Abirami Sabbani

CS294-82

Link to github: https://github.com/abiramisab/cs294_82

Chapter 6 – Generalization

6.1 a)

For this question, I generated random datasets of different sizes. I trained a model using K-Nearest Neighbors where k = 1. My conclusion was that n_full/n_avg is approximately 2.
n_full/n_avg ->2 as d grows which can be seen in the case of dimensionality 5, 6, and 7.

```
d=5: n_full=32, Avg. req. points for memorization n_avg==21.94, n_full/n_avg =1.4583333333
d=6: n_full=64, Avg. req. points for memorization n_avg==36.57, n_full/n_avg =1.7500000000
d=7: n_full=128, Avg. req. points for memorization n_avg==62.69, n_full/n_avg =2.0416666667
```

6.1 b)

For 4 classes (with dimensionalities 5, 6, and 8).

```
d=5: n_full=32, Avg. req. points for memorization n_avg==21.94, n_full/n_avg =1.4583333333
d=6: n_full=64, Avg. req. points for memorization n_avg==36.57, n_full/n_avg =1.7500000000
d=8: n_full=256, Avg. req. points for memorization n_avg==109.71, n_full/n_avg =2.3333333333
```

6.2 a)

I used the breast_cancer_wisconsin_data.csv dataset. The first step was to clean the dataset. Then, split into a training and testing set. Then train a decision tree classifier. Then, I wrote a tree_to_code recursive function which retrieves the if-then statements from the training table of the breast_cancer_wisconsin_data.csv dataset. After getting the if-then statements, I checked the accuracy. Then, I used different pruning techniques to minimize the number of if-then clauses. Note that some of the pruning techniques were most useful at reducing the number of if-then clauses than others. Even after reducing the number of if-then clauses, the accuracy was almost the same.

```
if concave points_mean <= 0.05:
  if radius_worst <= 16.83:
    if area_se <= 48.70:
      then predict 'Benign'
    else:  # if area_se > 48.70
      then predict 'Malignant'
  else:  # if radius_worst > 16.83
    if texture_mean <= 16.19:
      then predict 'Benign'
    else:  # if texture_mean > 16.19
      then predict 'Malignant'
else:  # if concave points_mean > 0.05
  if concave points_worst <= 0.15:
    if perimeter_worst <= 115.25:
      then predict 'Benign'
    else:  # if perimeter_worst > 115.25
      then predict 'Malignant'
  else:  # if concave points_worst > 0.15
    if fractal_dimension_se <= 0.01:
      then predict 'Malignant'
    else:  # if fractal_dimension_se > 0.01
      then predict 'Benign'
```

```python
from sklearn.metrics import accuracy_score

y_pred_check = dt_classifier_check.predict(X_test_check)

accuracy_check = accuracy_score(y_test_check, y_pred_check)

print(f"Accuracy: {accuracy_check:.11f}")

```

```
Accuracy: 0.94736842105
```

The above accuracy (without using any pruning techniques) is 0.94736842105. This is the original accuracy.

Now, reducing the number of if then clauses using different pruning techniques. Note that some pruning techniques worked better than others.

The first pruning method is maximum depth. This method significantly reduced the number of if-then clauses. Now the accuracy is 0.9298245614 which is close to the above, original accuracy of 0.94736842105.

```
if concave points_mean <= 0.05:
  if radius_worst <= 16.83:
    then predict 'Benign'
  else:  # if radius_worst > 16.83
    then predict 'Malignant'
else:  # if concave points_mean > 0.05
  if concave points_worst <= 0.15:
    then predict 'Benign'
  else:  # if concave points_worst > 0.15
    then predict 'Malignant'
```

```
y_pred_simplified = dt_classifier_simplified.predict(X_test_check)

accuracy_simplified = accuracy_score(y_test_check, y_pred_simplified)

print(f"Accuracy of the simplified model: {accuracy_simplified:.10f}")

```

```
Accuracy of the simplified model: 0.9298245614
```

The second pruning method is maximum leaf nodes. This method was as not as successful in reducing the number of if-then clasuses as the maximum depth pruning method above. The accuracy using maximum leaf nodes of 0.9385964912 is close to the original accuracy of 0.94736842105.

```
if concave points_mean <= 0.05:
  if radius_worst <= 16.83:
    if radius_se <= 0.63:
      then predict 'Benign'
    else:  # if radius_se > 0.63
      then predict 'Malignant'
  else:  # if radius_worst > 16.83
    if texture_worst <= 19.91:
      then predict 'Benign'
    else:  # if texture_worst > 19.91
      then predict 'Malignant'
else:  # if concave points_mean > 0.05
  if concave points_worst <= 0.15:
    if perimeter_worst <= 115.25:
      then predict 'Benign'
    else:  # if perimeter_worst > 115.25
      then predict 'Malignant'
  else:  # if concave points_worst > 0.15
    if concavity_se <= 0.14:
      then predict 'Malignant'
    else:  # if concavity_se > 0.14
      then predict 'Benign'
```

```
y_pred_max_leaf = dt_classifier_max_leaf.predict(X_test_check)

accuracy_max_leaf = accuracy_score(y_test_check, y_pred_max_leaf)

print(f"Accuracy of the max_leaf_nodes pruned model: {accuracy_max_leaf:.10f}")
```

```
Accuracy of the max_leaf_nodes pruned model: 0.9385964912
```

The third pruning method is minimum samples split nodes. This method was as not as successful in reducing the number of if-then clasuses as the maximum depth pruning method above. The accuracy using maximum leaf nodes of 0.9298245614 is close to the original accuracy of 0.94736842105.

```
if concave points_mean <= 0.05:
  if radius_worst <= 16.83:
    if area_se <= 48.70:
      then predict 'Benign'
    else:  # if area_se > 48.70
      then predict 'Malignant'
  else:  # if radius_worst > 16.83
    then predict 'Malignant'
else:  # if concave points_mean > 0.05
  if concave points_worst <= 0.15:
    then predict 'Benign'
  else:  # if concave points_worst > 0.15
    if concavity_se <= 0.14:
      then predict 'Malignant'
    else:  # if concavity_se > 0.14
      then predict 'Benign'
```

```
y_pred_min_samples_split = dt_classifier_min_samples_split.predict(X_test_check)

accuracy_min_samples_split = accuracy_score(y_test_check, y_pred_min_samples_split)

print(f"Accuracy of the min_samples_split pruned model: {accuracy_min_samples_split:.10f}")
```

```
Accuracy of the min_samples_split pruned model: 0.9298245614
```

6.2b)

I used the banana_quality_data.csv dataset and bank.csv.

This is for the banana_quality_data.csv dataset:

```
if Sweetness <= 0.61:
  if HarvestTime <= 0.58:
    if Ripeness <= 0.74:
      then predict 'Good'
    else:  # if Ripeness > 0.74
      then predict 'Good'
  else:  # if HarvestTime > 0.58
    if Ripeness <= -1.93:
      then predict 'Good'
    else:  # if Ripeness > -1.93
      then predict 'Bad'
else:  # if Sweetness > 0.61
  if Softness <= 1.19:
    if Weight <= -0.56:
      then predict 'Bad'
    else:  # if Weight > -0.56
      then predict 'Bad'
  else:  # if Softness > 1.19
    if Size <= -1.62:
      then predict 'Good'
    else:  # if Size > -1.62
      then predict 'Bad'
```

```
y_pred_check = dt_classifier_check_banana.predict(X_test_check)

accuracy_check = accuracy_score(y_test_check, y_pred_check)

print(f"Accuracy: {accuracy_check:.11f}")
```

```
Accuracy: 0.81062500000
```

The above accuracy (without using any pruning techniques) is 0.81062500000. This is the original accuracy.

The first pruning technique used was maximum depth. This was very successful in decreasing the number of if-then clauses. The accuracy was 0.7868750000 which is close to the original accuracy of 0.81062500000.

```
if Sweetness <= 0.61:
  if HarvestTime <= 0.58:
    then predict 'Good'
  else:  # if HarvestTime > 0.58
    then predict 'Bad'
else:  # if Sweetness > 0.61
  if Softness <= 1.19:
    then predict 'Bad'
  else:  # if Softness > 1.19
    then predict 'Bad'
```

```
y_pred_simplified = dt_classifier_banana_simplified.predict(X_test_check)

accuracy_simplified = accuracy_score(y_test_check, y_pred_simplified)

print(f"Accuracy of the simplified model: {accuracy_simplified:.10f}")
```

```
Accuracy of the simplified model: 0.7868750000
```

The second pruning techinque used was maximum leaf nodes. This was not successful in decreasing the number of if-then clauses. The accuracy was 0.81062500000 which is the same as the original accuracy of 0.81062500000.

```
if Sweetness <= 0.61:
  if HarvestTime <= 0.58:
    if Ripeness <= 0.74:
      then predict 'Good'
    else:  # if Ripeness > 0.74
      then predict 'Good'
  else:  # if HarvestTime > 0.58
    if Ripeness <= -1.93:
      then predict 'Good'
    else:  # if Ripeness > -1.93
      then predict 'Bad'
else:  # if Sweetness > 0.61
  if Softness <= 1.19:
    if Weight <= -0.56:
      then predict 'Bad'
    else:  # if Weight > -0.56
      then predict 'Bad'
  else:  # if Softness > 1.19
    if Size <= -1.62:
      then predict 'Good'
    else:  # if Size > -1.62
      then predict 'Bad'
```

```
y_pred_max_leaf = dt_classifier_banana_max_leaf.predict(X_test_check)

accuracy_max_leaf = accuracy_score(y_test_check, y_pred_max_leaf)

print(f"Accuracy of the max_leaf_nodes pruned model: {accuracy_max_leaf:.10f}")
```

```
Accuracy of the max_leaf_nodes pruned model: 0.8106250000
```

The third pruning technique used was minimum samples split. This was not successful in decreasing the number of if-then clauses. The accuracy was 0.8106250000 which is the same as the original accuracy of 0.8106250000.

```
if Sweetness <= 0.61:
  if HarvestTime <= 0.58:
    if Ripeness <= 0.74:
      then predict 'Good'
    else:  # if Ripeness > 0.74
      then predict 'Good'
  else:  # if HarvestTime > 0.58
    if Ripeness <= -1.93:
      then predict 'Good'
    else:  # if Ripeness > -1.93
      then predict 'Bad'
else:  # if Sweetness > 0.61
  if Softness <= 1.19:
    if Weight <= -0.56:
      then predict 'Bad'
    else:  # if Weight > -0.56
      then predict 'Bad'
  else:  # if Softness > 1.19
    if Size <= -1.62:
      then predict 'Good'
    else:  # if Size > -1.62
      then predict 'Bad'
```

```
y_pred_min_samples_split = dt_classifier_banana_min_samples_split.predict(X_test_check)

accuracy_min_samples_split = accuracy_score(y_test_check, y_pred_min_samples_split)

print(f"Accuracy of the min_samples_split pruned model: {accuracy_min_samples_split:.10f}")
```

```
Accuracy of the min_samples_split pruned model: 0.8106250000
```

This is for the bank.csv dataset.

```
if poutcome_unknown <= 0.50:
  if deposit_yes <= 0.50:
    if pdays <= 122.50:
      then predict 'Not Approve'
    else:  # if pdays > 122.50
      then predict 'Not Approve'
  else:  # if deposit_yes > 0.50
    if poutcome_other <= 0.50:
      then predict 'Approve'
    else:  # if poutcome_other > 0.50
      then predict 'Not Approve'
else:  # if poutcome_unknown > 0.50
  then predict 'Not Approve'
```

```
y_pred_check = dt_bank.predict(X_test_check)

accuracy_check = accuracy_score(y_test_check, y_pred_check)

print(f"Accuracy: {accuracy_check:.11f}")
```

```
Accuracy: 0.93864755934
```

The above accuracy (without using any pruning techniques) is 0.93864755934. This is the original accuracy.

The first pruning technique used was maximum depth. This was very successful in decreasing the number of if-then clauses. The accuracy was 0.9055082848 which is close to the original accuracy of 0.9386475593.

```
tree_to_code(dt_bank, X_check.columns)
```

```
if poutcome_unknown <= 0.50:
  if deposit_yes <= 0.50:
    then predict 'Not Approve'
  else:  # if deposit_yes > 0.50
    then predict 'Approve'
else:  # if poutcome_unknown > 0.50
  then predict 'Not Approve'
```

```
y_pred_simplified = dt_bank.predict(X_test_check)

accuracy_simplified = accuracy_score(y_test_check, y_pred_simplified)

print(f"Accuracy of the simplified model: {accuracy_simplified:.10f}")
```

```
Accuracy of the simplified model: 0.9055082848
```

The second pruning technique used was maximum leaf nodes. This was not successful in decreasing the number of if-then clauses. The accuracy was 0.9386475593 which is the original accuracy of 0.9386475593.

```
if poutcome_unknown <= 0.50:
  if deposit_yes <= 0.50:
    if pdays <= 122.50:
      then predict 'Not Approve'
    else:  # if pdays > 122.50
      then predict 'Not Approve'
  else:  # if deposit_yes > 0.50
    if poutcome_other <= 0.50:
      then predict 'Approve'
    else:  # if poutcome_other > 0.50
      then predict 'Not Approve'
else:  # if poutcome_unknown > 0.50
  then predict 'Not Approve'
```

```
y_pred_max_leaf = dt_classifier_bank_max_leaf.predict(X_test_check)

accuracy_max_leaf = accuracy_score(y_test_check, y_pred_max_leaf)

print(f"Accuracy of the max_leaf_nodes pruned model: {accuracy_max_leaf:.10f}")
```

```
Accuracy of the max_leaf_nodes pruned model: 0.9386475593
```

The third pruning technique used was minimum samples split nodes. This was not successful in decreasing the number of if-then clauses. The accuracy was 0.9386475593 which is the original accuracy of 0.9386475593.

```
if poutcome_unknown <= 0.50:
  if deposit_yes <= 0.50:
    if pdays <= 122.50:
      then predict 'Not Approve'
    else:  # if pdays > 122.50
      then predict 'Not Approve'
  else:  # if deposit_yes > 0.50
    if poutcome_other <= 0.50:
      then predict 'Approve'
    else:  # if poutcome_other > 0.50
      then predict 'Not Approve'
else:  # if poutcome_unknown > 0.50
  then predict 'Not Approve'
```

```
y_pred_min_samples_split = dt_classifier_bank_min_samples_split.predict(X_test_check)

accuracy_min_samples_split = accuracy_score(y_test_check, y_pred_min_samples_split)

print(f"Accuracy of the min_samples_split pruned model: {accuracy_min_samples_split:.10f}")
```

```
Accuracy of the min_samples_split pruned model: 0.9386475593
```

6.2c)

On a randomly generated dataset:

```
if Feature4 <= 68.50:
  if Feature4 <= 61.50:
    if Feature3 <= 64.00:
      then predict 'Less than 50'
    else:  # if Feature3 > 64.00
      then predict 'Less than 50'
  else:  # if Feature4 > 61.50
    then predict 'Less than 50'
else:  # if Feature4 > 68.50
  if Feature2 <= 42.00:
    if Feature3 <= 19.00:
      then predict 'Not Less than 50'
    else:  # if Feature3 > 19.00
      then predict 'Less than 50'
  else:  # if Feature2 > 42.00
    if Feature3 <= 10.00:
      then predict 'Less than 50'
    else:  # if Feature3 > 10.00
      then predict 'Not Less than 50'
```

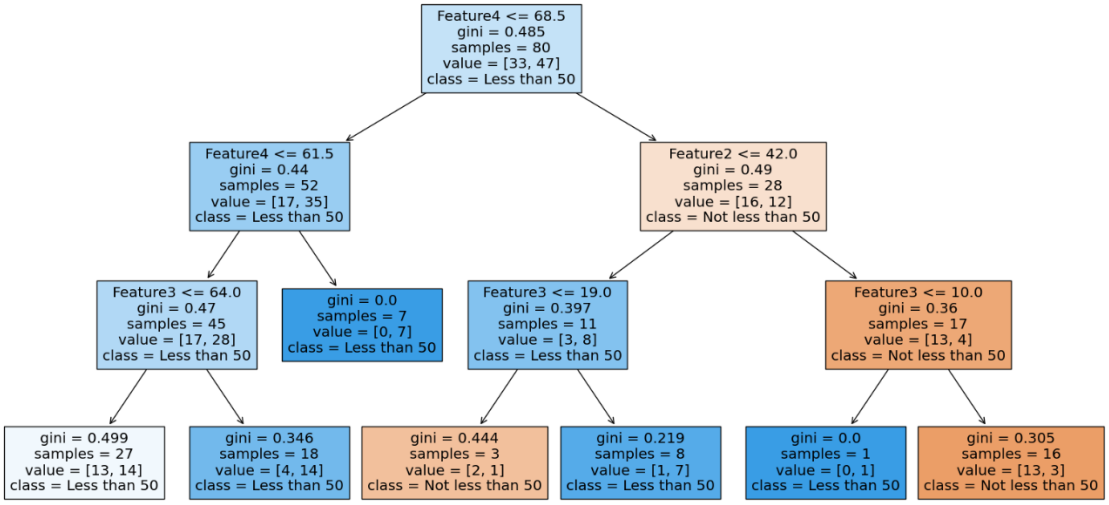
```
y_pred_check = dt_random.predict(X_test_check)

accuracy_check = accuracy_score(y_test_check, y_pred_check)

print(f"Accuracy: {accuracy_check:.2f}")

```

```
Accuracy: 0.60
```



The above accuracy (without using any pruning techniques) is 0.60. This is the original accuracy. Please see the figure above of the decision tree for the random dataset.

The first pruning method is maximum depth. This was very successful in decreasing the number of if-then clauses. The accuracy was 0.6500000000 which is somewhat close to the original accuracy of 0.60.

```
if Feature4 <= 68.50:
  if Feature4 <= 61.50:
    then predict 'Less than 50'
  else:  # if Feature4 > 61.50
    then predict 'Less than 50'
else:  # if Feature4 > 68.50
  if Feature2 <= 42.00:
    then predict 'Less than 50'
  else:  # if Feature2 > 42.00
    then predict 'Not Less than 50'
```

```
y_pred_simplified = dt_random.predict(X_test_check)

accuracy_simplified = accuracy_score(y_test_check, y_pred_simplified)

print(f"Accuracy of the simplified model: {accuracy_simplified:.10f}")
```

```
Accuracy of the simplified model: 0.6500000000
```

The second pruning method is maximum leaf nodes. This was not successful in decreasing the number of if-then clauses. The accuracy was 0.6500000000 which is somewhat close to the original accuracy of 0.60.

```
if Feature4 <= 68.50:
  if Feature4 <= 61.50:
    if Feature3 <= 64.00:
      then predict 'Less than 50'
    else:  # if Feature3 > 64.00
      then predict 'Less than 50'
  else:  # if Feature4 > 61.50
    then predict 'Less than 50'
else:  # if Feature4 > 68.50
  if Feature2 <= 42.00:
    if Feature1 <= 16.50:
      then predict 'Not Less than 50'
    else:  # if Feature1 > 16.50
      then predict 'Less than 50'
  else:  # if Feature2 > 42.00
    if Feature3 <= 10.00:
      then predict 'Less than 50'
    else:  # if Feature3 > 10.00
      then predict 'Not Less than 50'
```

```
y_pred_max_leaf = dt_random.predict(X_test_check)

accuracy_max_leaf = accuracy_score(y_test_check, y_pred_max_leaf)

print(f"Accuracy of the max_leaf_nodes pruned model: {accuracy_max_leaf:.10f}")
```

```
Accuracy of the max_leaf_nodes pruned model: 0.6500000000
```

The third pruning method is minimum samples leaf. This was very successful in decreasing the number of if-then clauses. The accuracy was 0.5500000000 which is somewhat close to the original accuracy of 0.60.

```
if Feature4 <= 68.50:
  if Feature4 <= 61.50:
    if Feature3 <= 64.00:
      then predict 'Less than 50'
    else:  # if Feature3 > 64.00
      then predict 'Less than 50'
  else:  # if Feature4 > 61.50
    then predict 'Less than 50'
else:  # if Feature4 > 68.50
  then predict 'Not Less than 50'
```

```
y_pred_min_samples_split = dt_random.predict(X_test_check)

accuracy_min_samples_split = accuracy_score(y_test_check, y_pred_min_samples_split)

print(f"Accuracy of the min_samples_split pruned model: {accuracy_min_samples_split:.10f}")
```

```
Accuracy of the min_samples_split pruned model: 0.5500000000
```

6.3 a)

For this exercise, I used Python to create a random string of length 1 million. Then I saved the string to a file. I used zip, 7z, and winzip to compress the file and get the compression ratio.

For 1000000 string length, before compression 977KB, after zip 811KB

For 1000000 string length, before compression 977KB, after 7z 817KB

For 1000000 string length, before compression 977KB, after winzip 815KB

For a string length of 1,000,000 characters:

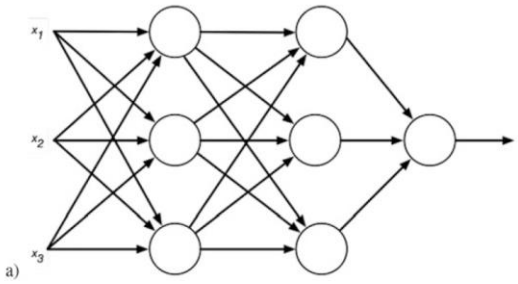
Using zip compression, the file size reduced from 977KB to 811KB, with a compression ratio of 1.2047. Using 7z compression, the file size reduced from 977KB to 817KB, resulting in a compression ratio of 1.1958. Using WinZip compression, the file size reduced from 977KB to 815KB, yielding a compression ratio of 1.1987.

6.3 b)

The expected compression ratio in 6.3a is 1. However, this can only occur if the string is truly random. The string that I have generated is psuedo-random. Therefore the compression ratio is close to 1 in for zip, 7z, and winzip. Note that even though the file compression algorithm used is different, the compression ratios are approximately the same, i.e., 1.2047 for zip, 1.1958 for 7z, and 1.1987 for winzip.

8.1 Maximum MEC of Neural Networks

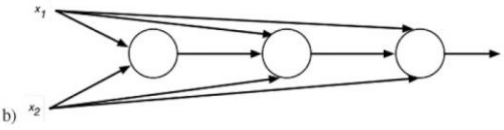
8.1a)



12 + min(3, 12) + min(3, 4) = 12 + 3 + 3 = 18 bits.

MEC is 18 bits.

8.1b)



3 + 4 + 4 = 11 bits.

MEC is 11 bits.

8.1c)

For the neural network in part 8.1a, 18 rows can be memorized for a binary classification table of 3 inputs.

For the neural network in part 8.1b, 11 rows can be memorized for a binary classification table of 2 inputs.

8.1 d)

For 4 classes instead of binary classification:

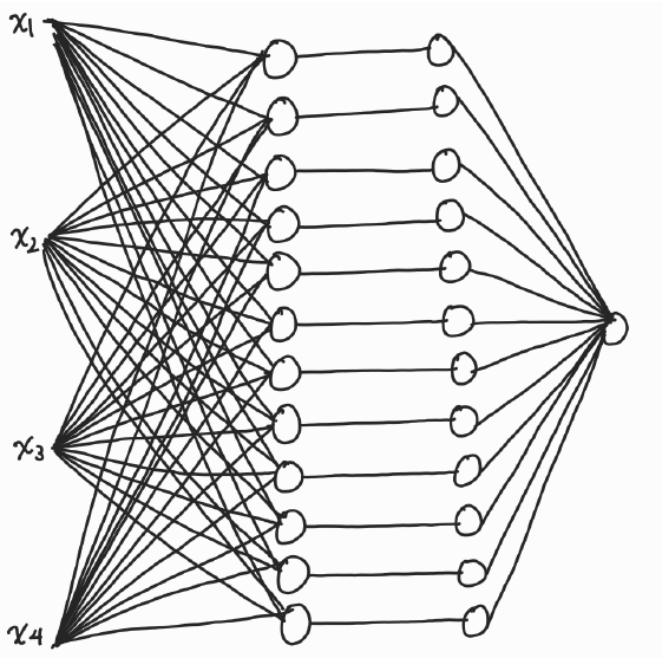
for 8.1a, 9 rows can be memorized for any 4 class classification training table of 3 inputs,

for 8.1b, 5 rows can be memorized for any 4 class classification training table of 2 inputs.

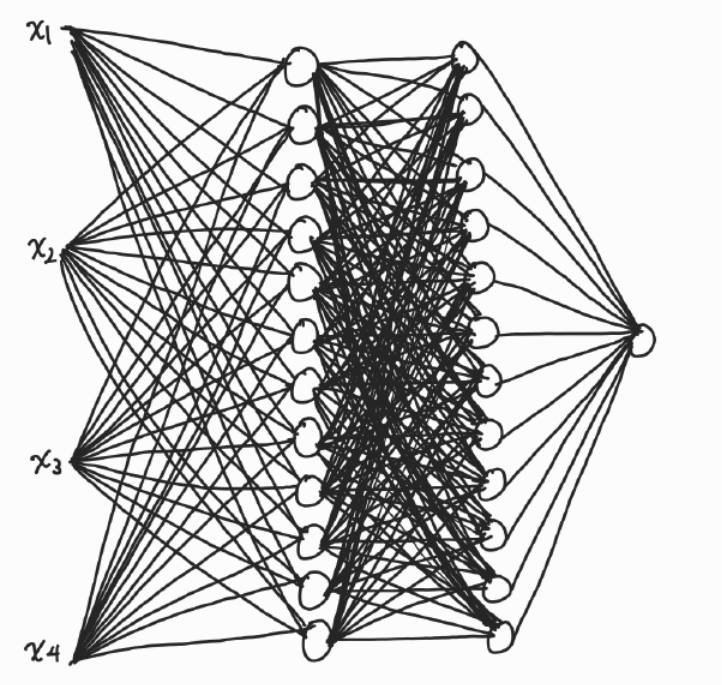
8.2)

Two different neural network architectures that can guarantee to memorize the training data of a 12-instance binary classification problem of 4-dimensional inputs (assuming perfect training):

1)



2)



8.4) Upper Bounds

8.4a)

Since you have 10^{11} neurons each making 1000 = 10^3 synaptic connections where the information capacity result for one neuron of two bits per connection applies, the Memory Equivalent Capacity (MEC) for your brain is $10^{11} * 10^3 = 10^{14}$.

First focusing on visual information. The retina of a single human eye has approximately $120 * 10^6$ rods and $6 * 10^6$ cones. For both eyes, there are approximately $240 * 10^6$ rods and $12 * 10^6$ cones. Rods allow us to see during nighttime or when there is no light while cones enable us to see during the day and see in color (i.e. color vision). If we want to see or observe something, we need to have the light on even at night. During nighttime, we either have the light on, or are asleep. So we are only concerned with the cones. Human beings have 3 separate color sensing cones, for red, green, and blue light. So we get, $12 * 10^6 * 3 = 36 * 10^6$. The human brain can process approximately 12 images per second. There are about 365.25 days in a year. I am 23 years old. To find out the number of images I have seen, the calculation is, 23 years * 365.25 days * 17 hours awake each day * 60 minutes per hour * 60 seconds per minute * 12 images per second= 6,169,510,800 images. Now, $6{,}169{,}510{,}800 * 36 * 10^6 = 222{,}102{,}388{,}800{,}000 \approx 2.22 * 10^{17}$ information. Since the brain can store 10^{14} bits of information and $2.22 * 10^{17} > 10^{14}$, just from the visual information, my brain is full. However, the human brain is not a hard disk as it responds to stimulus. We observe things and forget things on a day-to-day basis.

Second, focusing on auditory information. The human ear comprises of an outer ear, middle ear, and inner ear. The cochlea is involved in hearing. It is in the inner ear. We have a total of two cochleae. For simplicity, assume that the human brain only processes sounds that are words. It can process about 800 words per minute. There are about 365.25 days in a year and I am 23 years old. To find out the number of words I have heard, the calculation is, 23 years * 365.25 days * 17 hours awake each day * 60 minutes per hour * 800 words per minute = 6,855,012,000 words. Note that $6{,}855{,}012{,}000 < 10^{14}$. However, as mentioned above from visual information alone my brain is full. Again, the human brain is not a hard disk as it responds to stimulus. We observe things and forget things on a day-to-day basis.

Third, focusing on information gained through smell. The human nose has about 400 different scent receptors in your nasal cavity. Additionally the nose can detect 1 trillion separate odors. And the human brain can remember 50,000 scents. For simplicity, let's say that my brain remembers 50,000 scents throughout my life. There are about 365.25 days in a year and I am 23 years old. To find the number of scents that I have come across, the calculation is, 23 years * 365.25 days * 50,000 = 420,037,500. Note that $420{,}037{,}500 < 10^{14}$. However, as mentioned above from visual information alone my brain is full. Again, the human brain is not a hard disk as it responds to stimulus. We observe things and forget things on a day-to-day basis.

Fourth, focusing on information gained through taste. The tongue can process 5 different qualities of taste: sweet, sour, bitter, salty, and savory. An average human being has roughly 10,000 taste buds which get replaced every 2 weeks. An older person's taste buds do not get replaced. Since I am 23 years old, let's assume that I have 10,000 taste buds that get replaced every two weeks. There are about 365.25 days in a year. To find out how many tastes my brain has processed, the calculation is, 23 years * 365.25 days * 10,000 = 84,007,500. Note that $84{,}007{,}500 < 10^{14}$. However, as mentioned above from visual information alone my brain is full. Once again, the human brain is not a hard disk as it responds to stimulus. We observe things and forget things on a day-to-day basis.

Fifth, focusing on information gained through touch. The average human being has about $4 * 10^6$ touch receptors. Let's assume that my brain has processed the information from all the touch receptors. I am 23 years old and there are about 365.25 days in a year. To find out how many different "touches" my brain has processed, the calculation is 23 years * 365.25 days * $4 * 10^6$ touch receptors = 33,603,000,000. Note that $33{,}603{,}000{,}000 < 10^{14}$. However, as mentioned above from visual information alone my brain is full. Once again, the human brain is not a hard disk as it responds to stimulus. We observe things and forget things on a day-to-day basis.

Sixth, estimating the information content of the works of Shakespeare. There are approximately 884,647 words and 118,406 lines total in Shakespeare's works. The average English word contains 4.7 characters. Each character is 8 bits. To find the information content of the works of Shakespeare, the calculation is 884,647 words * 4.7 characters * 8 bits = 33,262,727.2 bits. Note that 33,262,727.2 < 10^{14} bits. So only reading all the works of Shakespeare would not necessarily fill my brain.

8.4b) Algorithm 8 to work with more than one binary classification

I implemented my Algorithm 8 with more than one binary classification on this example:

```
data = [
    [1, 2], [2, 1], [1, 0],
    [1, 3], [3, 1], [0, 1]
]
labels = [0, 1, 0, 2, 2, 1]

mec = memorize(data, labels)
print("Memory-equivalent capacity:", mec)

```

```
Memory-equivalent capacity: 10.287712379549449
```

8.4c) Algorithm 8 to work with regression

I implemented my Algorithm 8 with regression on this example:

```
data = [
    [1, 2], [2, 1], [1, 0],
    [1, 3], [3, 1], [0, 1]
]
targets = [1.2, 1.9, 0.5, 3.1, 3.0, 1.1]

mec = memorize_regression(data, targets, epsilon=0.5)
print("Memory-equivalent capacity for regression:", mec)

```

```
Memory-equivalent capacity for regression: 9.0
```

9.1)

For this question I first trained a model on MNIST. I used tensorflow. Then, I did hyperparameter tuning. To use the methods from class, I reduced the dataset by half randomly and again trained the model. Finally, I reduced the dataset by half randomly and did hyperparameter tuning and trained the model. I calculated the compression ratios for all of them.

Part 1 – Original Dataset

```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print('\nTest accuracy:', test_acc)

```

```
313/313 - 1s - loss: 0.0347 - accuracy: 0.9908 - 668ms/epoch - 2ms/step

Test accuracy: 0.9908000230789185
```

```
The original dataset compression ratio is: 119.03316326530613

The original dataset compression ratio after last layer: 27.591836734693878

The original dataset total compression ratio is: 0.36
[27.591836734693878, 1.4319526627218935, 0.36]

Compression Ratio for Conv2D layer of Original Dataset 1: 27.59
Compression Ratio for Conv2D layer of Original Dataset 2: 1.43
Compression Ratio for Conv2D layer of Original Dataset 3: 0.36
```

Part 2 – Hyperparameter Tuning

```
313/313 - 3s - loss: 0.0524 - accuracy: 0.9895 - 3s/epoch - 9ms/step

Modified Test accuracy: 0.9894999861717224

The hyperparameter tuning compression ratio is: 473.15561224489795

The hyperparameter tuning compression ratio after last layer: 55.183673469387756

The hyperparameter tuning total compression ratio is: 0.36
[55.183673469387756, 1.4319526627218935, 0.36]

The hyperparameter tuning compression ratio for Conv2D layer 1: 55.18
The hyperparameter tuning compression ratio for Conv2D layer 2: 1.43
The hyperparameter tuning compression ratio for Conv2D layer 3: 0.36
```

Part 3 – Reducing original dataset by half randomly

```
reduced_test_loss, reduced_test_acc = model.evaluate(reduced_test_images,  reduced_test_labels, verbose=2)
print('\nTest accuracy for reduced dataset:', reduced_test_acc)
```

```
156/156 - 0s - loss: 0.0789 - accuracy: 0.9754 - 318ms/epoch - 2ms/step

Test accuracy for reduced dataset: 0.9754279851913452
```

```
The reduced dataset compression ratio is: 119.03316326530613

The reduced dataset compression ratio after last layer is: 27.591836734693878

The reduced dataset total compression ratio is: 0.36
[27.591836734693878, 1.4319526627218935, 0.36]

The reduced dataset compression ratio for Conv2D layer 1: 27.59
The reduced dataset compression ratio for Conv2D layer 2: 1.43
The reduced dataset compression ratio for Conv2D layer 3: 0.36
```

Part 4 - Reducing original dataset by half randomly and hyperparameter tuning

```
1 reduced_test_loss, reduced_test_acc = model.evaluate(reduced_test_images,  reduced_test_labels, verbose=2)
2 print('\nTest accuracy for reduced dataset:', reduced_test_acc)
```

```
156/156 - 0s - loss: 0.0820 - accuracy: 0.9747 - 323ms/epoch - 2ms/step

Test accuracy for reduced dataset: 0.9747393727302551

The reduced dataset with hyperparameter tuning compression ratio is: 119.03316326530613

The reduced dataset with hyperparameter tuning compression ratio is: 27.591836734693878

The reduced dataset with hyperparameter tuning total compression ratio is: 0.36
[27.591836734693878, 1.4319526627218935, 0.36]

The reduced dataset with hyperparameter tuning compression ratio for Conv2D layer 1: 27.59
The reduced dataset with hyperparameter tuning compression ratio for Conv2D layer 2: 1.43
The reduced dataset with hyperparameter tuning compression ratio for Conv2D layer 3: 0.36
```

From this question, I saw empirically, that the dataset matters. The model behavior is not influenced by the hyperparameters, it is influenced by the dataset. Reducing the dataset by half randomly and training the model helped me arrive at this conclusion.