I have chosen to answer Questions 3 and 4, in addition to the mandatory Question 1

Question 1

A company has found that the widgets they manufacture contain some mysterious radioactive material. The company produced 100 widgets in 35 days before they realised the widgets contained radioactive material. Then it took 14 days before an investigation was started. The investigation took 90 days. During the investigation, a company worker would visit the location where a widget was shipped to, and measured the radioactive material in the widget. Only one measurement of the radioactive material was performed per widget.

The company has asked you whether you can infer the rate of radioactive decay of the material in the widgets. They know that the amount of radioactive material N at any time t is

$$N = N_0 \exp\left(-\alpha\left[t - t_0\right]\right)$$

where N is measured in grams and t is measured in seconds. They also know that the detectors they use to measure the amount of radioactive material reports uncertainties that are normally distributed.

The company does not know how much radioactive material was initially put into each widget, but they know that the amount is different for each widget. However, they do know that the initial amount of radioactive material must be less than 20 grams in every widget.

Unfortunately, the company also does not know when each widget was made. All they can tell you is that all 100 widgets were made in 35 days. You can see how this careless operation has led to such a problematic situation!

Task 1

Infer the decay rate α for the mysterious radioactive material.

One can consider this to be a hierarchical Bayesian problem where the decay lifetime of the radioactive material is given by the mathematical relationship described above. Since we seek to infer the decay rate α, one can assume the following priors on the amount of initial material N_0 and the time of manufacture t_0 for the i-th widget is:

$$N_{0,i} \sim U(0, 20 grams)$$

$$t_{0,i} \sim U(0, 35 days)$$

In addition to this, one can assume a uniform prior on α of the following form: $\alpha \sim U\left(0, 10^{-6}\right)$ while noting that the measured radioactive material has uncertainties that are normally distributed: $N_{1,i} \sim N\left(N_{1,i}, \sigma_{N_i}\right)$ The probabilistic graphical model for

this problem has been included in the handwritten notes accompanying this assignment

One can construct a probabilistic model with pystan in order to infer the decay rate, α of the radioactive material as follows

```
$$import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
N_widgets = 100
N_initial_max = 20
data = pd.read_csv('decay.csv')
t_measured = np.array(data['time_measured_in_seconds'])
days_to_seconds = 24 * 60 * 60
t_initial_max = 35 * days_to_seconds
N_measured = np.array(data['grams_measured'])
sigma_N_measured = np.array(data['uncertainty_in_grams_measured'])
alpha = np.random.uniform()*1e-6

import os
import logging
import pickle
import arviz as az
import pystan as stan

def load_stan_model(path, cached_path=None, recompile=False,
overwrite=True,
    verbose=True):
    r"""
    Load a Stan model from a file. If a cached file exists, use it by
default.

    :param path:
        The path of the Stan model.

    :param cached_path: [optional]
        The path of the cached Stan model. By default this will be the
same  as
        :path:, with a `.cached` extension appended.

    :param recompile: [optional]
        Recompile the model instead of using a cached version. If the
cached
        version is different from the version in path, the model will
be
        recompiled automatically.
    """

    cached_path = cached_path or "{}.cached".format(path)
```

```
    with open(path, "r") as fp:
        model_code = fp.read()

    while os.path.exists(cached_path) and not recompile:
        with open(cached_path, "rb") as fp:
            model = pickle.load(fp)

        if model.model_code != model_code:
            if verbose:
                logging.warn("Cached model at {} differs from the code
in {}; "\
                             "recompiling model".format(cached_path,
path))
            recompile = True
            continue

        else:
            if verbose:
                logging.info("Using pre-compiled model from
{}".format(cached_path))
            break

    else:
        model = stan.StanModel(model_code=model_code)

        # Save the compiled model.
        if not os.path.exists(cached_path) or overwrite:
            with open(cached_path, "wb") as fp:
                pickle.dump(model, fp)


    return model


def sampling_kwds(**kwargs):
    r"""
    Prepare a dictionary that can be passed to Stan at the sampling
stage.
    Basically this just prepares the initial positions so that they
match the
    number of chains.
    """

    kwds = dict(chains=4)
    kwds.update(kwargs)

    if "init" in kwds:
        kwds["init"] = [kwds["init"]] * kwds["chains"]
```

```
    return kwds

model = load_stan_model("decay.stan")

# Data.
data_dict = dict(
    N_widgets=N_widgets,
    t_initial_max=t_initial_max,
    t_measured=t_measured,
    N_measured=N_measured,
    sigma_N_measured=sigma_N_measured,
    N_initial_max=N_initial_max,
)

init_dict = dict(alpha=alpha)

# Run optimisation.
opt_stan = model.optimizing(
    data=data_dict,
    init=init_dict
)

# Run sampling.
samples = model.sampling(**sampling_kwds(
    chains=2,
    iter=2000,
    data=data_dict,
    init=opt_stan
))


fig = samples.traceplot(("alpha", ))

# Draw true value for comparison.
display(az.summary(samples,round_to=10))
```

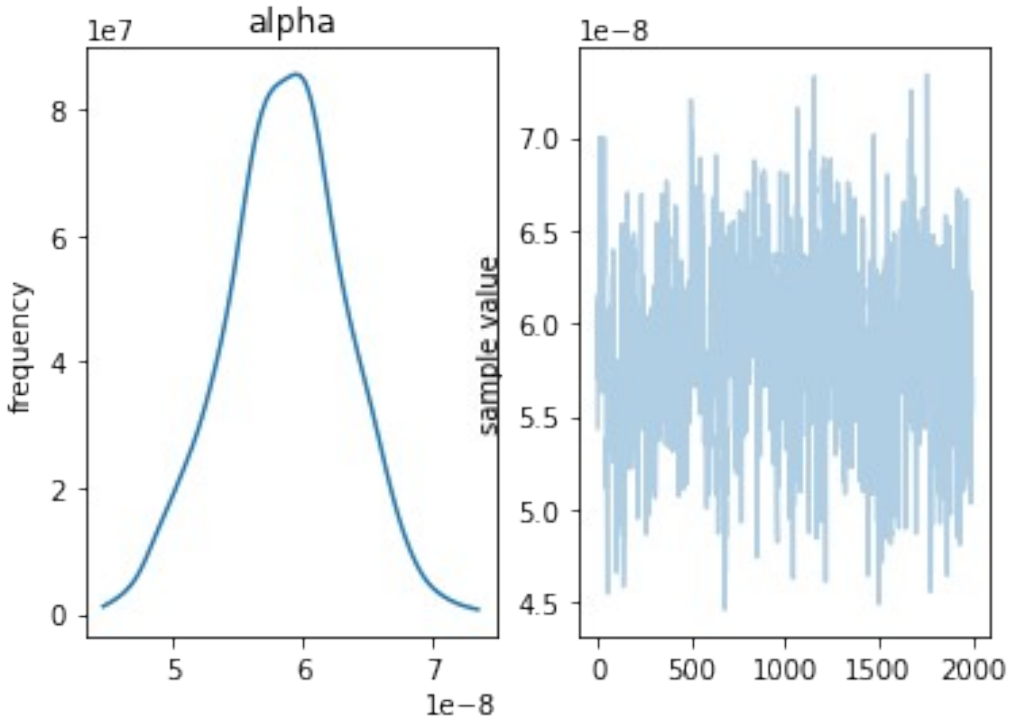
```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL
anon_model_f2f173f8348b412e1d2de93aa90dd577 NOW.
WARNING:pystan:Deprecation warning. PyStan plotting deprecated, use
ArviZ library (Python 3.5+). `pip install arviz`;
`arviz.plot_trace(fit)`)
```

	mean	sd	hdi_3%	hdi_97%
alpha	5.860000e-08	4.700000e-09	4.950000e-08	6.720000e-08

N_initial[0]	5.597593e+00	1.909176e+00	1.947543e+00	9.117121e+00
N_initial[1]	6.484752e+00	1.255203e+00	4.262643e+00	8.887671e+00
N_initial[2]	1.801424e+01	1.424589e+00	1.535676e+01	1.998751e+01
N_initial[3]	9.338789e+00	9.094292e-01	7.541379e+00	1.097231e+01
...	...	...	...	...
t_initial[95]	1.469162e+06	8.686711e+05	4.676332e+04	2.852651e+06
t_initial[96]	1.428557e+06	8.789972e+05	4.015114e+04	2.862905e+06
t_initial[97]	1.731879e+06	8.387565e+05	2.939469e+05	3.023540e+06
t_initial[98]	1.460072e+06	8.589448e+05	1.325739e+03	2.805546e+06
t_initial[99]	1.443012e+06	8.626526e+05	5.826047e+03	2.801744e+06

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha	3.000000e-10	2.000000e-10	278.194788	524.851691	1.003937
N_initial[0]	3.421170e-02	2.419359e-02	2973.214278	1023.106231	1.000814
N_initial[1]	2.316436e-02	1.638130e-02	3031.932595	1072.370875	1.003112
N_initial[2]	2.721456e-02	1.924565e-02	2241.988975	1199.188495	1.001282
N_initial[3]	1.837452e-02	1.315367e-02	2490.610784	1347.909080	1.000202
...	...	...	...	...	...
t_initial[95]	1.292842e+04	1.160257e+04	4073.659816	1070.383634	1.000034
t_initial[96]	2.038950e+04	1.441982e+04	1929.744468	1390.257449	1.002142
t_initial[97]	1.487035e+04	1.080126e+04	3015.876035	1241.781712	0.999222
t_initial[98]	1.429300e+04	1.159566e+04	3202.253124	1092.359646	1.001558
t_initial[99]	1.681578e+04	1.212766e+04	2499.412195	1529.218577	1.001758

[201 rows x 9 columns]



From the above summary, it is evident that the decay rate, α has a value of 5.7×10^{-8} units

Task 2

The company is delighted with your work. However, there is a problem. The investigation team used three different detectors to measure the amount of radioactive material in each widget. Each detector costs $1,000,000, and they are suspicious that one of those detectors is giving biased measurements. They need you to infer which detector is giving biased measurements, and to infer what the level of that bias is. After accounting for the biases, have your inferences on α changed?

One can regard this as a similar hierarchical Bayesian problem to task 1 with one minor alteration, namely the addition of a bias term, which represents the level of bias in the measurement of each detector. This model can be represented as follows:

$$N_t = N_0 \exp\left(\alpha \left[t_1 - t_0\right]\right) + B_a$$

where the term B_a has been added to account for the biases of each of the three detectors (i.e. a can take values of 0, 1, and 2, depending on the detector of interest. One can assume a uniform prior on B_a of the form: $B_a \sim U(-20, 20)$ since the initial amount of radioactive material in each widget is known to be less than 20 grams. Once again, one can construct a similar model to that in task 1 while accounting for the biases associated with each detector in order to make inferences on which of the detectors is biased

```
model = load_stan_model("decayBiased.stan")

# Data.
data_dict = dict(
    n_det = n_det,
    detector_bias = detector_name,
    N_widgets=N_widgets,
    t_initial_max=t_initial_max,
    t_measured=t_measured,
    N_measured=N_measured,
    sigma_N_measured=sigma_N_measured,
    N_initial_max=N_initial_max,
)

init_dict = dict(alpha=alpha)

# Run optimisation.
opt_stan = model.optimizing(
    data=data_dict,
    init=init_dict
  )




# Run sampling.
samples = model.sampling(**sampling_kwds(
      chains=2,
      iter=2000,
      data=data_dict,
      init=opt_stan
  ))
fig = samples.traceplot(("alpha", ))

# Draw true value for comparison.
display(az.summary(samples,round_to=10))
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL
anon_model_f455e07ed23fcaba175b1482c7b9aafe NOW.

0     2
1     1
2     2
3     3
4     3
     ..
95    3
96    3
```

```
97    2
98    1
99    3
Name: detector_name, Length: 100, dtype: int64
```

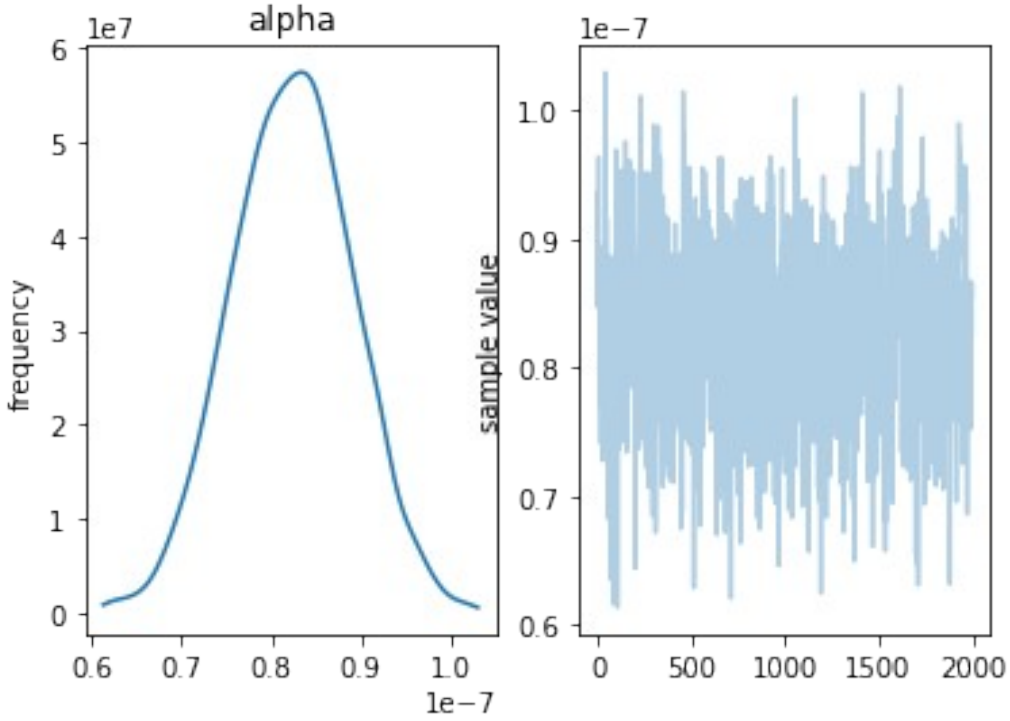
```
WARNING:pystan:Deprecation warning. PyStan plotting deprecated, use
ArviZ library (Python 3.5+). `pip install arviz`;
`arviz.plot_trace(fit)`)
```

	mean	sd	hdi_3%	hdi_97%
alpha	8.230000e-08	6.700000e-09	6.900000e-08	9.380000e-08
N_initial[0]	2.682482e+00	1.676223e+00	4.915353e-02	5.639496e+00
N_initial[1]	5.788699e+00	1.382635e+00	3.369651e+00	8.561247e+00
N_initial[2]	1.652300e+01	2.138613e+00	1.290755e+01	1.996266e+01
N_initial[3]	9.710258e+00	1.162062e+00	7.547819e+00	1.184695e+01
...	...	...	...	...
t_initial[98]	1.446327e+06	8.845554e+05	2.750159e+04	2.818187e+06
t_initial[99]	1.454841e+06	8.894153e+05	8.169483e+03	2.854447e+06
bias[0]	9.131267e-01	3.259298e-01	2.001629e-01	1.419541e+00
bias[1]	3.166550e+00	4.537543e-01	2.350767e+00	3.996240e+00
bias[2]	2.512660e-01	3.867631e-01	-4.510196e-01	9.335476e-01

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha	3.000000e-10	2.000000e-10	380.234654	761.716657	1.004683
N_initial[0]	3.584922e-02	2.547082e-02	1867.128810	1065.091384	1.001828
N_initial[1]	2.631586e-02	1.878753e-02	2745.728437	1495.004686	1.000230
N_initial[2]	5.305370e-02	3.912943e-02	1316.535102	749.252297	1.001686
N_initial[3]	3.593208e-02	2.565811e-02	1057.753565	1461.587106	1.001060
...	...	...	...	...	...
t_initial[98]	1.287037e+04	1.115953e+04	4716.641540	1310.593744	

```
0.999426
t_initial[99]  1.445122e+04  1.141363e+04  3784.038703  1612.815555
1.000273
bias[0]        1.293388e-02  9.149839e-03   673.468476   931.248000
1.005392
bias[1]        2.392292e-02  1.692979e-02   360.641485   579.381949
1.006118
bias[2]        2.118263e-02  1.499150e-02   349.184727   549.059869
1.004217

[204 rows x 9 columns]
```



From the summary generated above, it is evident that detector 2 (i.e. bias[1]) has the largest R_hat value, which indicates that it is the source of the biased measurements. One can also note that this has caused the inference on the value of α to increase in value from 5.7×10^{-8} to 8.23×10^{-8}

Question 3

The following files contain images of hand-written digits:

- training set images
- test set images

YOUR FIRST TASK

You will recall that an autoencoder is a neural network with a specific structure. Your task is to train an autoencoder using these images. Specify the network architecture in as much detail as you would explain a generative model.

Demonstrate that the latent space of your autoencoder smoothly represents digits by inputting the test-set images and recording the latent space values found for each image. Plot the latent space values coloured by the known test set labels.

An autoencoder is a dimensionality reduction technique that does not require labelled data for the training set (in other words, it is an unsupervised technique) which aims to reconstruct the training set inputs, while learning the salient features of the dataset in the process (i.e. during the "bottleneck" phase). Since autoencoders are able to learn non-linear mappings, this implies that this is a non-linear dimensionality reduction technique

Training the autoencoder results in two separable neural networks, namely the encoder and the decoder, which are responsible for mapping the data to and from the low-dimensional bottleneck to the input and output layers respectively. In this problem, I have chosen to train a variational autoencoder (VAE) on the MNIST dataset, which contains images of handwritten digits.This choice was motivated by the presence of multiple classes of objects (i.e.a set of 10 digits). With the implementation of such an autoencoder, one must note that the latent space is now made to be continuous, thereby allowing for more straightforward interpolation and sampling between groups of objects in the training set. Two steps must be followed in order to achieve this, namely:

- Altering the objective function to include a term that includes the Kullback-Leibler divergence of the latent space z to a standard normal distribution

- Having the encoder output two vectors of length n, containing the means μ and standard deviations σ respectively

Both of the aforementioned steps have been accounted for in the below implementation of the VAE, which has been completed using Python's keras package, which provides a Python interface for artificial neural networks. Additionally,the latent space is deemed to be two-dimensional and the majority of the neurons in the encoder and decoder are deemed to have RELU activation functions. The output of the decoder, on the other hand is set to have a sigmoid activation function, which ensures that the outputs assume a continuum between 0 and 1.

YOUR SECOND TASK

Is to use the autoencoder to generate new images of digits that have not been drawn by a human! Create a grid of latent space values. At each grid point, use the decoder to make a prediction of an image. Show that image on a grid point. Repeat this process until you have smoothly mapped out (two dimensions of) the latent space.

```
# Implementation referenced from:
https://keras.io/examples/generative/vae/
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

class Sampling(layers.Layer):


    def func(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        eps = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * eps

#Build the encoder
latent_dim = 2
enc_inputs = keras.Input(shape=(28,28,1))
x = layers.Conv2D(32, 3, activation="relu", strides=2, padding="same")
(enc_inputs)
x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")
(x)
x = layers.Flatten()(x)
x = layers.Dense(16, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
z = Sampling()([z_mean, z_log_var])
encoder = keras.Model(enc_inputs, [z_mean, z_log_var, z],
name="encoder")



#Build the decoder
latent_inputs = keras.Input(shape=(latent_dim,))
x = layers.Dense(7 * 7 * 64, activation="relu")(latent_inputs)
x = layers.Reshape((7, 7, 64))(x)
x = layers.Conv2DTranspose(64, 3, activation="relu", strides=2,
padding="same")(x)
x = layers.Conv2DTranspose(32, 3, activation="relu", strides=2,
padding="same")(x)
dec_outputs = layers.Conv2DTranspose(1, 3, activation="sigmoid",
```

```
padding="same")(x)
decoder = keras.Model(latent_inputs, dec_outputs, name="decoder")



#Create a class for the Variational Autoencoder wherein methods and
functions specific to Var_AEs can be initialised
class Var_AE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(Var_AE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.total_loss_tracker =
keras.metrics.Mean(name="total_loss")
        self.reconstruction_loss_tracker = keras.metrics.Mean(
            name="reconstruction_loss"
        )
        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss")

    @property
    def metrics(self):
        return [
            self.total_loss_tracker,
            self.reconstruction_loss_tracker,
            self.kl_loss_tracker,
        ]

    def train_step(self, data):
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            reconstruction_loss = tf.reduce_mean(
                tf.reduce_sum(
                    keras.losses.binary_crossentropy(data,
reconstruction), axis=(1, 2)
                )
            )
            kl_loss = -0.5 * (1 + z_log_var - tf.square(z_mean) -
tf.exp(z_log_var))
            kl_loss = tf.reduce_mean(tf.reduce_sum(kl_loss, axis=1))
            total_loss = reconstruction_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads,
self.trainable_weights))
        self.total_loss_tracker.update_state(total_loss)

self.reconstruction_loss_tracker.update_state(reconstruction_loss)
        self.kl_loss_tracker.update_state(kl_loss)
```

```
        return {
            "loss": self.total_loss_tracker.result(),
            "reconstruction_loss":
self.reconstruction_loss_tracker.result(),
            "kl_loss": self.kl_loss_tracker.result(),
        }

#Train the Var_AE on the MNIST data
(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()
mnist_digits = np.concatenate([x_train, x_test], axis=0)
mnist_digits = np.expand_dims(mnist_digits, -1).astype("float32") /
255

Var_AE = Var_AE(encoder, decoder)
Var_AE.compile(optimizer=keras.optimizers.Adam())
Var_AE.fit(mnist_digits, epochs=10, batch_size=32)
```

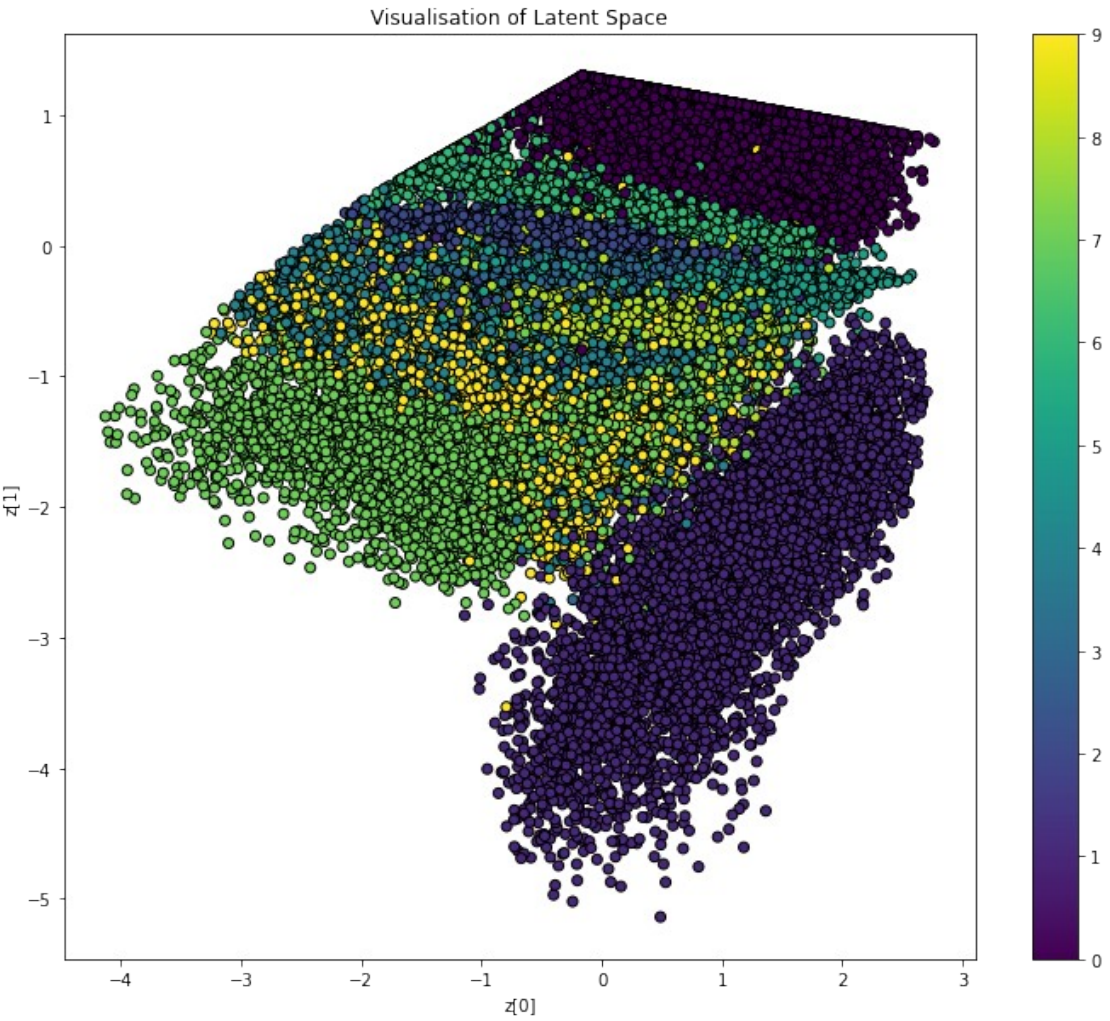
```
Epoch 1/10
2188/2188 [==============================] - 44s 20ms/step - loss:
216.9139 - reconstruction_loss: 192.2159 - kl_loss: 3.1729
Epoch 2/10
2188/2188 [==============================] - 44s 20ms/step - loss:
176.5521 - reconstruction_loss: 164.8253 - kl_loss: 4.9743
Epoch 3/10
2188/2188 [==============================] - 47s 22ms/step - loss:
160.0564 - reconstruction_loss: 153.1159 - kl_loss: 5.8250
Epoch 4/10
2188/2188 [==============================] - 48s 22ms/step - loss:
156.6697 - reconstruction_loss: 149.9764 - kl_loss: 6.0247
Epoch 5/10
2188/2188 [==============================] - 56s 25ms/step - loss:
154.7286 - reconstruction_loss: 148.2165 - kl_loss: 6.1177
Epoch 6/10
2188/2188 [==============================] - 55s 25ms/step - loss:
152.7240 - reconstruction_loss: 147.0674 - kl_loss: 6.1717
Epoch 7/10
2188/2188 [==============================] - 52s 24ms/step - loss:
152.5048 - reconstruction_loss: 146.3255 - kl_loss: 6.2203
Epoch 8/10
2188/2188 [==============================] - 55s 25ms/step - loss:
151.9506 - reconstruction_loss: 145.5857 - kl_loss: 6.2543
Epoch 9/10
2188/2188 [==============================] - 55s 25ms/step - loss:
151.4412 - reconstruction_loss: 145.0059 - kl_loss: 6.2739
Epoch 10/10
2188/2188 [==============================] - 56s 25ms/step - loss:
150.8958 - reconstruction_loss: 144.6084 - kl_loss: 6.3115

<keras.callbacks.History at 0x7fc4a6389a90>
```

```
import matplotlib.pyplot as plt
def plot_label_clusters(vae, data, labels):
    # display a 2D plot of the digit classes in the latent space
    z_mean, _, _ = vae.encoder.predict(data)
    plt.figure(figsize=(12, 10))
    plt.scatter(z_mean[:, 0], z_mean[:, 1],
c=labels,edgecolors='black')
    plt.colorbar()
    plt.xlabel("z[0]")
    plt.ylabel("z[1]")
    plt.title('Visualisation of Latent Space of MNIST Data')
    plt.show()


(x_train, y_train), _ = keras.datasets.mnist.load_data()
x_train = np.expand_dims(x_train, -1).astype("float32") / 255

plot_label_clusters(vae, x_train, y_train)
```

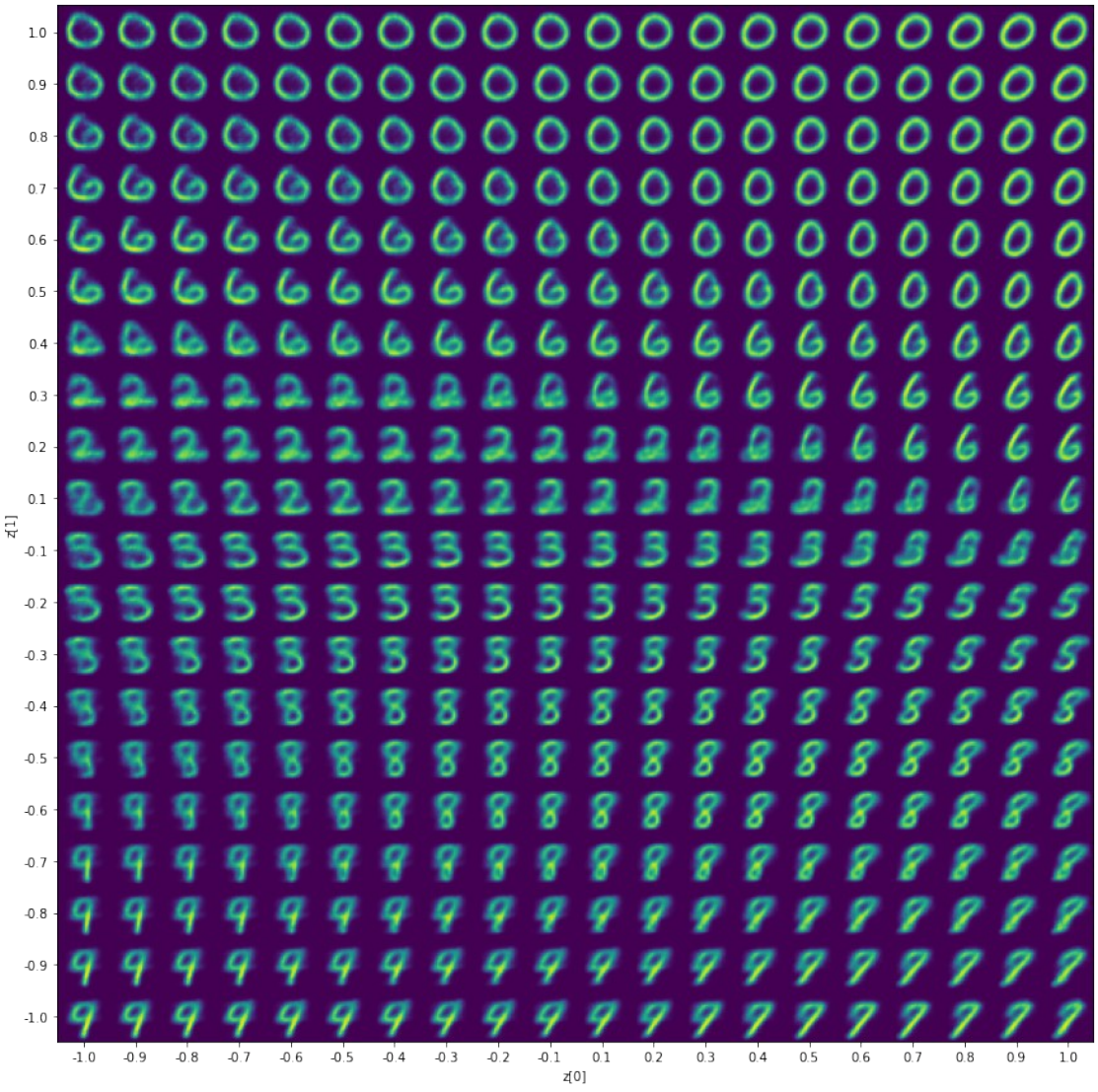


```
def plot_latent_space(vae, n=20, figsize=15):
    # display a n*n 2D plot of digits
    digit_size = 28
    scale = 1.0
    figure = np.zeros((digit_size * n, digit_size * n))
    # linearly spaced coordinates corresponding to the 2D plot
    # of digit classes in the latent space
    grid_x = np.linspace(-scale, scale, n)
    grid_y = np.linspace(-scale, scale, n)[::-1]

    for i, yi in enumerate(grid_y):
        for j, xi in enumerate(grid_x):
            z_sample = np.array([[xi, yi]])
            x_decoded = vae.decoder.predict(z_sample)
            digit = x_decoded[0].reshape(digit_size, digit_size)
            figure[
                i * digit_size : (i + 1) * digit_size,
                j * digit_size : (j + 1) * digit_size,
            ] = digit

    plt.figure(figsize=(figsize, figsize))
    start_range = digit_size // 2
    end_range = n * digit_size + start_range
    pixel_range = np.arange(start_range, end_range, digit_size)
    sample_range_x = np.round(grid_x, 1)
    sample_range_y = np.round(grid_y, 1)
    plt.xticks(pixel_range, sample_range_x)
    plt.yticks(pixel_range, sample_range_y)
    plt.xlabel("z[0]")
    plt.ylabel("z[1]")
    plt.imshow(figure)
    plt.show()


plot_latent_space(vae)
```



Question 4

The dataset (download here; 200 MB) contains 25,753 coloured images (g, r, and i band) each 69 x 69 pixels in size. There are 10 classes of galaxies: Full dataset (25753 images)

├── Class 0 (3461 images): Disk, Face-on, No Spiral

├── Class 1 (6997 images): Smooth, Completely round

├── Class 2 (6292 images): Smooth, in-between round

├── Class 3 (394 images): Smooth, Cigar shaped

├── Class 4 (3060 images): Disk, Edge-on, Rounded Bulge

├── Class 5 (17 images): Disk, Edge-on, Boxy Bulge

├── Class 6 (1089 images): Disk, Edge-on, No Bulge

├── Class 7 (1932 images): Disk, Face-on, Tight Spiral

├── Class 8 (1466 images): Disk, Face-on, Medium Spiral

└── Class 9 (1045 images): Disk, Face-on, Loose Spiral

You will need h5py to load these data

Train five classifiers (different techniques) to classify galaxies. You can choose these techniques, but you should choose them to be appropriate for the problem and suitable to give good results on this data set. Ensure that you split your data set appropriately to train, validation (where appropriate), and test.

Evaluate these classification techniques and make a recommendation for which is most suitable. As part of evaluating performance, you should consider and discuss things like precision, recall, confusion matrices, biased/unbiased training sets, and cross-validation.

Five different classifiers from the scikit-learn package were trained in order to classify the aforementioned images into one of ten different classes, whose labels are represented by integers ranging from 0 to 9 as indicated above. The chosen classifiers were deemed to be appropriate for the classification of images, and were as follows:

1. Random Forest Classifier
2. Decision Tree Classifier
3. Gaussian Naive Bayes Classifier
4. K Nearest Neighbours Classifier
5. SGD Classifier **A classification report and a confusion matrix have been produced for each of the five classifiers. The classification report provides a more detailed overview of the process followed by each algorithm to categorise the images. It includes information on metrics such as the precision, recall, and f1 score of each classification made by the model. In addition to this, it includes a measure of the accuracy of the model. The confusion matrix provides a class-by-class breakdown of correct and incorrect classifications made by each model. Based on these metrics, one can determine the most appropriate model that can be used to classify the images of these galaxies. It should be noted that the data had been split into training and test sets in an 85-15 ratio (i.e. 85% of the data was used for training the models, while the remaining 15% was used as the test set). This ratio was kept consistent for all five models**

```
import h5py
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
```

```
with h5py.File('/Users/admzlm4/Desktop/Galaxy10.h5', 'r') as fp:
    images = np.array(fp['images'])
    labels = np.array(fp['ans'])
X_train, X_test, y_train, y_test = train_test_split(images, labels,
test_size = 0.15)

#Normalise the data
X_train = X_train/255.0
X_test = X_test/255.0

nsamples, nx, ny, nz = X_train.shape

X_train_clf = X_train.reshape((nsamples, nx*ny*nz))


nsamples, nx, ny, nz = X_test.shape
X_test_clf = X_test.reshape((nsamples, nx*ny*nz))


# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train_clf, y_train)
y_pred = model.predict(X_test_clf)

print(accuracy_score(y_pred, y_test))
print(classification_report(y_pred, y_test))
print(confusion_matrix(y_pred, y_test))
print(cross_val_score(model, X_train_clf, y_train, cv=2))
```

```
0.6884944920440637
              precision    recall  f1-score   support

           0       0.47      0.41      0.44       579
           1       0.92      0.78      0.85      1259
           2       0.77      0.74      0.75       981
           3       0.06      0.50      0.10         6
           4       0.82      0.75      0.78       259
           5       0.00      0.00      0.00         0
           6       0.81      0.70      0.75        93
           7       0.23      0.41      0.29        81
           8       0.01      0.33      0.03         6
           9       0.01      0.25      0.02         4

    accuracy                           0.69      3268
   macro avg       0.41      0.49      0.40      3268
weighted avg       0.76      0.69      0.72      3268
```

```
[[238  45  79   7   5   0   1  64  87  53]
 [114 988 130   0   0   0   0  14   9   4]
 [117  30 727  11  28   0   1  29  17  21]
 [  0   0   0   3   2   0   1   0   0   0]
 [ 13   0   2  23 193   1  12   5   2   8]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   1   3   8   8   1  65   0   0   7]
 [ 22   5   5   0   0   0   0  33  16   0]
 [  2   0   0   0   0   0   0   1   2   1]
 [  2   0   0   0   0   0   0   0   1   1]]
```

```
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/
metrics/_classification.py:1245: UndefinedMetricWarning: Recall and F-
score are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/metri
cs/_classification.py:1245: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/metri
cs/_classification.py:1245: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
[0.67458689 0.67444372]
```

```
# Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train_clf, y_train)
y_pred_dtc = dtc.predict(X_test_clf)
y_pred_dtc
accuracy_score(y_pred_dtc, y_test)
print(classification_report(y_pred_dtc, y_test))
print(confusion_matrix(y_pred_dtc, y_test))
print(cross_val_score(dtc, X_train_clf, y_train, cv=2))
```

```
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/
metrics/_classification.py:1245: UndefinedMetricWarning: Recall and F-
score are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/metri
cs/_classification.py:1245: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/metri
```

```
cs/_classification.py:1245: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

           0       0.36      0.32      0.34       563
           1       0.66      0.69      0.67      1023
           2       0.58      0.58      0.58       958
           3       0.12      0.12      0.12        50
           4       0.53      0.58      0.55       216
           5       0.00      0.00      0.00         0
           6       0.69      0.57      0.62        97
           7       0.29      0.26      0.28       165
           8       0.13      0.15      0.14       116
           9       0.14      0.16      0.15        80

    accuracy                           0.52      3268
   macro avg       0.35      0.34      0.34      3268
weighted avg       0.52      0.52      0.52      3268

[[182 127 110   8  20   0   4  39  48  25]
 [ 82 706 186   2  14   0   0  13  15   5]
 [121 180 551  12  29   0   1  26  15  23]
 [  8   0   7   6  19   0   5   2   0   3]
 [ 13   4  38  11 125   2  10   3   2   8]
 [  0   0   0   0   0   0   0   0   0   0]
 [  2   1   0   9  20   0  55   0   0  10]
 [ 42  22  21   1   1   0   2  43  31   2]
 [ 34  22  19   1   1   0   0  16  17   6]
 [ 24   7  14   2   7   0   3   4   6  13]]
[0.48396155 0.46889177]
```

```
# Naive Bayes Classifier
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train_clf, y_train)
y_pred_nb = nb.predict(X_test_clf)
accuracy_score(y_pred_nb, y_test)
print(classification_report(y_pred_nb, y_test))
confusion_matrix(y_pred_nb, y_test)
print(cross_val_score(dtc, X_train_clf, y_train, cv=2))
```

```
              precision    recall  f1-score   support

           0       0.03      0.19      0.05        84
           1       0.13      0.49      0.21       290
           2       0.33      0.48      0.39       652
           3       0.27      0.09      0.14       155
```

```
           4       0.18      0.27      0.22       162
           5       0.00      0.00      0.00        85
           6       0.81      0.10      0.17       678
           7       0.09      0.26      0.13        50
           8       0.54      0.08      0.13       968
           9       0.04      0.03      0.03       144

    accuracy                           0.21      3268
   macro avg       0.24      0.20      0.15      3268
weighted avg       0.43      0.21      0.19      3268

[0.48763365 0.47677684]
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p =
2)
knn.fit(X_train_clf, y_train)
y_pred_knn = knn.predict(X_test_clf)
accuracy_score(y_pred_knn, y_test)
print(classification_report(y_pred_knn, y_test))
print(confusion_matrix(y_pred_knn, y_test))
print(cross_val_score(knn, X_train_clf, y_train, cv=2))
```

```
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/
metrics/_classification.py:1245: UndefinedMetricWarning: Recall and F-
score are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/metri
cs/_classification.py:1245: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/metri
cs/_classification.py:1245: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

           0       0.32      0.43      0.37       375
           1       0.93      0.69      0.79      1427
           2       0.77      0.69      0.73      1055
           3       0.23      0.38      0.29        32
           4       0.70      0.79      0.74       210
           5       0.00      0.00      0.00         0
           6       0.71      0.71      0.71        80
           7       0.12      0.24      0.16        71
           8       0.02      0.21      0.04        14
```

```
           9       0.03      0.75      0.06         4

    accuracy                           0.65      3268
   macro avg       0.38      0.49      0.39      3268
weighted avg       0.76      0.65      0.70      3268

[[162  32  49   3  14   0   1  40  42  32]
 [164 990 162   0  12   0   1  45  42  11]
 [139  42 731  13  26   0   0  38  29  37]
 [  3   0   0  12   5   0   6   2   1   3]
 [  6   1   0  17 165   1  15   2   0   3]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   7  13   1  57   1   0   1]
 [ 29   4   3   0   0   0   0  17  16   2]
 [  5   0   1   0   1   0   0   1   3   3]
 [  0   0   0   0   0   0   0   0   1   3]]
[0.64121395 0.63210197]
```

```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(random_state=42, max_iter=1000, tol=1e-3)
sgd_clf.fit(X_train_clf, y_train)
y_pred_sgd = sgd_clf.predict(X_test_clf)
accuracy_score(y_pred_sgd, y_test)
print(classification_report(y_pred_sgd, y_test))
print(confusion_matrix(y_pred_sgd, y_test))
print(cross_val_score(sgd_clf, X_train_clf, y_train, cv=2))
```

```
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/
metrics/_classification.py:1245: UndefinedMetricWarning: Recall and F-
score are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/metri
cs/_classification.py:1245: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/admzlm4/opt/anaconda3/lib/python3.8/site-packages/sklearn/metri
cs/_classification.py:1245: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

           0       0.29      0.33      0.30       446
           1       0.76      0.43      0.55      1897
           2       0.07      0.45      0.13       154
           3       0.10      0.11      0.10        45
           4       0.53      0.70      0.61       179
```

```
              5       0.00      0.00      0.00         0
              6       0.62      0.44      0.52       113
              7       0.08      0.46      0.14        26
              8       0.07      0.22      0.10        41
              9       0.27      0.07      0.11       367

    accuracy                           0.39      3268
   macro avg       0.28      0.32      0.26      3268
weighted avg       0.57      0.39      0.43      3268

[[145  94  77   1   9   0   4  54  47  15]
 [251 816 668  16  53   0   8  26  28  31]
 [ 14  48  70   5   7   0   2   5   1   2]
 [  6   5  11   5   8   0   5   2   2   1]
 [  9  14  13   6 126   1   2   2   2   4]
 [  0   0   0   0   0   0   0   0   0   0]
 [  6   8  14   7  16   1  50   1   2   8]
 [  3   1   0   0   0   0   1  12   7   2]
 [  4   4   5   0   0   0   0  13   9   6]
 [ 70  79  88  12  17   0   8  31  36  26]]
[0.37898261 0.40051847]
```

In a binary classification problem, precision refers to the proportion of positive identifications that were actually correct. It is defined mathematically as: $P=\frac{TP}{TP+FP}$ **where TP (true positive) refers to the number of identifications that were correctly identified by the algorithm as "positive", and FP (false positive) refers to the total number of "positive" identifications that were incorrectly classified. This definition can be extended to a multicalss problem like the one presented above. For instance, the precision of the class of smooth, completely round galaxies can be defined as the number of smooth, completely round galaxies that were correctly classified by the model divided by all of the galaxies that were classified as being smooth and completely round (including incorrect classifications). Recall, on the other hand, refers to the total number of correct identifications of actual positives divided by the total number of correct identifications. Mathematically:** $R=\frac{TP}{TP+FN}$

where FN = false negative In the case of the above classification problem the recall for the class of smooth, completely round galaxies is given by the total number of galaxies that were correctly identified as being completely round by the model divided by the total number of galaxies that were correctly identified by the model. The F1 score is a combination of the precision and recall scores**

A confusion matrix provides a class-by-class breakdown of the number of correct and incorrect predictions made by a model, and can be used to, for instance, determine the classes that are being incorrectly predicted by the model

(K-fold)Cross validation is the process by which the provided data is split into K **training and test sets on which the model can be iteratively trained and tested. This**

can help avoid the issue of overfitting (i.e. where the model performs well on the training data, but does not generalise as well to the unseen test data). Note that in order to minimise computational time, 2-fold cross validation has been implemented for all of the above models

Overall, it is evident that all of the above classifers were able to correctly identify galaxies belonging to Class 0 and Class 1. Based on the above measures, it is evident that the most appropriate classifier to classify the images of the galaxies is the Random Forest Classifier, as it is able to accurate categorise almost 70% of the images, and it also has the highest cross validation score out of the five classifiers, indicating that it is the most likely to perform better on unseen data, and is therefore the least prone to overfitting. However, it must be noted that none of the classifiers trained above were able to consistently and accurately classify disk, edge-on boxy bulge galaxies, as characterised by the value of zero for the f1 score, and also the corresponding entry in the confusion matrix.

Assignment 1: Handwritten Notes

Saturday, 16 April 2022 12:09 pm

Probabilistic Graphical Model for Task 1 of Question 1: