CSC442 Introduction to Artificial Intelligence

Project 4: Learning

Haoning Hu

Introduction

As instructed by the assignment, I implement and evaluate two machine learning algorithm, decision tree and neural network. I use three datasets to train and test these two algorithms. AIMA restaurant dataset, iris dataset and balloon dataset from UCI Machine Learning Archive. The report is divided into two parts, part 1 is about decision tree algorithm and part 2 is neural network, each part consists of analyze, design and demo.

1. Decision tree

1.1 Analyze

For decision tree algorithm, the first thing we have to think about is how to represent a decision tree. From its name "decision tree" we can easily get that the best choice is a tree, whose nodes are either attributes to test (at the internal nodes) or values to return (at the leaves), as well as the attributes and their domains of values themselves. Which means, the root of the tree is a test and it is the "best" to start the decision tree, and the leaves are the final classes of the test. In AIMA restaurant dataset they are "Yes" and "No". Another question is how we decide which attribute is the "best", or which is more "important" than others now. Here comes the entropy-based attribute selection method. Entropy is a measure of the uncertainty of a random variable; acquisition of information corresponds to a reduction in entropy. A random variable with only one value, a coin that always comes up heads has no uncertainty and thus its entropy is defined as zero; thus, we gain no information by observing its value. A flip of a fair coin is equally likely to come up heads or tails, 0 or 1, and we will soon show that this counts as "1 bit" of entropy. The roll of a fair *four*-sided die has 2 bits of entropy, because it takes two bits to describe one of four equally probable choices. Now consider an unfair coin that comes up heads 99% of the time. Intuitively, this coin has less uncertainty than the fair coin—if we guess heads, we'll be wrong only 1% of the time, so we would like it to have an entropy measure that is close to zero, but positive. In general, the entropy of a random variable V with value v_k, each with probability $P(v_k)$ is defined as

$$\text{Entropy:} \quad H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

Figure 1.1 Definition of entropy

An attribute A with d distinct values divides the training set E into subsets E1, . . ., Ed. Each subset Ek has pk positive examples and nk negative examples, so if we go along that branch, we will need an additional $B(p_k/(p_k + n_k))$ bits of information to answer the

question. A randomly chosen example from the training set has the kth value for the attribute with probability $(p_k + n_k)/(p + n)$, so the expected entropy remaining after testing attribute A is

$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B(\frac{p_k}{p_k+n_k})$$

Figure 1.2 Expected entropy remaining after testing attribute A

The information gain from the attribute test on A is the expected reduction in entropy:

$$Gain(A) = B(\frac{p}{p+n}) - Remainder(A)$$

Figure 1.3 Information gain

In fact, $Gain(A)$ is just what we need to implement the attribute selection function.

1.2 Design

The pseudo-code for decision tree algorithm is showed in figure 1.4. We can see that the algorithm contains two main functions, plurality-value and importance.

function DECISION-TREE-LEARNING($examples$, $attributes$, $parent\_examples$) **returns** a tree

if $examples$ is empty **then return** PLURALITY-VALUE($parent\_examples$)
else if all $examples$ have the same classification **then return** the classification
else if $attributes$ is empty **then return** PLURALITY-VALUE($examples$)
else
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, $examples$)
  $tree \leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e : e \in examples$ **and** $e.A = v_k\}$
    $subtree \leftarrow$ DECISION-TREE-LEARNING(exs, $attributes - A$, $examples$)
    add a branch to $tree$ with label $(A = v_k)$ and subtree $subtree$
  return $tree$

Figure 1.4 Pseudo-code of decision tree algorithm

The main function DECISION-TREE-LEARNING takes three parameters: examples, attributes and parent_examples. Examples is all the instances we have for this iteration, the parameter attributes is all the remaining attributes in this iterations and parent_examples is the examples from the last iteration. DECISION-TREE-LEARNING works iteratively, during each iteration it has three special condition, if the parameter examples is empty then it returns the function PLURALITY-VALUE with parameter parent_examples of this iteration. If all the instance from examples has the same classification, then return this classification. If attributes is empty then return PLURALITY-VALUE with the parameter examples. If none of the condition above, it will calculate the importance of all the attributes in this iteration and assign it to A, then create a new decision tree with root test A. Next, it will go into this loop assign the examples to exs with the instance e within which

the value of attribute A is vk, call the function DECISION-TREE-LEARNING with parameter exs, attributes without A and examples as the parent_examples. Assign the return value of this function to variable subtree. Add a branch with label A and subtree to the original tree. We know that the return value of DECISION-TREE-LEARNING will finally be a classification, so we will get all the value of the leaf nodes. The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly.

As it is mentioned above, what I use to represent a decision tree is actually a tree. It has two attributes, value and children. Where value is the value of this node and children is a dictionary. The keys of this dictionary are the labels of the test attributes and the values are the corresponding selection result or the subtree of this label. The final result of the program is a decision tree, which looks like this.

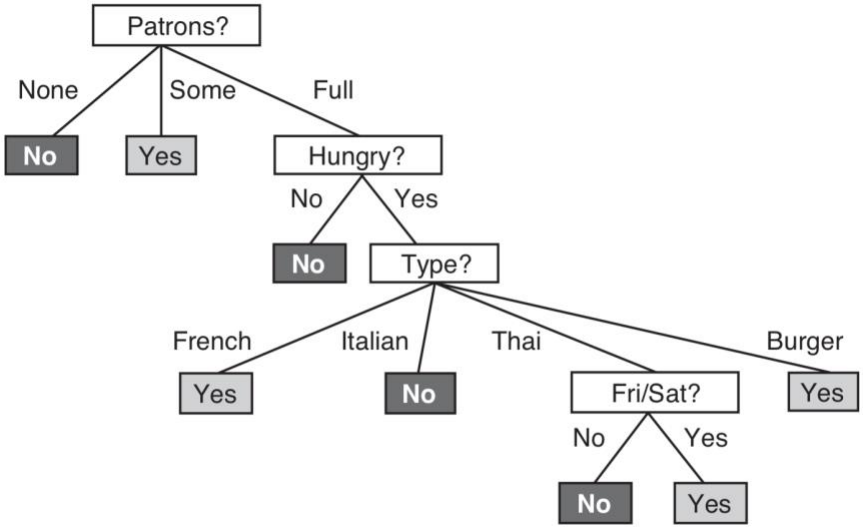

Figure 1.5 Decision tree for AIMA restaurant dataset

We know that a machine learning algorithm requires a training set and testing set, we can use the testing set to test the result and accuracy of the training models. To test the accuracy of the model I have a test function, which takes an instance of the examples and a tree as parameters, do a search of the tree to see if this instance satisfies the decision tree. It will return true if this example satisfies this tree, false if not. Then I write this loop to see how many examples satisfies this tree, divide it by the number of example of examples to get a accuracy.

1.3 Demo

To test the program I use three dataset, AIMA restaurant dataset, UCI iris dataset and UCI car evaluation dataset. UCI car evaluation dataset is considerably large, it contains 1728 examples with 6 attributes and 4 class values. I use an array list to store the data and separate them into training and testing set in the ratio of 8:2 to train and test the decision

tree. This ratio will be remained to perform classification on neural network so we can compare the performance between decision tree and neural network. The results are as follows.

```
AIMA restaurant dataset, Precision:90.00%
Iris dataset, Precision:90.00%
Car evaluation dataset, Correctness:94.22%
```

Figure 1.6 Result 1 of decision tree

As we can see, the precision of the three different dataset is pretty high, all of them is higher or equal to 90 percent, which proves that decision tree model has a high precision under the ratio of training set size and testing set size of 8:2. However, the result can change, because the sampling is totally random and even with the same training and testing sets it can also have different outcome. The restaurant dataset has the lowest precision mainly because the number of examples in this dataset is considerably small.

2. Neural Network

2.1 Analyze

Neural network, also known as Artificial neural network, is a computational model based on the structure and functions of biological neural networks. Information that flows through the network affects the structure of the neural network because a neural network changes or learns, in a sense based on that input and output. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

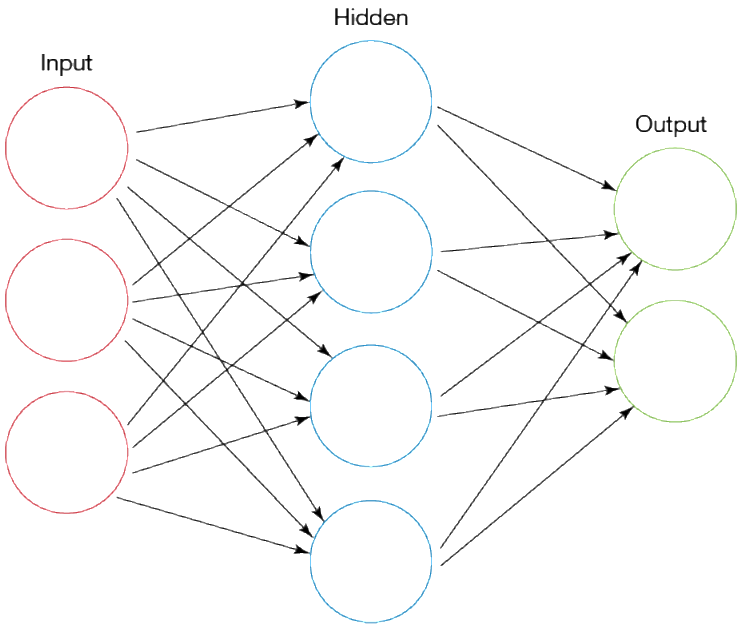

Figure 2.1 A simple neural network

2.2 Design

For this project I use the Python API tensorflow to implement the neural network classifier. TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

The name tensorflow is actually two words, "tensor" and "flow", this can explain the mechanism how tensorflow work. Tensorflow works by passing "tensor", tensor is a general term for unit of data. It is also represented with a rank, like in Matrix. They are geometric objects that describe linear relations between geometric vectors, scalars, and other tensors. For example, a 0-dimensional tensor is a scalar, a 1-dimensional tensor is a vector, a 2-dimensional tensor is a matrix and so on. Updated a tensor consists of a set of primitive values shaped into an array of any number of dimensions. In tensorflow we can create nodes with values, lets say a=tf.add(2,4), then we and create a session object to evaluate the graph to fetch the value of a. A session object encapsulates the environment in which operation objects are executed, and tensor objects are evaluated. Session will also allocate memory to store the current values of variables. Multiple graphs require multiple sessions, each will try to use all available resources by default.

How to create a neural network with Tensorflow, it is simple. We can use an Estimate

object to finish that. Estimator is TensorFlow's high-level representation of a complete model. It handles the details of initialization, logging, saving and restoring, and many other features. To write a TensorFlow program based on pre-made Estimators we can just create one or more input functions, define the model's feature columns. Instantiate an Estimator, specifying the feature columns and various hyperparameters. Then call one or more methods on the Estimator object, passing the appropriate input function as the source of the data. For the iris dataset, I use two hidden layers with ten nodes on each layer. To train the neural network, use the *train* function of the Estimator object with your training set and you can get a trained neural network! To test the precision of the model, we can use the function *evaluate* with the pre-defined testing set.

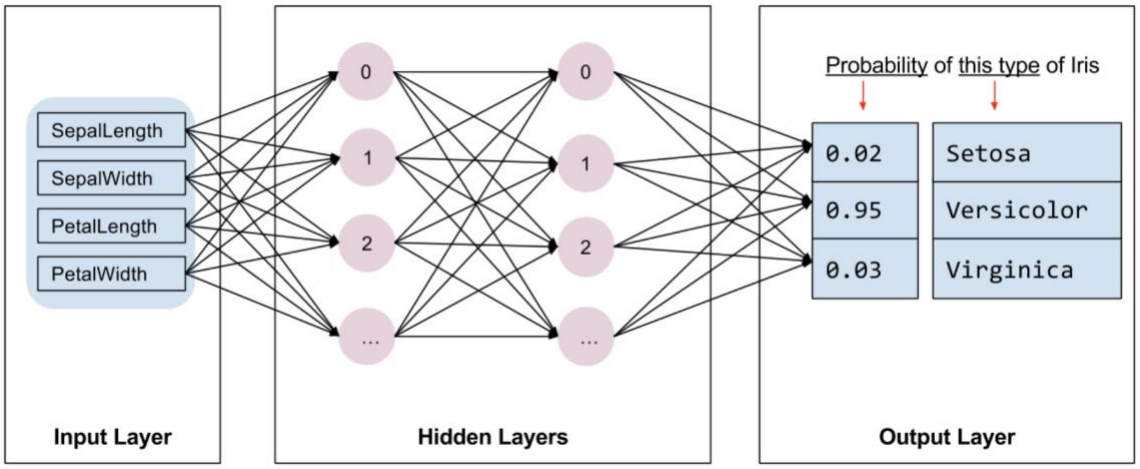

Figure 2.2 The neural network for iris dataset

2.3 Demo

To test the accuracy of the trained neural network I use the UCI iris dataset, cannot use the other two because the neural network I write cannot handle labels with categorical value, it will report error. I test the model with ratio of training set size and testing set size 8:2, the same with decision tree. The size of batch is 100, and it will train 1000 times. The result is as follow.

```
WARNING:tensorflow:Using temporary folder as model directory: /
2018-12-13 22:35:43.200925: I tensorflow/core/platform/cpu_feat
Precision: 0.9666666666666667
```

Figure 2.3 Result of neural network

We can see that the precision of the neural network model is 96.67 percent, remember the precision of decision tree for iris dataset is only 90 percent, we can assume that under this condition, the accuracy of neural network is better than decision tree.

3. Summary

To evaluate these two models, I make some plots to see how certain variables affect the performance of them.

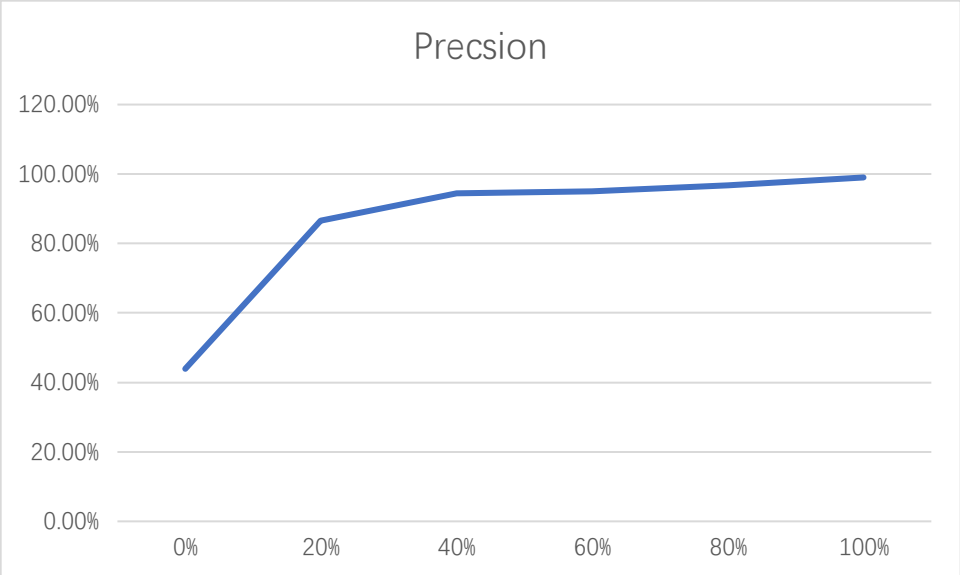

Figure 3.1 Precision varies with training set percentage in decision tree

This graph shows the relationship between precision and the percentage of train set size with the number of examples in this dataset, this is tested using the iris dataset. We can see that when percentage is 40 percent the precision becomes over 90 percent, this is because the iris dataset is pretty small, we can say that it is easier for the decision tree to predict.

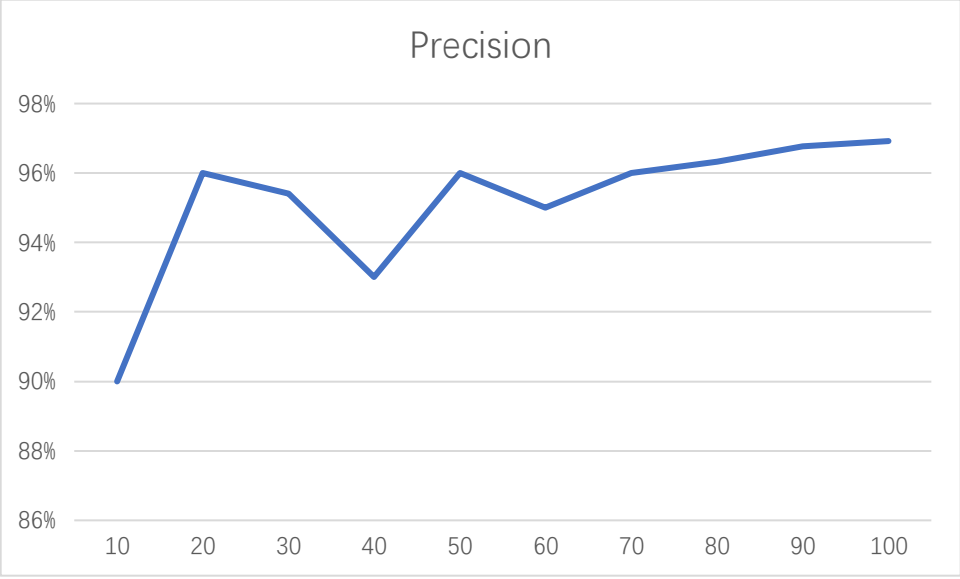

Figure 3.2 Precision varies with batch size in neural network

From the graph between precision and batch size we can get that batch size actually has really small impact on the precision of neural network. No matter how batch size changes,

the precisions are all above 90 percent.

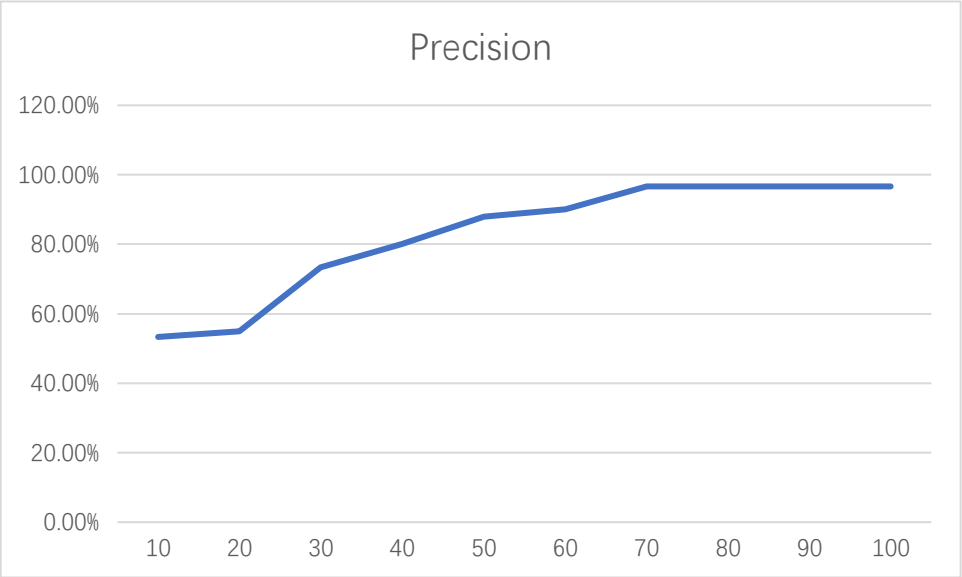

Figure 3.3 Precision varies with time in neural network

In Figure 3.3 we can see that with the number of times the model is trained the precision also increases, when time is bigger than 70, precision stays above 90 percent stably.