+6+3+2+3+1+4+0} = \frac{11}{24}$$

$$prior(neutral) = \frac{3+2}{1+4+0+6+3+2+3+1+4+0} = \frac{5}{24}$$

$$prior(positive) = \frac{3+1+4+0}{1+4+0+6+3+2+3+1+4+0} = \frac{8}{24}$$

Note 1: In likelihood_ratio(), add one to each word count so as to avoid issues with zero word count. This is known as Add-1 smoothing. It is a type of additive smoothing. For the numerator, we just add 1 at the end. For the denominator, we add 1 for each feature (in this example, for each word).

$$likelihood(negative) = [\frac{1+0+1}{1+0+1+4+6+1} \quad \frac{4+6+1}{1+0+1+4+6+1}] = [\frac{2}{13} \quad \frac{11}{13}]$$

$$likelihood(neutral) = [\frac{3+1}{3+2+1} \ \frac{2+1}{3+1+2+1}] = [\frac{4}{7} \ \frac{3}{7}]$$

$$likelihood(positive) = [\frac{3+4+1}{3+4+1+1+0+1} \ \frac{1+0+1}{3+4+1+1+0+1}] = [\frac{8}{10} \ \frac{2}{10}]$$

Note 2: In `analyze_sentiment()`, we can calculate the posterior probability given the count for each word

	happy	useless
Count	3	4

$$P(negative) = \left(\frac{2}{13}\right)^3 * \left(\frac{11}{13}\right)^4 * \frac{11}{24}$$

$$P(neutral) = \left(\frac{4}{7}\right)^3 * \left(\frac{3}{7}\right)^4 * \frac{5}{24}$$

$$P(positive) = \left(\frac{8}{10}\right)^3 * \left(\frac{2}{11}\right)^4 * \frac{8}{24}$$

The prediction will then be the label with the highest probability

No Local Test for Naive Bayes

There is no local test provided for Naive Bayes, however, we have constructed a toy dataset for you from the above example in **local_tests_folder/nb_test.py**. You can use this to create your own local tests, if your implementation is correct, the output from your functions should match the output from above.

In [135…
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from nb import NaiveBayes
```

In [136…
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

def build_and_test_model(X_train, y_train, X_test, y_test):
    list_of_labels = [X_train[y_train == label] for label in np.unique(y_train)]
    NB = NaiveBayes()
    likelihood_ratio = NB.likelihood_ratio(list_of_labels)
    priors_prob = NB.priors_prob(list_of_labels)
    resolved = NB.analyze_sentiment(likelihood_ratio, priors_prob, X_test)
    return np.sum(resolved == y_test) / len(resolved) * 1.

train = pd.read_csv("./data/news-data.csv",
                    encoding='cp437', header=None)
```

```
class_to_label_mappings = {
    "negative": 0,
    "neutral": 1,
    "positive": 2
}

train.columns = ["Sentiment", "News"]
train.drop_duplicates(inplace=True)

train["Sentiment"] = train["Sentiment"].map(
    class_to_label_mappings)

stop_words = text.ENGLISH_STOP_WORDS
vectorizer = text.CountVectorizer(stop_words=stop_words)

X = train['News'].values
y = train['Sentiment'].values

RANDOM_SEED = 5
BOW = vectorizer.fit_transform(X).toarray()
X_train, X_test, y_train, y_test = train_test_split(
    BOW, y, test_size=0.2, random_state=RANDOM_SEED)

accuracy = build_and_test_model(X_train, y_train, X_test, y_test)

# You should be getting around 70% of accuracy
print('Naive Bayes Model Test Accuracy:', round(accuracy * 100, 3), '%')
```

```
Naive Bayes Model Test Accuracy: 70.455 %
```

4.3 Accuracy result analysis [5pts] **[W]**

Are you satisfied with the result? Why or why not? What assumption did your model make that limits the accuracy?

Answer

I am not satisfied with the result, since it only reached around 70% accuracy.

Q5: Noise in PCA and Linear Regression [15 Points Bonus for All] **[W]**

Both PCA and least squares regression can be viewed as algorithms for inferring (linear) relationships among data variables. In this part of the assignment, you will develop some intuition for the differences between these two approaches and develop an understanding of the settings that are better suited to using PCA or better suited to using the least squares fit.

The high level bit is that PCA is useful when there is a set of latent (hidden/underlying) variables, and all the coordinates of your data are linear combinations (plus noise) of those variables. The least squares fit is useful when you have direct access to the independent variables, so any noisy coordinates are linear combinations (plus noise) of known variables.

5.1 Slope Functions [5 pts] **[W]**

In the **following cell**, complete the following:

1. **pca_slope**: For this function, assume that X is the first feature and Y is the second feature for the data. Write a function, that takes in the first feature vector X and the second feature vector Y. Stack these two feature vectors into a single Nx2 matrix and use this to determine the first principal component vector of this dataset. Finally, return the slope of this first component. You should use the PCA implementation from Q2.

2. **lr_slope**: Write a function that takes X and y and returns the slope of the least squares fit. You should use the Linear Regression implementation from Q3 but do not use any kind of regularization. Think about how weight could relate to slope.

In later subparts, we consider the case where our data consists of noisy measurements of x and y. For each part, we will evaluate the quality of the relationship recovered by PCA, and that recovered by standard least squares regression.

As a reminder, least squares regression minimizes the squared error of the dependent variable from its prediction. Namely, given (x_i, y_i) pairs, least squares returns the line $l(x)$ that minimizes $\sum_i (y_i - l(x_i))^2$.

In [ ]:
```
import numpy as np
from pca import PCA
from regression import Regression

def pca_slope(X, y):
    """
    Calculates the slope of the first principal component given by PCA

    Args:
        x: N x 1 array of feature x
        y: N x 1 array of feature y
    Return:
        slope: (float) scalar slope of the first principal component
    """

    # raise NotImplementedError



def lr_slope(X, y):
    """
    Calculates the slope of the best fit returned by linear_fit_closed()

    For this function don't use any regularization

    Args:
        X: N x 1 array corresponding to a dataset
        y: N x 1 array of labels y
    Return:
        slope: (float) slope of the best fit
    """

    # raise NotImplementedError
```

We will consider a simple example with two variables, x and y, where the true relationship between the variables is $y = 4x$. Our goal is to recover this relationship—namely, recover the coefficient "4". We set $X = [0, .02, .04, .06, \ldots, 1]$ and $y = 4x$. Make sure both functions return 4.

In [ ]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
x = np.arange(0, 1.02, 0.02).reshape(-1, 1)

y = 4 * np.arange(0, 1.02, 0.02).reshape(-1, 1)

print("Slope of first principal component", pca_slope(x, y))

print("Slope of best linear fit", lr_slope(x, y))

fig = plt.figure()
plt.scatter(x, y)
plt.xlabel("x")
plt.ylabel("y")

if not STUDENT_VERSION:
    fig.text(0.5, 0.5, EO_TEXT, transform=fig.transFigure,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA, fontname=EO_FONT,
        ha='center', va='center', rotation=EO_ROT*0.8)

plt.show()
```

5.2 Analysis Setup [5 pts] **[W]**

Error in y

In this subpart, we consider the setting where our data consists of the actual values of x, and noisy estimates of y. Run the following cell to see how the data looks when there is error in y.

In [ ]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
base = np.arange(0.001, 1.001, 0.001).reshape(-1, 1)
c = 0.5
X = base
y = 4 * base + np.random.normal(loc=[0], scale=c, size=base.shape)

fig = plt.figure()
plt.scatter(X, y)
plt.xlabel("x")
plt.ylabel("y")

if not STUDENT_VERSION:
    fig.text(0.5, 0.5, EO_TEXT, transform=fig.transFigure,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA, fontname=EO_FONT,
        ha='center', va='center', rotation=EO_ROT)

plt.show()
```

In **following cell**, you will implement the **addNoise** function:

1. Create a vector X where $X = [x_1, x_2, \ldots, x_{1000}] = [.001, .002, .003, \ldots, 1]$.

2. For a given noise level c, set $\hat{y}_i \sim 4x_i + \mathcal{N}(0, c) = 4i/1000 + \mathcal{N}(0, c)$, and $\hat{Y} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{1000}]$. You can use the np.random.normal function, where scale is equal to noise level, to add noise to your points.

3. Notice the parameter **x_noise** in the **addNoise** function. When this parameter is set to $True$, you will have to add noise to X. For a given noise level c, let $\hat{x}_i \sim x_i + \mathcal{N}(0, c) = i/1000 + \mathcal{N}(0, c)$, and $\hat{X} = [\hat{x}_1, \hat{x}_2, \ldots . \hat{x}_{1000}]$

4. Return the **pca_slope** and **lr_slope** values of this X and $\hat{Y}$ dataset you have created where $\hat{Y}$ has noise ($X = X$ or $\hat{X}$ depending on the problem).

Hint 1: Refer to the above example on how to add noise to X or Y

Hint 2: Be careful not to add double noise to X or Y

In [ ]:
```
def addNoise(c, x_noise = False, seed = 1):
    """
    Creates a dataset with noise and calculates the slope of the dataset
    using the pca_slope and lr_slope functions implemented in this class.

    Args:
        c: (float) scalar, a given noise level to be used on Y and/or X
        x_noise: (Boolean) When set to False, X should not have noise added
                 When set to True, X should have noise.
                 Note that the noise added to X should be different from the
                 noise added to Y. You should NOT use the same noise you add
                 to Y here.
        seed: (int) Random seed
    Return:
        pca_slope_value: (float) slope value of dataset created using pca_slope
        lr_slope_value: (float) slope value of dataset created using lr_slope

    """
    np.random.seed(seed) #### DO NOT CHANGE THIS ####
    ############## START YOUR CODE BELOW #############
```

A scatter plot with c on the horizontal axis and the output of **pca_slope** and **lr_slope** on the vertical axis has already been implemented for you.

A sample $\hat{Y}$ has been taken for each c in $[0, 0.05, 0.1, \ldots, 0.95, 1.0]$. The output of **pca_slope** is plotted as a red dot, and the output of **lr_slope** as a blue dot. This has been repeated 30 times, you can see that we end up with a plot of 1260 dots, in 21 columns of 60, half red and half blue.

Note: Here, `x_noise = False` since we only want Y to have noise.

In [ ]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
pca_slope_values = []
```

```
linreg_slope_values = []
c_values = []
s_idx = 0

for i in range(30):
    for c in np.arange(0, 1.05, 0.05):

        # Calculate pca_slope_value (psv) and lr_slope_value (lsv)
        psv, lsv = addNoise(c, seed = s_idx)

        # Append pca and lr slope values to list for plot function
        pca_slope_values.append(psv)
        linreg_slope_values.append(lsv)

        # Append c value to list for plot function
        c_values.append(c)

        # Increment random seed index
        s_idx += 1

fig = plt.figure()
plt.scatter(c_values, pca_slope_values, c='r')
plt.scatter(c_values, linreg_slope_values, c='b')
plt.xlabel("c")
plt.ylabel("slope")

if not STUDENT_VERSION:
    fig.text(0.6, 0.4, EO_TEXT, transform=fig.transFigure,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.5, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)

plt.show()
```

Error in x and y

We will now examine the case where our data consists of noisy estimates of **both** x and y. Run the following cell to see how the data looks when there is error in both.

In [ ]:

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
base = np.arange(0.001, 1, 0.001).reshape(-1, 1)
c = 0.5
X = base + np.random.normal(loc=[0], scale=c, size=base.shape)
y = 4 * base + np.random.normal(loc=[0], scale=c, size=base.shape)

fig = plt.figure()
plt.scatter(X, y)
plt.xlabel("x")
plt.ylabel("y")

if not STUDENT_VERSION:
    fig.text(0.5, 0.5, EO_TEXT, transform=fig.transFigure,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.8, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)

plt.show()
```

In the below cell, we graph the predicted PCA and LR slopes on the vertical axis against the value of c on the horizontal axis.

In [ ]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
pca_slope_values = []
linreg_slope_values = []
c_values = []
s_idx = 0

for i in range(30):
    for c in np.arange(0, 1.05, 0.05):

        # Calculate pca_slope_value (psv) and lr_slope_value (lsv), notice x_noi
        psv, lsv = addNoise(c, x_noise = True, seed = s_idx)

        # Append pca and lr slope values to list for plot function
        pca_slope_values.append(psv)
        linreg_slope_values.append(lsv)

        # Append c value to list for plot function
        c_values.append(c)

        # Increment random seed index
        s_idx += 1

fig = plt.figure()
plt.scatter(c_values, pca_slope_values, c='r')
plt.scatter(c_values, linreg_slope_values, c='b')
plt.xlabel("c")
plt.ylabel("slope")

if not STUDENT_VERSION:
    fig.text(0.5, 0.5, EO_TEXT, transform=fig.transFigure,
        fontsize=EO_SIZE/2, color=EO_COLOR, alpha=EO_ALPHA*0.5, fontname=EO_FONT
        ha='center', va='center', rotation=EO_ROT)

plt.show()
```

5.3. Analysis [5 pts] **[W]**

Based on your observations from previous subsections answer the following questions about the two cases (error in Y and error in both X and Y) in 2-3 lines.

Note:

1. The closer the value of slope to actual slope ("4" here) the better the algorithm is performing.
2. You don't need to provide a mathematical proof for this question.

Questions:

1. Which case does PCA perform worse in? Why does PCA perform worse in this case? (2 Pts)
2. Why does PCA perform better in the other case? (1 Pt)

3. Which case does Linear Regression perform well? Why does Linear Regression perform well in this case? (2 Pts)

6 Feature Reduction Implementation [25 Points Bonus for All] **[P]** | **[W]**

6.1 Implementation [18 Points] **[P]**

Feature selection is an integral aspect of machine learning. It is the process of selecting a subset of relevant features that are to be used as the input for the machine learning task. Feature selection may lead to simpler models for easier interpretation, shorter training times, avoidance of the curse of dimensionality, and better generalization by reducing overfitting.

In the **feature_reduction.py** file, complete the following functions:

- **forward_selection**
- **backward_elimination**

These functions should each output a list of features.

Forward Selection:

In forward selection, we start with a null model, start fitting the model with one individual feature at a time, and select the feature with the minimum p-value. We continue to do this until we have a set of features where one feature's p-value is less than the confidence level.

Steps to implement it:

1. Choose a significance level (given to you).
2. Fit all possible simple regression models by considering one feature at a time.
3. Select the feature with the lowest p-value.
4. Fit all possible models with one extra feature added to the previously selected feature(s).
5. Select the feature with the minimum p-value again. if p_value < significance, go to Step 4. Otherwise, terminate.

Backward Elimination:

In backward elimination, we start with a full model, and then remove the insignificant feature with the highest p-value (that is greater than the significance level). We continue to do this until we have a final set of significant features.

Steps to implement it:

1. Choose a significance level (given to you).
2. Fit a full model including all the features.
3. Select the feature with the highest p-value. If (p-value > significance level), go to Step 4, otherwise terminate.
4. Remove the feature under consideration.

5. Fit a model without this feature. Repeat entire process from Step 3 onwards.

TIP 1: The p-value is known as the observed significance value for a null hypothesis. In our case, the p-value of a feature is associated with the hypothesis H_0: $\beta_j = 0$. If $\beta_j = 0$, then this feature contributes no predictive power to our model and should be dropped. We reject the null hypothesis if the p-value is smaller than our significance level. Some more information about p-values can be found here: https://towardsdatascience.com/what-is-a-p-value-b9e6c207247f

TIP 2: For this function, you will have to install statsmodels if not installed already. To do this, run `pip install statsmodels` in command line/terminal. In the case that you are using an Anaconda environment, run `conda install -c conda-forge statsmodels` in the command line/terminal. For more information about installation, refer to https://www.statsmodels.org/stable/install.html. The statsmodels library is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. You will have to use this library to choose a regression model to fit your data against. Some more information about this module can be found here: https://www.statsmodels.org/stable/index.html

TIP 3: For step 2 in each of the forward and backward selection functions, you can use the `sm.OLS` function as your regression model. Also, do not forget to add a bias to your regression model. A function that may help you is the `sm.add_constants` function.

TIP 4: You should be able to implement these function using only the libraries provided in the cell below.

In [ ]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################

from feature_reduction import FeatureReduction
```

In [ ]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
boston = load_boston()
bos = pd.DataFrame(boston.data, columns = boston.feature_names)
bos['Price'] = boston.target
X = bos.drop("Price", 1)       # feature matrix
y = bos['Price']               # target feature
featureselection = FeatureReduction()
#Run the functions to make sure two lists are generated, one for each method
print("Features selected by forward selection:", FeatureReduction.forward_select
print("Features selected by backward elimination:", FeatureReduction.backward_el
```

6.2 Feature Selection - Discussion [7pts] **[W]**

Question 6.2.1:

We have seen two regression methods namely Lasso and Ridge regression earlier in this assignment. Another extremely important and common use-case of these methods is to

perform feature selection. According to you, which of these two methods are more appropriate for feature selection? Why? **(3 pts)**

Answer

...

Question 6.2.2:

We have seen that we use different subsets of features to get different regression models. These models depend on the relevant features that we have selected. Using forward selection, what fraction of the total possible models can we explore? Assume that the total number of features that we have at our disposal is N. Remember that in stepwise feature selection (like forward selection and backward elimination), we always include an intercept in our model, so you only need to consider the N features. **(4 pts)**

Answer

...

In [ ]: