Write-up on project 4

Working Content

I finish this project independently. The working content includes:

1. Decision tree classification
2. Visualizing decision tree
3. Evaluating decision tree by K-Fold cross validation
4. 3-layer neural network (unfinished for gradient mistake of softmax)
5. Write-up on this project

Decision tree classification

The decision tree is one of the most commonly used classification techniques; recent survey claim that it's the most commonly used technique.

1.1 Tree construction

To build a decision tree, we need to make a first decision on the dataset to dictate which feature is used to split the data. To determine this, I try every feature and measure which split will give me the best results. After that, we will split the dataset into subsets. The subsets will then traverse down the branches of the first decision node. If the data on the branches is the same class, then we've properly classified it and don't need to continue splitting it. If the data isn't the same, then we need to repeat the splitting process on this subset. The decision on how to split this subset is done the same way as the original dataset, and we repeat this process until we have classified all the data.

Pseudo-code for a function called createBranch() would look like this:

Check if every item in the dataset is in the same class:

If so return the class label

Else

Find the best feature to split the data

Split the dataset

Create a branch node

For each split

Call createBranch and add the result to the branch node

Return branch node

1.2 Information gain

We choose to split our dataset in a way that makes our unorganized data more organized. There are multiple ways to do this, and each has its own advantages and disadvantages. One way to organize this messiness is to measure the information. Using information theory, we can measure the information before and after the split. The change in information before and after the split is known as information gain. The split with the highest information gain is our best option.

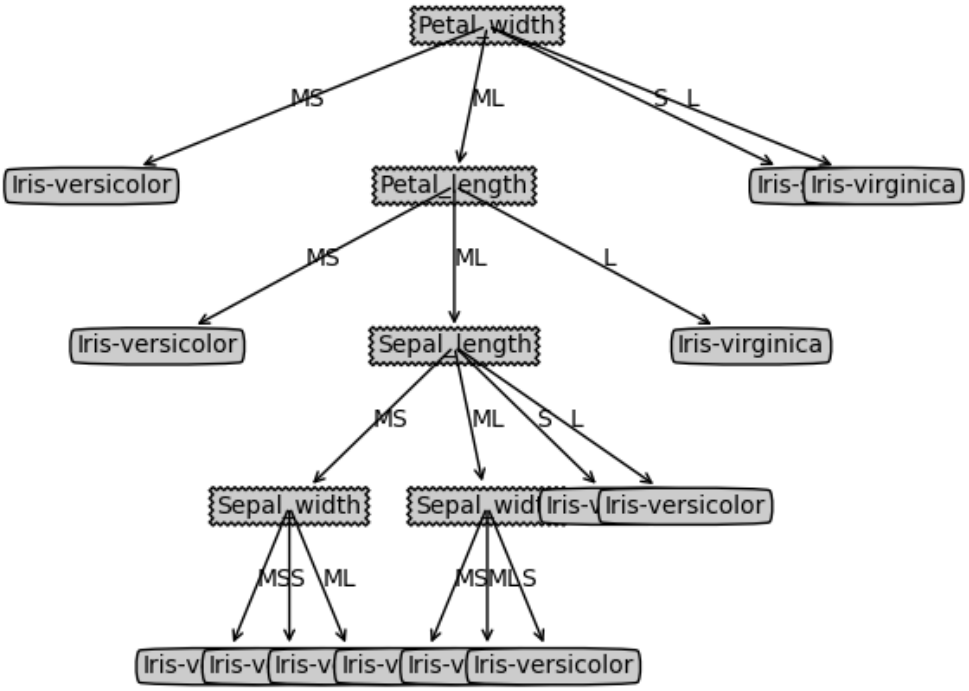
The measure of information of a set is known as Shannon entropy, or entropy for short. Entropy is defined as the expected value of the information. First, we need to define information. If we are classifying something that can take on multiple values, the information for symbol x^i is defined as $l(x_i) = log_2 p(x_i)$ where $p(x_i)$ is the probability of choosing this class. To calculate entropy, we need the expected value of all the information of all possible values of our class. This is given by

$$H = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

Where n is the number of classes.

1.3 Visualizing decision tree

One of the greatest strengths of decision trees is that humans can easily understand them. The plotting library I use is matplotlib. Matplotlib has a great tool, called annotations, that can add text near data in a plot. Annotations are usually used to explain some part of the data. But having the text on top of the data looks ugly, so the tool has a built-in arrow that allows us to draw the text a safe distance from the data yet show that data we are talking about. The following is an example of plotting decision tree using Iris dataset:



1.4 Evaluating decision tree

K-fold cross-validation is used to evaluate the decision tree. Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

The general procedure is as follows:

- Shuffle the dataset randomly.
- Split the dataset into k groups.
- For each unique group:

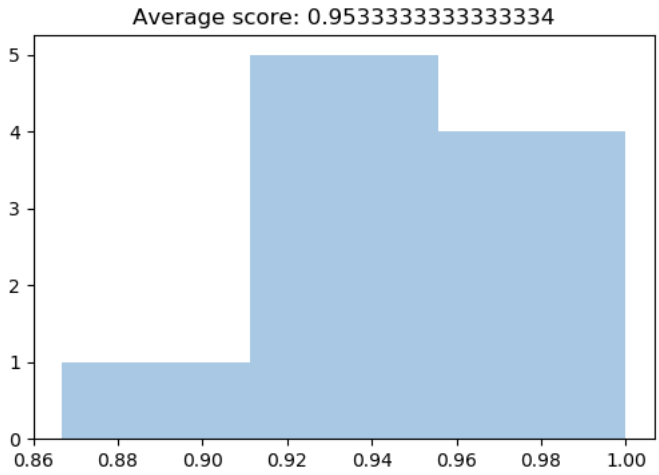
  Take the group as a hold out or test dataset

  Take the remaining groups as training data set

  Fit a model on the training set and evaluate it on the test set

  Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores.

By using 10-fold cross-validation, the average score of decision tree's performance is about 95.3%.



1.5 Summary

A decision tree classifier is just like a work-flow diagram with the terminating blocks representing classification decisions. Starting with a dataset, we can measure the inconsistency of a set or the entropy to find a way to split the set until all the data belongs to the same class. ID3 algorithm can split nominal-valued datasets. Recursion is used in tree-building algorithms to turn a dataset into a decision tree. The tree is represented in a python dictionary rather than a special data structure.

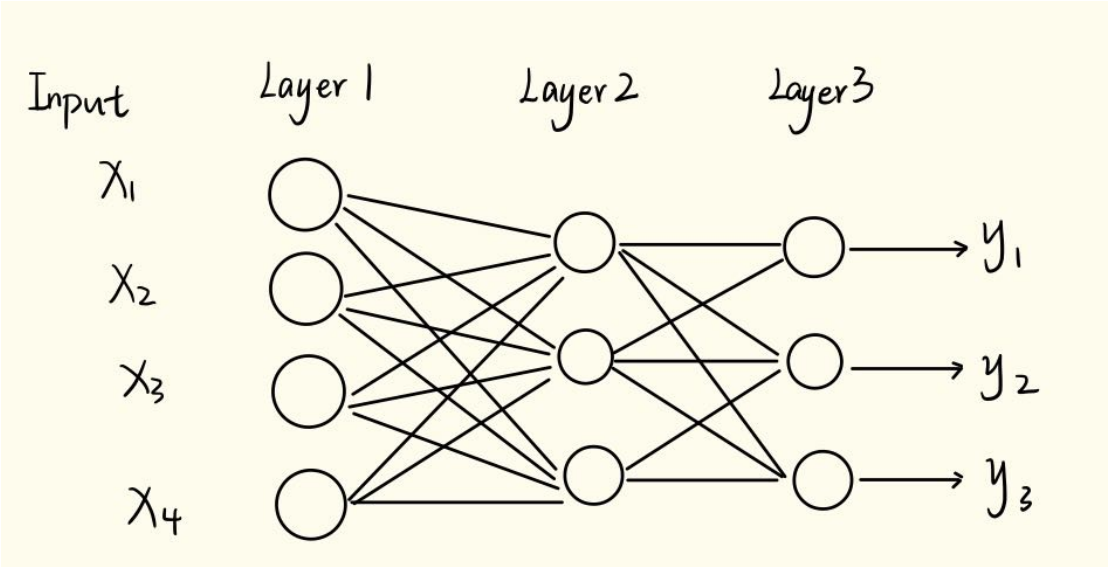
3-layer Neural Network

I tried to implement a 3-layer neural network to make a classification on Iris dataset. Everything goes smoothly except that some unsolved mistake occurs in the calculation of gradient of softmax activation function. The following content is a introduction of current work.

1. Design idea

The goal of this 3-layer neural network is to make a classification on Iris dataset. As we know logistic regression is an essential part of implementing a neural network, after getting our predicted output, we need to compare it with correct label y and calculate the loss –----- this is called forward propagation. This process works well for many dataset. However, we notice that the correct label y is contained by categorical values, and there are three categories. That is, we need to implement a multiclass classification and make some operations to y in order to make it work in logistic regression. One hot encode is a popular choice for us which converts categorical data to binary values. Besides, softmax needs to be our last activation function. The first activation I adopt is ReLU activation function.

This network would look like as follow:

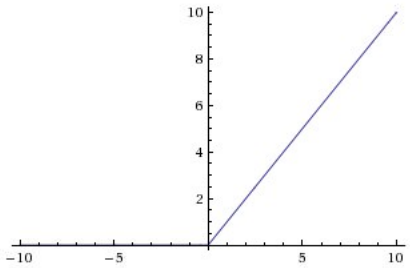


The input X has four features which correspond to width and length of sepal, width and length of petal. The output Y corresponds to three species of iris flower: Iris-setosa, Iris-versicolor, and Iris-virginica.

2. Activation functions

2.1 ReLU (Rectifier linear unit)

The rectified linear unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as $f(x)=max(0,x)$. Graphically it looks like this:



It's surprising that such a simple function can allow your model to account for non-linearities and interactions so well. But the ReLU function works great in most applications, and it is widely used as a default activation function.

2.2 Softmax

Softmax function takes an N-dimensional vector of real numbers and transforms it into a vector of real number in range (0, 1) which add up to 1. For each i=1, ..., n, the i-th coordinate of softmax(x_1, ..., x_n) is defined to be

$$\frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

As the name suggests, softmax function is a "soft" version of max function. Instead of selecting one maximum value, it breaks the whole (1) with maximal element getting the largest portion of the distribution, but other smaller elements getting some of it as well.

This property of softmax function that it outputs a probability distribution makes it suitable for probabilistic interpretation in classification tasks.

3. Cross entropy loss

Cross entropy indicates the distance between what the model believes the output distribution should be, and what the original distribution really is. It is defined as, $H(y, p) = -\sum_i y_i log(p_i)$. Cross entropy measure is a widely used alternative of squared error. It is used when activations can be understood as representing the probability that each hypothesis might be true, i.e. when the output is a probability distribution. Thus it is used as a loss function in neural networks which have softmax activations in the output layer.

4. Vectorization

In order to improve the performance of neural networks and get rid of for loops, I use matrix to store the values. Specifically, in Iris dataset, there are 150 instances. Therefore, the input $\boldsymbol{X}$ is a matrix with dimension 4x150, 4 comes from four features of each instance. The weight of layer 1 is a matrix of shape 3x4, the bias of layer 1 is a column vector with dimension 3x1. Similarly, the weight in layer 2 is a matrix of shape 3x3, the bias of layer 2 is also a column vector with dimension 3x1. The calculations between matrix are done with help of Numpy, such as Numpy.dot(), Numpy.sum(), Numpy.mulitply(), etc.

It turns out that vectorization not only makes our networks more efficient but also make it logically more clear.