Project 4: Learning

Collaboration: Tianqi Yang& Kening Hu& Yunfeng Yu

Goal:

1. Implement a decision tree learning algorithm
2. Implement a multilayer perception
3. Evaluate the two machine learning algorithms.

Algorithm:

Decision Tree

Decision tree is one of most successful machine learning algorithms despite the fact that it's straightforward and simple. Given a training set, the construction of a decision tree firstly picks the most informative feature as a node of the tree. The norms to judge whether the feature is the most useful among all features are always based on how much entropy can be reduced. For example, if we can split the whole dataset into several groups according to the feature's values, and all examples in each group belong to the same class then we can say the feature is the most informative. After selecting the most informative feature, examples in dataset are grouped according to the value of the feature of each example. The construction then is run on each sub-dataset recursively until meeting some terminal conditions.

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 ← argmaxa ∈ attributes Information_Gain(a, examples)
            tree ← a new decision tree with root test A
            for each value vk of A do
                exs ←{e : e∈ examples and e.A = vk}
                subtree ← DECISION-TREE-LEARNING(exs, attributes − A, examples)
                add a branch to tree with label (A = vk) and subtree subtree
        return tree


    function PLURALITY-VALUE(examples) returns a classification
        class_distribution = number of each class / total number of examples;
        class = class_distribution[random a float]
        return class


    function Information_Gain(a, examples) returns a float number
        result = 0.0
        for each value_k in a.values:
            result += -P(value_k)*log(P(value_k))
        return result;
```

Neural Network

The neural network consists of input layer, hidden layer and output layer. The number of layers and the number of neurons are set before training the model. The weights between layers are usually randomly generated, and then via loss function weights are changed to minimize the loss via backward propagation . The backward propagation are described as follows:

function BACK-PROP-LEARNING(*examples*,*network*) *returns a neural network*

inputs: examples, a set of examples, each with input vector x *and output vector* y

network , a multilayer network with L *layers, weights* $w_{i,j}$ *, activation function* g

local variables: Δ*, a vector of errors, indexed by network node*

repeat

for each weight $w_{i,j}$ *in network do*

$w_{i,j} \leftarrow$ *a small random number*

for each example (x, y) *in examples do*

/∗ *Propagate the inputs forward to compute the outputs* ∗/ *for each node* i *in the input layer do*

$a_i \leftarrow x_i$

for $l = 2$ *to* L *do*

for each node j *in layer* l *do*

$in_j \leftarrow {}_i\, w_{i,j}\, a_i$

$a_j \leftarrow g(in_j)$

/∗ *Propagate deltas backward from output layer to input layer* ∗/

for each node j *in the output layer do*

$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$

for $l = L - 1$ *to* 1 *do*

for each node i *in layer* l *do*

$\Delta[i] \leftarrow g'(in_i)\ {}_j\, w_{i,j}\, \Delta[j]$

/∗ *Update every weight in network using deltas* ∗/

for each weight $w_{i,j}$ *in network do*

$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$

until some stopping criterion is satisfied

return network

Evaluation and Analysis

We have run the two models on two datasets, namely “iris.data.txt” and “scale1.data.txt”.

Decision tree

At first attempt we divided 80% data as training set and the left 20% as test set, without shuffling the examples. The decision tree is as following:


```
{ name: 3 value: ['S', 'ML', 'MS', 'L'] }

{ name: Iris-setosa value: [] }{ name: 2 value: ['S', 'ML', 'MS', 'L'] }{ name: Iris-versicolor value: [] }{ name: Iris-virginica value: [] }

{ name: Iris-versicolor value: [] }{ name: 0 value: ['S', 'MS', 'L', 'ML'] }{ name: Iris-versicolor value: [] }{ name: Iris-virginica value: [] }

{ name: Iris-virginica value: [] }{ name: 1 value: ['ML', 'MS', 'L', 'S'] }{ name: Iris-versicolor value: [] }{ name: Iris-versicolor value: [] }

{ name: Iris-versicolor value: [] }{ name: Iris-versicolor value: [] }{ name: Iris-versicolor value: [] }{ name: Iris-versicolor value: [] }
```


The first node of the tree is attribute_3, whose choice has ’S’, ‘ML’, ‘MS’, ‘L’. If the the attribute equals ’S’ then the example will go to the left node whose classification is Iris-setosa. But if the attribute of an example equals ‘ML’ then it will go to the second node to continue its classification. Until an example touches a leaf node, the classification will assign a label to it.

When run on the second dataset, the tree look like following:


```
{ name: 2 value: [1, 2, 3, 4, 5] }

{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 3 value: [1, 2, 3, 4, 5] }

{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 0 value: [1, 2, 3, 4]
 }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 0 value: [1, 2, 3, 4] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 0 value: [1, 2, 3, 4] }{ name: 1 value: [1, 2, 3, 4, 5]
}{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 0 value: [1, 2, 3, 4] }{ name: 1 value: [1, 2, 3
, 4, 5] }{ name: 0 value: [] }

{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 2 value: [] }{ name: 2 value: [] }{ name
: 3 value: [1, 2, 3, 4, 5] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 2 value: [] }{ name: 2 value: [] }{ n
ame: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ na
me: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 2 value: [] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 3 value: [1, 2, 3
, 4, 5] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 3 value: [1, 2, 3, 4, 5] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name:
 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4]
}{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name: 1 value: [
1, 2, 3, 4, 5] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value:
[1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [
1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1
, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [1, 2, 3, 4] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [1, 2, 3, 4, 5] }{ name:
 1 value: [1, 2, 3, 4, 5] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [1, 2, 3, 4] }

{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 0
 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 1 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 1 value: [
] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 1 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name
: 2 value: [] }{ name: 2 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value
: [] }{ name: 1 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ n
ame: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 1 va
lue: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 1 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }
{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 1 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2
 value: [] }{ name: 2 value: [] }{ name: 1 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 1 value: [] }{ name: 2 value: [
] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name
: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value
: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ n
ame: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 va
lue: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }
{ name: 2 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2
 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [
] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name
: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value
: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ n
ame: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 va
lue: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 2 value: [] }{ name: 2 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 0 value: [] }{ name: 1 value: [] }
```

And the classification of the two test sets are as followings:

```
[['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-versicolor'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-versicolor'], ['Iris-versicolor'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica'
], ['Iris-versicolor'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-versicolor'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica
'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-virginica'], ['Iris-versicolor']]
```

```
[[2], [2], [2], [1], [0], [2], [1], [0], [0], [0], [2], [0], [0], [0], [0], [1], [0], [0], [0], [0], [0], [0], [0], [0], [0], [2], [2], [2], [2], [2], [2], [2], [2], [1], [0], [2], [2], [0], [0], [0], [2], [1], [0], [0], [0], [2], [0], [0], [0], [0], [2], [2],
[2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [1], [0], [2], [2], [1], [0], [0], [2], [2], [0], [0], [0], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [1], [0], [2], [2], [2], [0], [0], [2], [2], [2], [2], [
2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [2], [1], [2], [2], [2], [1], [0]]
```

We can see that the first node is attribute_2 with 5 distinct values and when choosing value 1 or 2 the classification will see attribute_1 and otherwise it will see attribute_3. The whole process is almost the same as that of the first dataset.

However, the method of splitting data is unreasonable for the reason that the proportion of the three classes in the training set is imbalanced and the distribution of three classes in the test set is extremely unbalanced. Therefore, we split the training set and test set according to the original class distribution to train a model with more accuracy. Besides, the result of one split is obviously biased and inaccuracy and thus we looped the split for 10 times and compute the average accuracy and precision. The results of the decision trees on the two datasets are as followings:

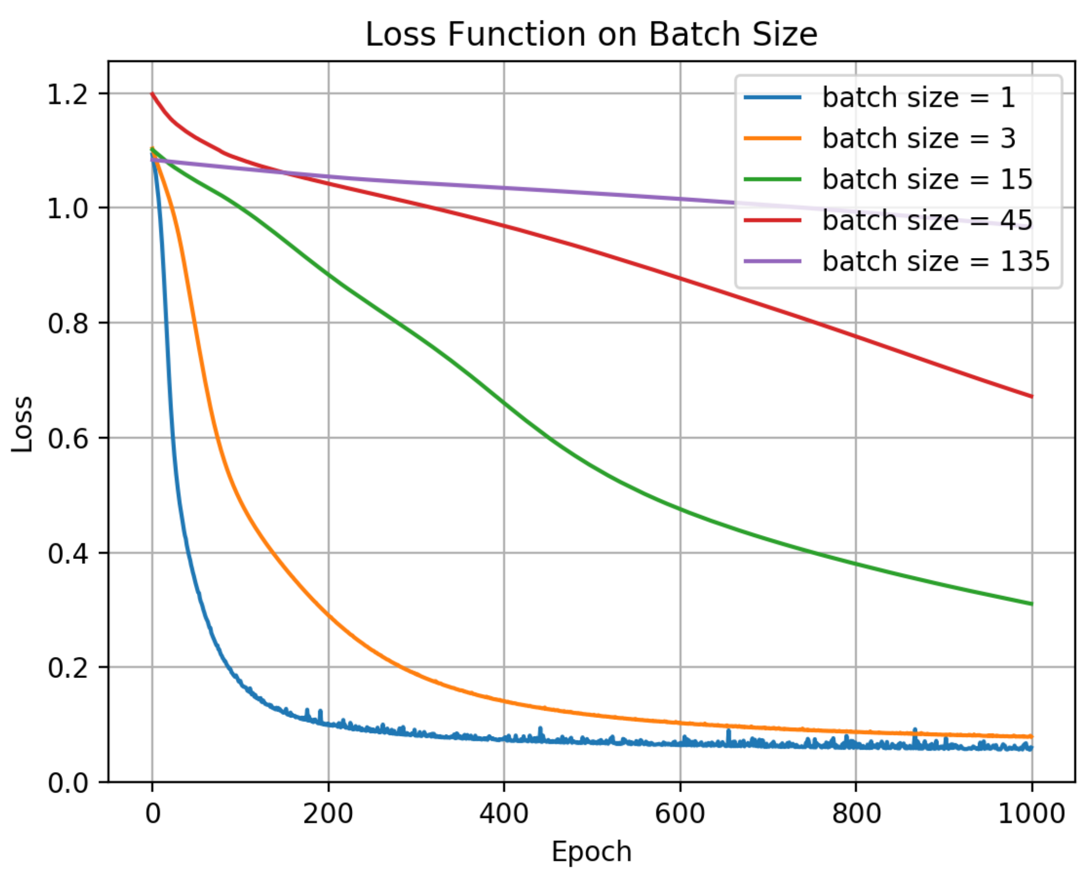
```
the accuracy_score: 0.8888888888888888 the avarage_precision_score_score: 0.833664021164021  the precision_score of each class : {0: 1.0, 1: 0.7486111111111111, 2: 0.7523809523809524}
the accuracy_score: 0.7555555555555555 the avarage_precision_score_score: 0.684287317620651  the precision_score of each class : {0: 0.9555555555555556, 1: 0.5565656565656566, 2: 0.5407407407407407}
the accuracy_score: 0.9111111111111111 the avarage_precision_score_score: 0.8633333333333333  the precision_score of each class : {0: 0.9555555555555556, 1: 0.8388888888888889, 2: 0.7955555555555556}
the accuracy_score: 0.7777777777777778 the avarage_precision_score_score: 0.7054353054353054  the precision_score of each class : {0: 0.9111111111111111, 1: 0.6242424242424243, 2: 0.5809523809523809}
the accuracy_score: 0.8 the avarage_precision_score_score: 0.7279487179487178  the precision_score of each class : {0: 0.8666666666666667, 1: 0.6077777777777778, 2: 0.7094017094017093}
the accuracy_score: 0.8666666666666667 the avarage_precision_score_score: 0.808869395711501  the precision_score of each class : {0: 0.9555555555555556, 1: 0.7099415204678362, 2: 0.7611111111111111}
the accuracy_score: 0.8444444444444444 the avarage_precision_score_score: 0.7825925925925925  the precision_score of each class : {0: 0.9111111111111111, 1: 0.7611111111111111, 2: 0.6755555555555556}
the accuracy_score: 0.7777777777777778 the avarage_precision_score_score: 0.7054353054353054  the precision_score of each class : {0: 0.9111111111111111, 1: 0.5809523809523809, 2: 0.6242424242424243}
the accuracy_score: 0.8888888888888888 the avarage_precision_score_score: 0.8362773029439697  the precision_score of each class : {0: 0.9555555555555556, 1: 0.8051282051282052, 2: 0.7481481481481482}
the accuracy_score: 0.9111111111111111 the avarage_precision_score_score: 0.8633333333333333  the precision_score of each class : {0: 0.9555555555555556, 1: 0.8388888888888889, 2: 0.7955555555555556}
after 10 times, the average precision_score is 0.781117662551873  the average accuracy_score is 0.8422222222222222
```

```
the accuracy_score: 0.6648936170212766 the avarage_precision_score_score: 0.4505186993283247  the precision_score of each class : {0: 0.658247492240565, 1: 0.0797872340425532, 2: 0.6135213717018559}
the accuracy_score: 0.6861702127659575 the avarage_precision_score_score: 0.46147685936325894  the precision_score of each class : {0: 0.6554107810086338, 1: 0.0797872340425532, 2: 0.6492325630385898}
the accuracy_score: 0.6808510638297872 the avarage_precision_score_score: 0.4513395221285887  the precision_score of each class : {0: 0.6443506969919295, 1: 0.0797872340425532, 2: 0.6298806353512834}
the accuracy_score: 0.6808510638297872 the avarage_precision_score_score: 0.4614272855456491  the precision_score of each class : {0: 0.6485137508615132, 1: 0.0797872340425532, 2: 0.6559808717328808}
the accuracy_score: 0.6276595744680851 the avarage_precision_score_score: 0.4419266659663128  the precision_score of each class : {0: 0.6135213717018559, 1: 0.0797872340425532, 2: 0.6324713921545293}
the accuracy_score: 0.7074468085106383 the avarage_precision_score_score: 0.47832505689935195  the precision_score of each class : {0: 0.6754971673829286, 1: 0.0797872340425532, 2: 0.6796907692725741}
the accuracy_score: 0.6648936170212766 the avarage_precision_score_score: 0.44812454492890824  the precision_score of each class : {0: 0.6191297555443128, 1: 0.0797872340425532, 2: 0.6454566451998588}
the accuracy_score: 0.6914893617021277 the avarage_precision_score_score: 0.4492772150660384  the precision_score of each class : {0: 0.6392064205828909, 1: 0.0797872340425532, 2: 0.6288379905726712}
the accuracy_score: 0.6595744680851063 the avarage_precision_score_score: 0.4458828963118086  the precision_score of each class : {0: 0.6499818148754319, 1: 0.0797872340425532, 2: 0.6078796400174407}
the accuracy_score: 0.6861702127659575 the avarage_precision_score_score: 0.4690475423127405  the precision_score of each class : {0: 0.6697876321611584, 1: 0.0797872340425532, 2: 0.6575677607345098}
after 10 times, the average precision_score is 0.45573462878509813  the average accuracy_score is 0.675
```

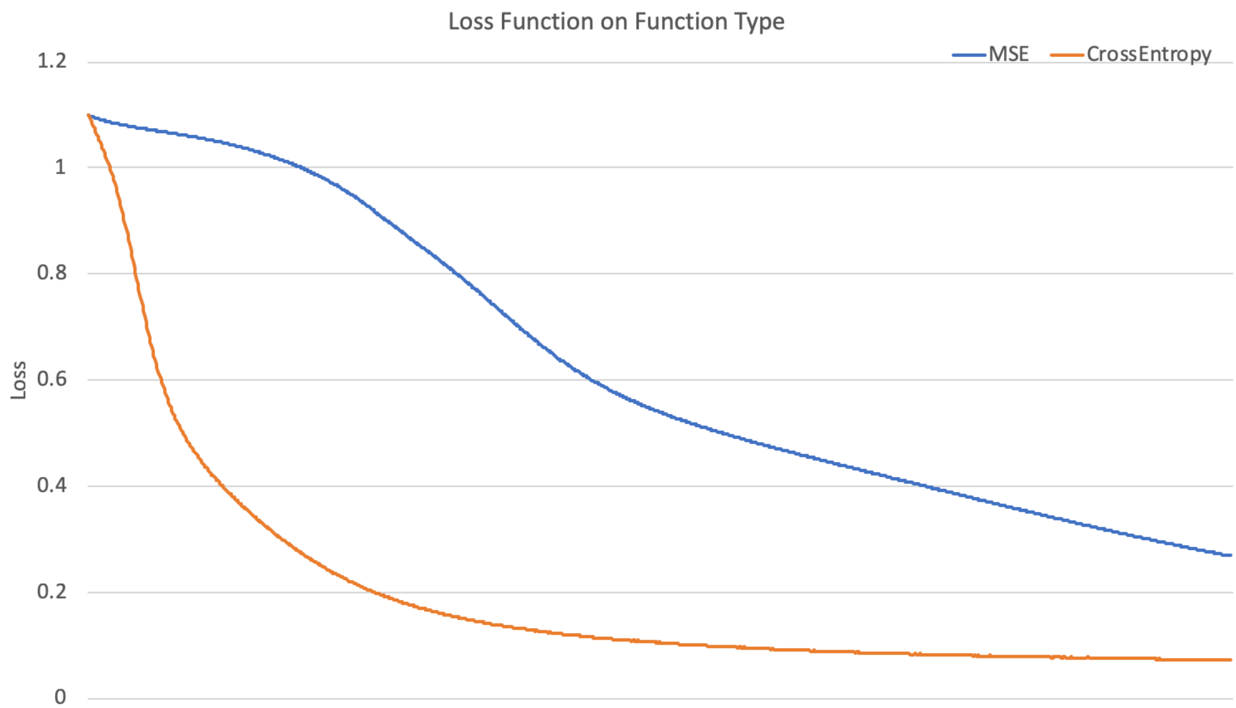
Obviously we can clearly see each label's precision score and the average accuracy. For the first dataset, the performance of the trained decision tree is relatively good because the average precision score is nearly 80% and the average accuracy score nearly 85%. However, when run on the second dataset the decision tree can only achieve 67% on accuracy and 45% on precision score. One of the most significant reason is that the distribution of classes in the second dataset is imbalanced. When looking at label 1, the precision score in each iteration is only about 8% and in fact the proportion of label 1 in the original dataset is far less than 30%.

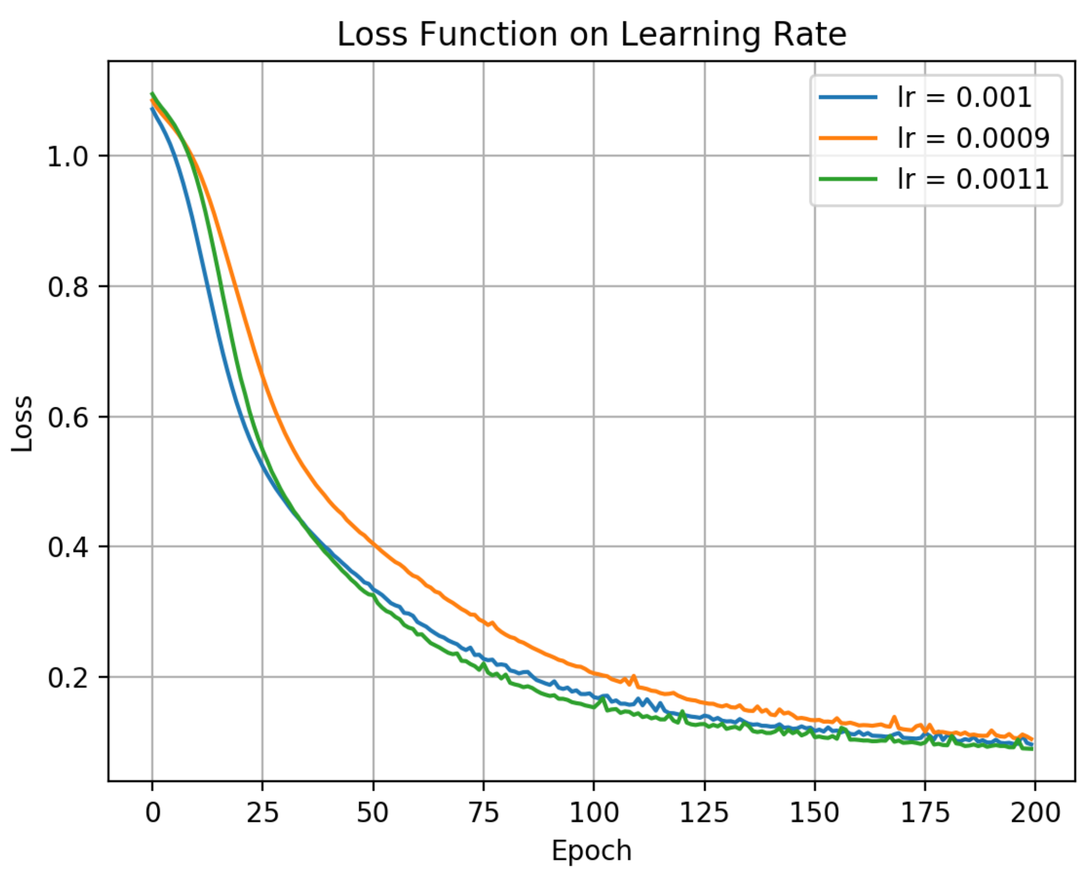
Neural Network

When other functions and parameters are fixed, we can see that the more less the batch is, the less loss the model can get. The line chart is described as below:



The relationships between loss and function type and between loss and learning rate are described as below:





models\accuracy	accuracy on iris	accuracy on scale1
Decision Tree	0.78	0.45
Neural Network	0.98	0.87

We can see that no matter on which dataset, the accuracy of neural network always performs better than decision tree does.