6.1:

a)

Binary case:

```
[22] num_exp = 20
     test_dimensions = [2, 4, 8]
     for dim in test_dimensions:
         n_full = 2 ** dim
         n_avg = 0
         for i in range(num_exp):
             dataset = generate_random_dataset(dim, n_full, 2)
             X = dataset.drop('target', axis=1)
             y = dataset['target']
             X_train = X.copy()
             y_train = y.copy()
             for idx in range(n_full):
                 X_train = X.drop(index=idx)
                 y_train = y.drop(index=idx)
                 knn = KNeighborsClassifier(n_neighbors=1)
                 knn.fit(X_train, y_train)
                 y_pred = knn.predict(X)
                 different = False
                 for i in range(len(y)):
                     if y[i] != y_pred[i]:
                         different = True
                         break
                 if different:
                     n_avg += 1
         n_avg /= num_exp
         print(f"d={dim}: n_full={n_full}, Avg. req. points for memorization n_avg={n_avg:.2f}, n_full/n_avg={n_full/n_avg}")

     d=2: n_full=4, Avg. req. points for memorization n_avg=2.40, n_full/n_avg=1.6666666666666667
     d=4: n_full=16, Avg. req. points for memorization n_avg=8.20, n_full/n_avg=1.9512195121951221
     d=8: n_full=256, Avg. req. points for memorization n_avg=129.15, n_full/n_avg=1.9821912504839334
```

b)

Multiclass case:

```
num_exp = 20
test_dimensions = [2, 4, 8]
for dim in test_dimensions:
    n_full = 2 ** dim
    n_avg = 0
    for i in range(num_exp):
        dataset = generate_random_dataset(dim, n_full, 5)
        X = dataset.drop('target', axis=1)
        y = dataset['target']
        X_train = X.copy()
        y_train = y.copy()
        for idx in range(n_full):
            X_train = X.drop(index=idx)
            y_train = y.drop(index=idx)
            knn = KNeighborsClassifier(n_neighbors=1)
            knn.fit(X_train, y_train)
            y_pred = knn.predict(X)
            different = False
            for i in range(len(y)):
                if y[i] != y_pred[i]:
                    different = True
                    break
            if different:
                n_avg += 1
    n_avg /= num_exp
    print(f"d={dim}: n_full={n_full}, Avg. req. points for memorization n_avg={n_avg:.2f}, n_full/n_avg={n_full/n_avg}")

d=2: n_full=4, Avg. req. points for memorization n_avg=3.45, n_full/n_avg=1.1594202898550725
d=4: n_full=16, Avg. req. points for memorization n_avg=12.45, n_full/n_avg=1.285140562248996
d=8: n_full=256, Avg. req. points for memorization n_avg=203.35, n_full/n_avg=1.2589132038357511
```

6.2:

a)

Using the heart dataset:

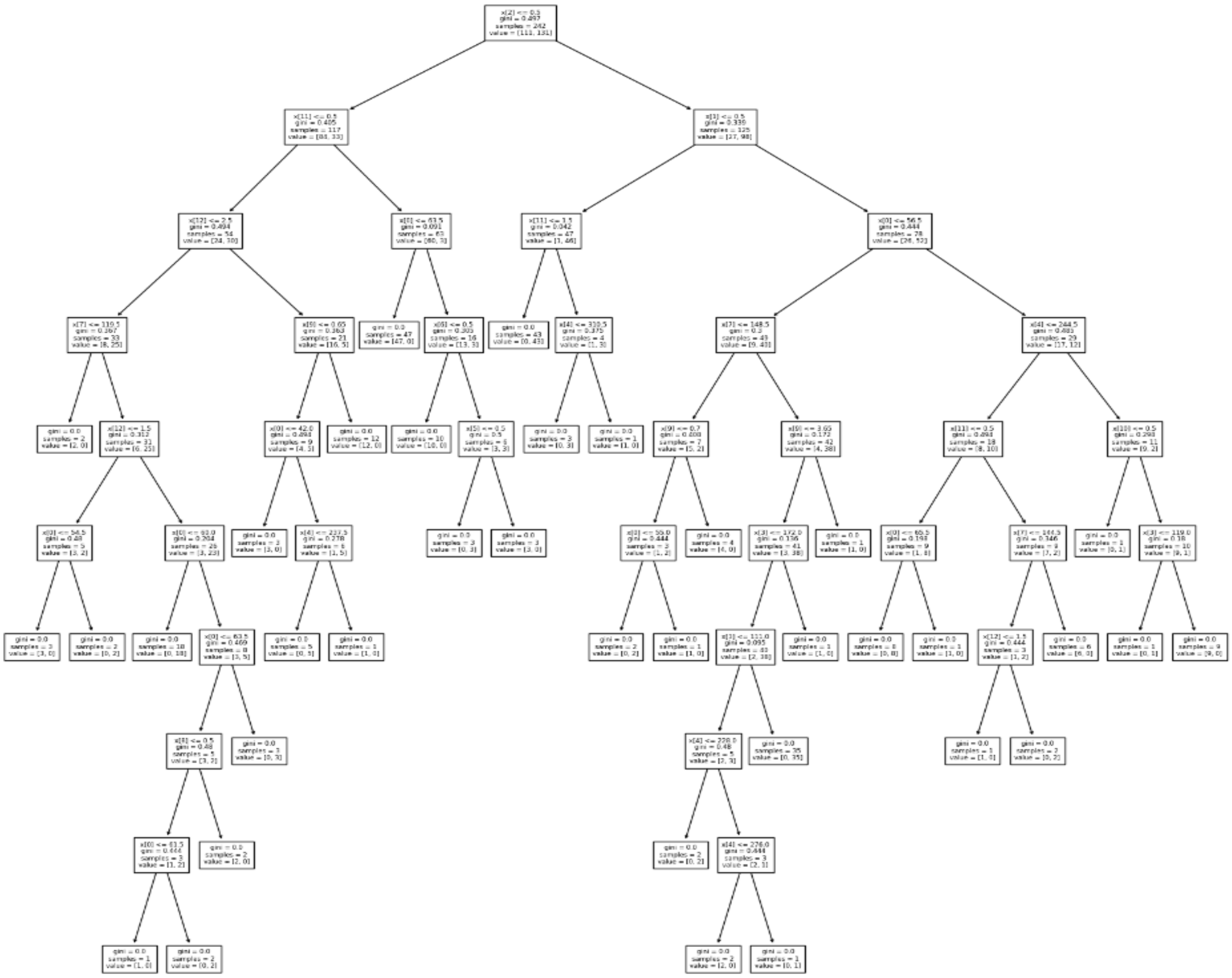
```
data_raw = pd.read_csv('/content/heart.csv')
X = data_raw.drop('output', axis=1)
y = data_raw['output']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

X_train.head()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall
74	43	0	2	122	213	0	1	165	0	0.2	1	0	2
153	66	0	2	146	278	0	0	152	0	0.0	1	1	2
64	58	1	2	140	211	1	0	165	0	0.0	2	0	2
296	63	0	0	124	197	0	1	136	1	0.0	1	0	2
287	57	1	1	154	232	0	0	164	0	0.0	2	1	2

Initial decision tree:

```
dt1 = tree.DecisionTreeClassifier()

dt1.fit(X_train, y_train)

y_pred = dt1.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

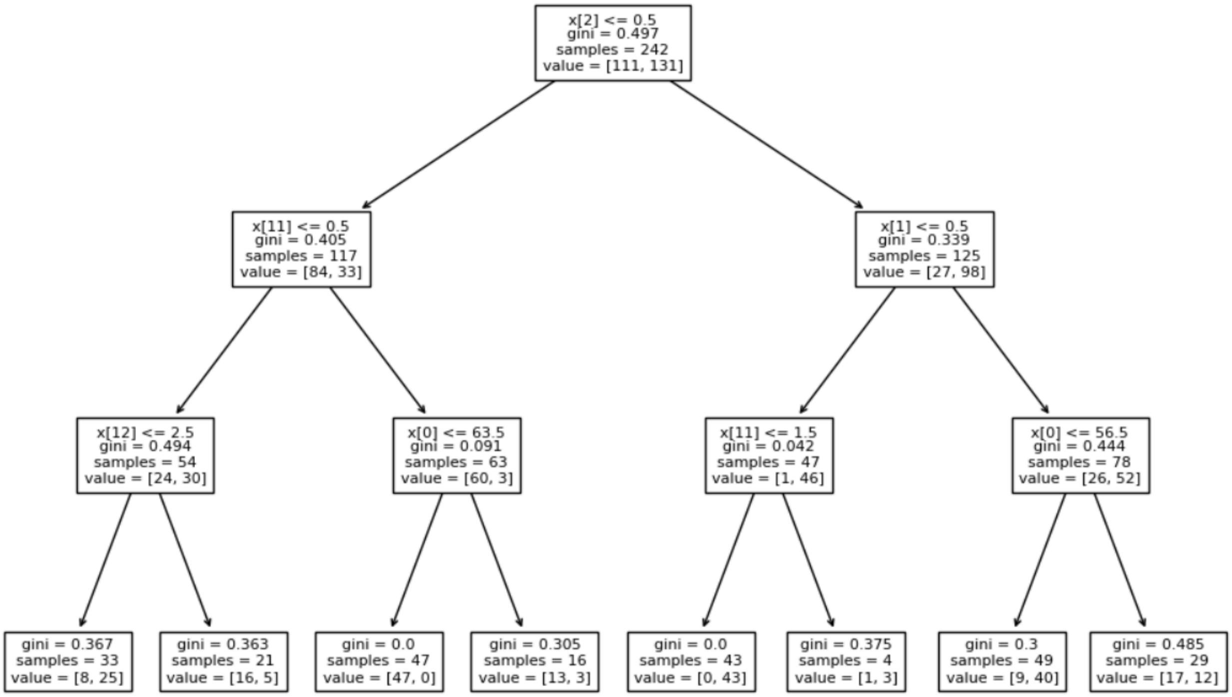
```
Accuracy: 0.7868852459016393
```



Limiting depth to 3:

```
dt2 = tree.DecisionTreeClassifier(max_depth=3)

dt2.fit(X_train, y_train)

y_pred = dt2.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

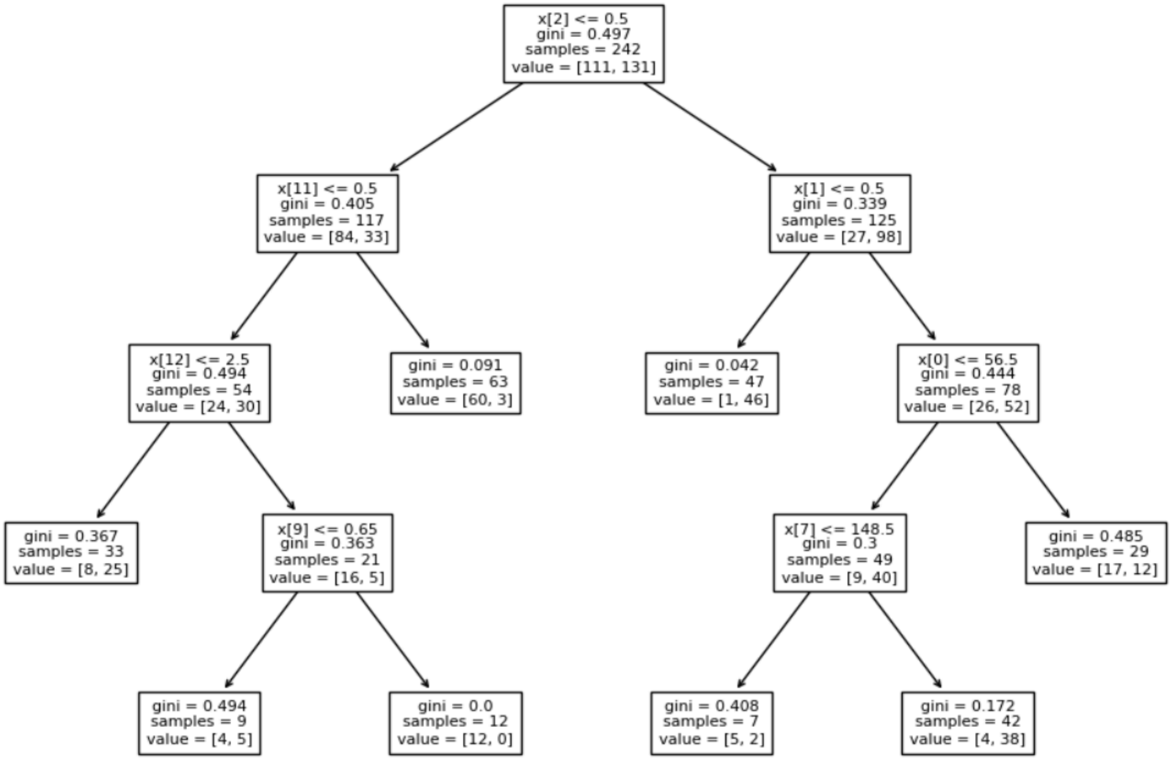
```
Accuracy: 0.819672131147541
```



Limiting leaf nodes to 8:

```
dt3 = tree.DecisionTreeClassifier(max_leaf_nodes=8)

dt3.fit(X_train, y_train)

y_pred = dt3.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7540983606557377
```



b)

Using the stroke dataset:

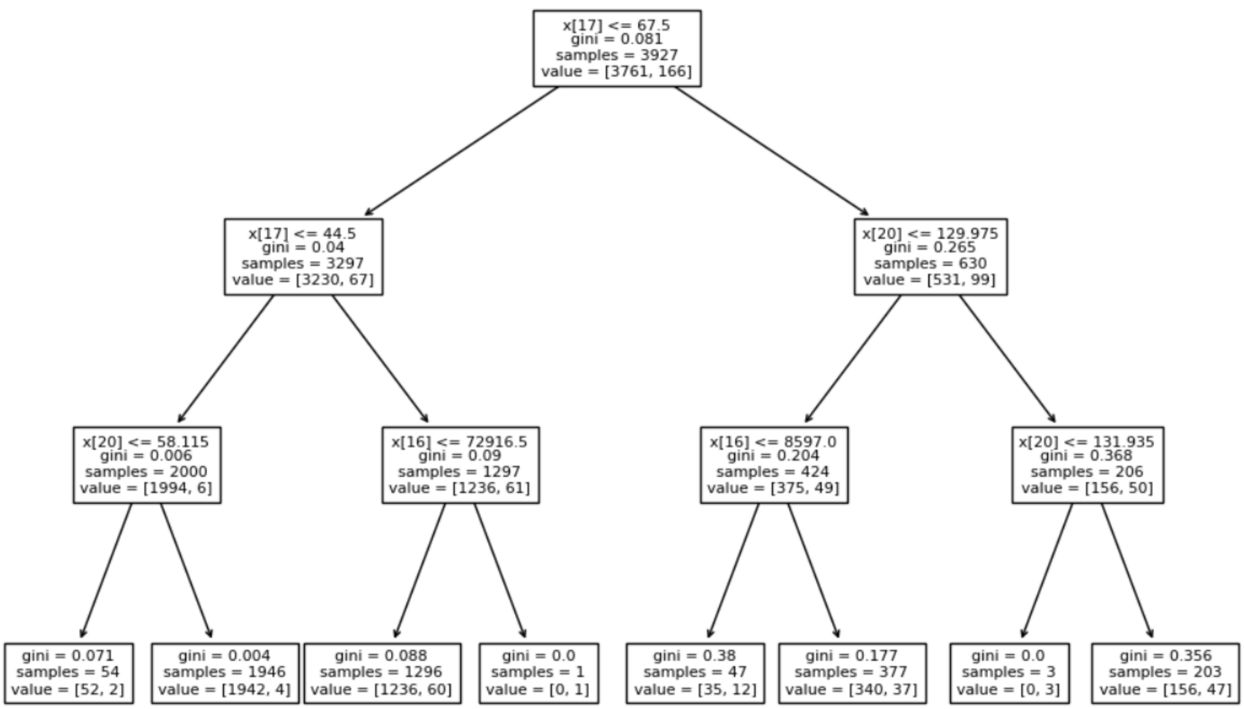
```
data_raw = pd.read_csv('/content/healthcare-dataset-stroke-data.csv')
data_raw = data_raw.dropna()
X = data_raw.drop('stroke', axis=1)
y = data_raw['stroke']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

X_train.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
5036	57159	Male	56.0	0	0	Yes	Self-employed	Rural	125.87	24.6	never smoked
3179	23893	Male	24.0	0	0	Yes	Private	Urban	103.45	25.1	smokes
4149	57080	Female	81.0	1	1	Yes	Self-employed	Urban	59.11	20.7	formerly smoked
1649	40393	Female	32.0	0	0	No	Private	Urban	68.19	21.1	never smoked
2981	69329	Female	62.0	0	0	Yes	Private	Rural	203.57	29.1	Unknown

Limiting depth to 3:

```python
categorical_cols = ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']

preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), categorical_cols)
    ],
    remainder='passthrough'
)

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', tree.DecisionTreeClassifier(max_depth=3))
])

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9541751527494908
```



c)
Using random dataset:

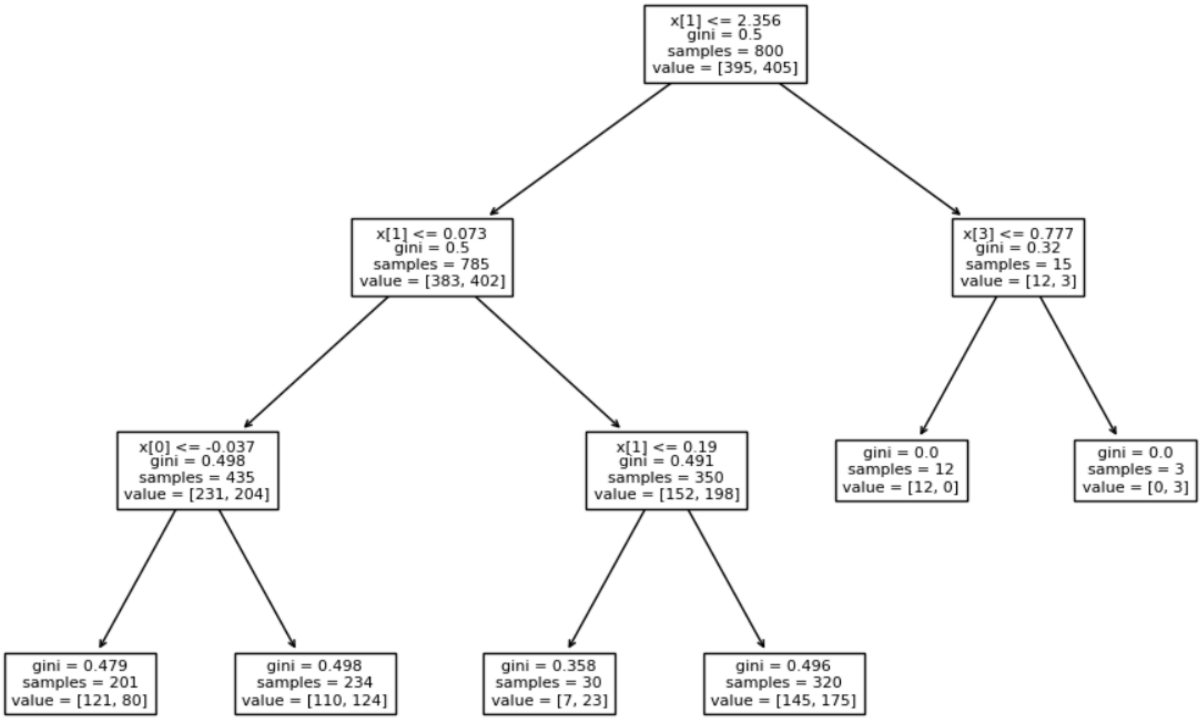
```
random_dataset = generate_random_dataset(5, 1000, 2)
X = random_dataset.drop('target', axis=1)
y = random_dataset['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

X_train.head()
```

	feature_1	feature_2	feature_3	feature_4	feature_5
687	0.893375	1.645318	-3.154200	1.410895	-1.109553
500	-1.438866	0.579128	-1.788463	-0.692195	-0.315527
332	0.626235	-0.353869	1.370535	0.073198	0.322181
979	-0.330763	-0.291903	0.119736	-0.115354	-0.255583
817	1.635701	0.703540	-0.277989	2.043371	-0.870054

Limiting depth to 3:

```
dt5 = tree.DecisionTreeClassifier(max_depth=3)

dt5.fit(X_train, y_train)

y_pred = dt5.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.48
```



6.3:

```
[ ] original_string = generate_random_string(200)
    compressed_data = zlib.compress(original_string.encode())

    original_size = sys.getsizeof(original_string)
    compressed_size = sys.getsizeof(compressed_data)

    print("Original String:", original_string)
    print("Original String Size:", original_size)
    print("Compressed Data:", compressed_data)
    print("Compressed Data Size:", compressed_size)
    print("Compression Ratio:", compressed_size / original_size)
```

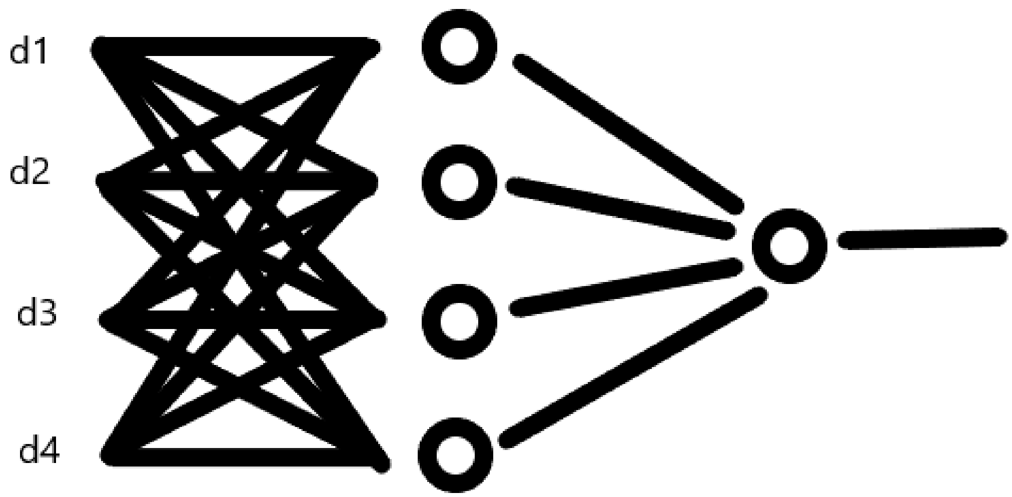
```
Original String: kp3ijzSQruGn4ucezwwsgvyW8vNdOYqgGZj1CHbBctSNzH7FrbyBjvbzb52YuPUBzkqY76ENvcaE0d14UDOfQ5UD2J8m8HJThyRT6Ah
Original String Size: 249
Compressed Data: b'x\x9c\x05\xc1\xc9\x16C0\x14\x00\xd0_2\x86m\r\x95\xe3\x10UT\xd3\x1d1\x87h\xd5P\xef\xeb{/\x7f\xab\xfd\x
Compressed Data Size: 215
Compression Ratio: 0.8634538152610441
```

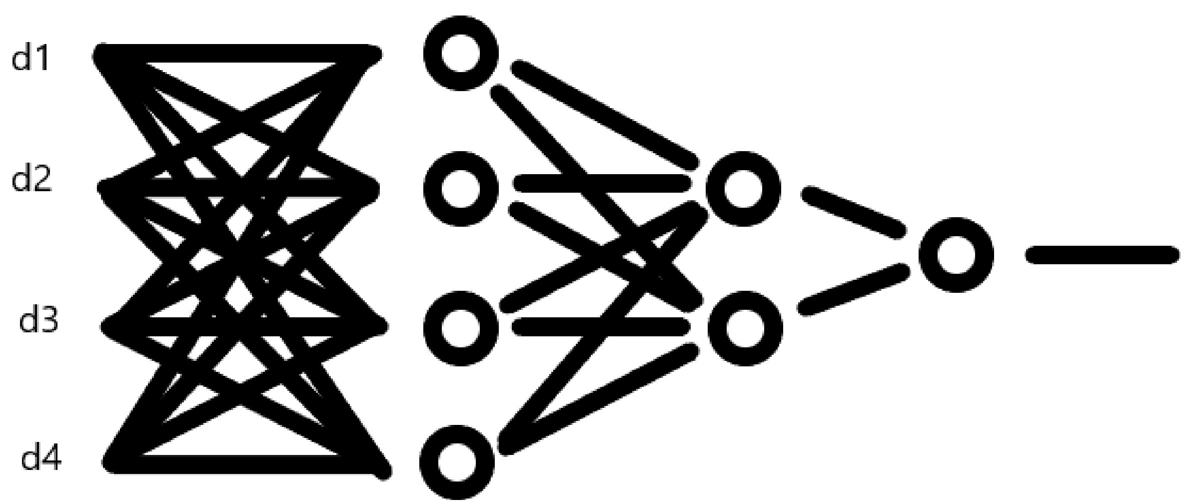
We see that the compression ratio is not very good. This is because the string is random, and there are no patterns/redundency for the compression algorithm to exploit. In a completely random string, the expected compression ratio should be close to 1:1.

8.1:

```
a)
12 + 3 + 3 = 18

b)
3 + 4 + 4 = 11

c)
a) can memorize 18 rows, b) can memorize 11 rows

d)
a) can memorize 9 rows, b) can memorize 5 rows
```

8.2:



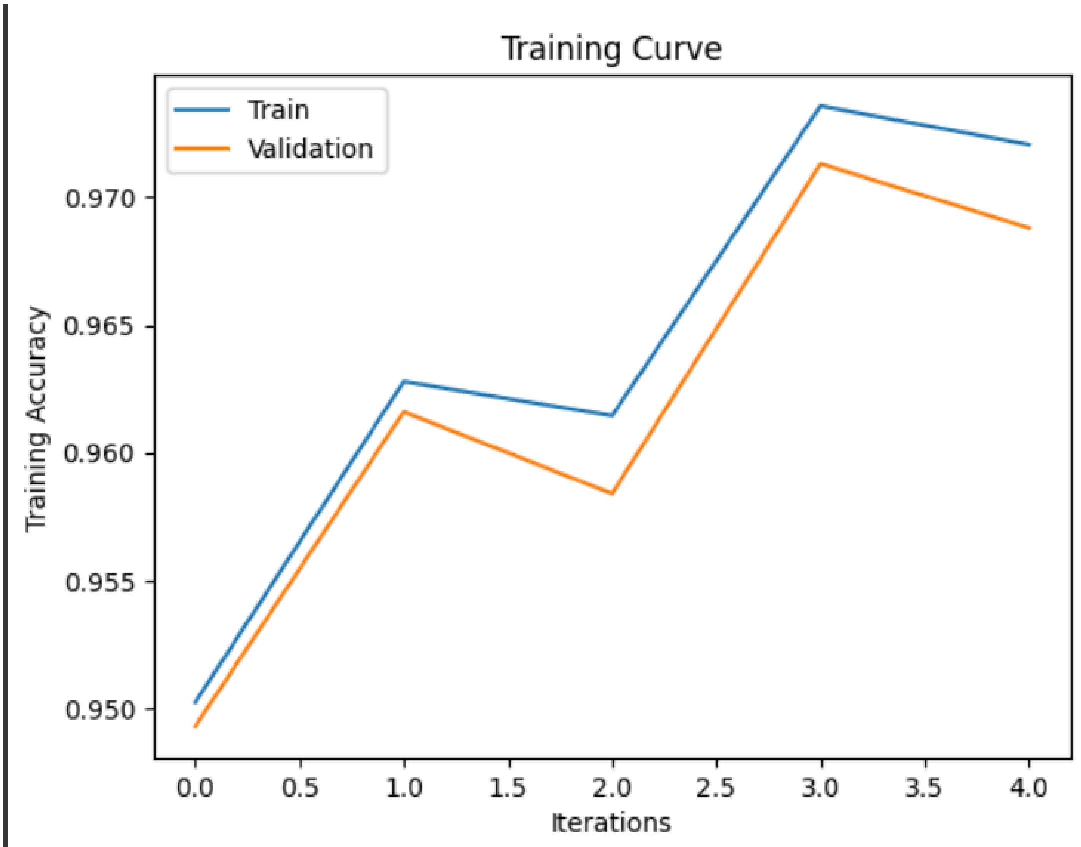


8.4:

```
a)
My brain's capacity: 10^11 x 1000 x 2 = 2E14

Total visual information in bits:
    Assume my brain can process 30 images per second, and each image is RGB (3 x 8 = 24 bits) and full HD (1920 x 1080 =
    2073600 pixels) and I am awake 16 hours per day

    Total visual information in bits = 23(years) x 365(days) x 16(hours) x 3600(seconds) x 30(images) x 2073600(pixels)
    x 24(bits) = 7.2E17

Total audio information in bits:
    Assume my brain can process 80 Kbps audio, this is 2.88E8 bits/hour

    Total audio information in bits = 23(years) x 365(days) x 16(hours) x 2.88E8(bits) = 1.1E13

Audio information size is negligible compared to visual information, and the combined size is a lot larger than my
brain's capacity. This makes sense because I don't remember all of this information.

Total bits in Shakespeare's works:
    There are 884647 words in Shakespeare's works, each word consists of 6.47 letters, and each letter consists of 1.6 bits.
    Total bits in Shakespeare = 884647 x 6.47 x 1.6 = 9.2E6

This is a lot less than the capacity of my brain.

==============================================================================================================================

b)
We can use the same algorithm to handle non-binary classification cases, since we are simply counting the number of
times the label changes

==============================================================================================================================

c)
We can simply define a small value 'e', in the second for loop of the algorithm we check if the next class is at least
'e' away from the last. If yes, then we increment threshold. Otherwise we treat it as the same class and do not
increment threshold
```

9.1:
Initial implementation using MNIST dataset:

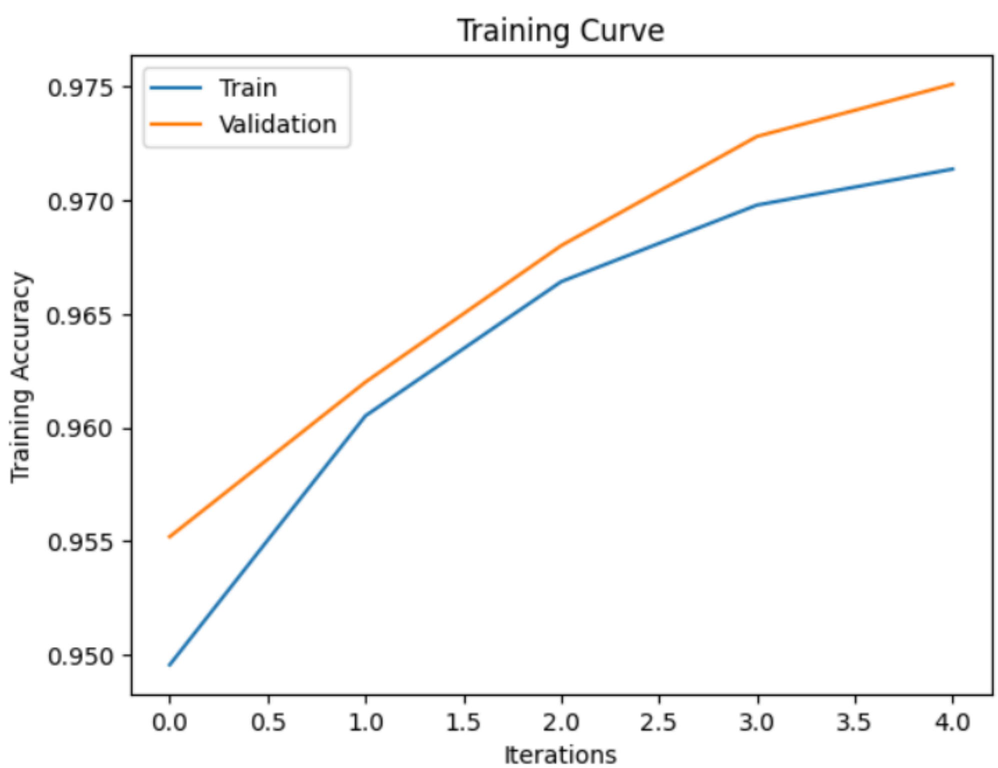
```
class SmallNet(nn.Module):
    def __init__(self):
        super(SmallNet, self).__init__()
        self.conv = nn.Conv2d(1, 3, 3, 1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(3 * 13 * 13, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv(x)))
        x = x.view(-1, 3 * 13 * 13)
        x = self.fc(x)
        x = x.squeeze(1) # Flatten to [batch_size]
        return x
```




```
Final Training Accuracy: 0.97205
Final Validation Accuracy: 0.9688
```

In the above case, the MEC of the fully connected layer is 3x13x13x10 = 5070, which is very large. We can make the MEC of the fully connected layer smaller by compressing more information in the CNN layers:

```
class SmallerNet(nn.Module):
    def __init__(self):
        super(SmallerNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 3, 5, 1)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(3, 3, 3, 1)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(3 * 5 * 5, 10)

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.view(-1, 3 * 5 * 5)
        x = self.fc(x)
        x = x.squeeze(1) # Flatten to [batch_size]
        return x
```




```
Final Training Accuracy: 0.9713666666666667
Final Validation Accuracy: 0.9751
```

This model has a smaller fully connected layer with MEC of 3x5x5x10=750, but it can classify just as well as the previous model.