Data Mining Homework 2 Report

Introduction

Classification methods have gained importance in many fields as obtaining data has become easier. However, due to the sheer amount of data available, it remains time-consuming to determine the best way to classify the items in a dataset. Data can come in many forms. Most of the time, we don't have a reference class for the data we use in training our models, that is, the data is unlabeled and the problem is "unsupervised". Depending on the objective of the classification and the nature of the data, different algorithms are more appropriate. For instance, a Decision Tree classifier, as used in this assignment, is used with labeled data, and provides an intuitive and visual explanation for its decision. On the other hand, neural networks act more like a "black box" since the programmer designs the model and then the parameters change during the backpropagation step in training.

In order to further understand the performance of different classification, three classification methods were implemented: a Decision Tree Classifier, a Naive Bayes classifier, and a simple neural network. These classifiers were then tested using two datasets: the famous iris dataset, which contains 150 samples of four attributes each, with three classes; and a custom dataset with 427 training samples each with 30 attributes, each sample belonging to class 0 or 1.

The iris dataset was chosen due to its small size, which was used as a test during the development process. It was also useful to see if there was any relevant difference in the same model with data containing different amounts of attributes. The second dataset, the custom dataset, was selected to measure the resilience of our models with more complex data. This dataset had already been used in a previous related assignment, so it already was a fitting dataset for a classificaiton problem. Both the datasets contained an approximately equal proportion of samples from each class.

Training and "Real World" Data

When training to perform a classification task, the relation between the training and testing data is important. If the training data is not representative of the testing data, the information learned during training will not adequately predict the testing data. For the datasets used in this assignment, we used part of the training data for testing with sklearns `train_test_split` function, which gave us 80% of the data to use for training and 20% for testing.

While this approach helped us ensure similarity between the training and testing data, when a model is deployed this is not usually the case. A company might have to collect its own training data to produce the best model. A model which can learn useful information from the training set and apply it in the testing set is said to **generalize** well. Generalizaiton and its specifics are an entire area of research in its own. Thus, to enusre the best possible performance, the two important factors to consider are the nature of the training and testing set, as well as the model's ability to apply learned informaiton to an unseen dataset.

Decision Trees

Decision Trees have existed for many decades, and remain a useful model because they are intuitive and easy to understand for many people. Decision Trees are trees whose leaf nodes correposond to class labels and whose internal nodes correspond to a decision. When assigning a class to a test item, we perform a test at each internal node, e.g. *is feature a of item $j \geq x$?* After splitting the dataset according to this test, we recurse on the left and right children. When we reach a leaf node, we assign the leaf node's class label to our current item. The internal nodes can perform a binary split or they can have a set of buckets as their children depending on the characteristics of the data. In this assignment we implement binary decision trees, which might sacrifice some accuracy at the cost of simplicity of implementation.

The training and building of the decision tree is done as follows. At each node, we take in a dataset D. Then we determine the best feature to perform the split on and the best value of that feature according to our attribute selection metric, discussed shortly. Once we find the best attribute, we split D according to the attribute selection metric. Then we create a new right and left child and recurse with the split datasets. Since the tree is binary, the test we perform checks whether the each value of the selected feature in D is greater than or less than the value given by our attribute selection metric.

Attribute Selection Metric

How do we find the features to perform the split on at each internal node? We need some way to quantify the quality of using a certain value to split our dataset. While there are many measures, such as information gain and the gain ratio, in this implementation we used the Gini index.

The Gini index measures the *impurity* of a dataset, given by

$$\text{Gini}(D) = 1 - \sum_{i=1}^{m} p_i^2 \tag{1}$$

where m is the number of classes and p_i is the probability that a sample belongs to class i. This measures the amount of "impurity" or mixture in a dataset. As p_i becomes greater, meaning that more items in the dataset belong to the same class, the gini index will be lower.

Since we are checking the purity of the proposed split, the total Gini index of the attribute is the weighted sum of the gini index of both datasets.

$$\text{Gini}_A(D) = \frac{|D_1|}{|D|}\text{Gini}(D_1) + \frac{|D_2|}{|D|}\text{Gini}(D_2) \tag{2}$$

The final decision for the attribute and split value is done by choosing the value that mimiizes the

difference

$$\Delta\text{Gini}(A) = \text{Gini}(D) - \text{Gini}_A(D) \tag{3}$$

Naive Bayes Classifier

The second classsifier that was implemented was the Naive Bayes classifier, which assigns the class based on the result of Bayes' Theorem

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)} \tag{4}$$

Where $P(C_i|X)$ is the probability a sample X belongs to class C_i. The algorithm is straightforward. For each of the m classes, we calculate the probability that the given input belongs in that class and choose the class that maximizes $P(C_i|X)$.

Since $P(X)$ is a constant for all classes, we only consider the top part of the fraction. The posterior probability $P(X|C_i)$ is estimated as a random normal variable with mean μ_{C_i} equal to the proportion of items belonging to $C_i \in D$ and likewise σ_{C_i} the standard deviation of all items belonging to the class. Finally, $P(C_i)$ is obtained from $|C_i|/|D|$. The class which maximizes $P(C_i|X)$ is the class we assign to the input.

Neural Network

Neural Networks have become increasingly popular in recent years. Their ability to learn any arbitrary linearly separable data made it an excellent candidiate for this assignment. The neural network used in this assignment is a simple feedforward neural network, where each neuron in layer l takes in the outputs of layer $l-1$, multiplies it by a weight and adds a bias, then runs it through an activation function. In the final output layer, the value with the highest activation is the final class assignment.

Neural networks presented the most challenging model due to my own personal inexperience with them: tuning the hyperparameters, choosing the amount of hidden layers and neurons in those hidden layers, etc… is a technique where I don't have much experience. However, I attempted to design a simple network with only one hidden layer.

The input layer to our model has the same amount of neurons as features in our dataset. The hidden layer in our network also has the same amount of neurons, one for each attribute in the dataset. At the output layer, there is one neuron per class label.

Results

As mentioned, two datasets of different attribute counts were used to test our models. The first test, the iris dataset, contains 4 attribute columns and 150 samples. The second dataset contains 427 training samples each with 30 categories. Following are the accuracy results for teh models on the corresponding datasets. For reference, the `sklearn` implementation of Decision Trees and Naive Bayes were also considered.

	DT (sklearn)	DT (ours)	Naive Bayes (sklearn)	Naive Bayes (ours)
iris dataset	0.932	0.922	0.922	0.956
custom dataset	0.558	0.597	0.582	0.655

The neural network trained for 100 epochs, with batch size of 32 and learning rate of 0.25. The optimizer used was Stochastic Gradient Descent. The neural network accuracy is as follows

	3-Layer Neural Network
iris dataset	88.6
custom dataset	61.3

From the first table, it can be seen that the increase in attributes has a large effect on our model's accuracy. There are more clear separations between the items in the iris dataset, which is why the accuracy remains higher both on the sklearn model as well as our own. Wihthout pruning or without setting a maximum tree height, the decision tree can also overfit on the training data, which would give a larger difference on the custom dataset.

The Naive Bayes classifier performed about as well as our Decision Tree and `sklearn`'s. Even though this is a simpler classifier, it can still achieve good results.

For the neural network, although a small network managed to achieve relatively good performance on the iris dataset, I believe that underfitting of the data prevented the model from achieving better performance on the custom dataset. In classification problems we need to ensure that we have enough variables to model the relationships in the dataset. However, with the small model used here it might not have been enough. In a future implementation, more consideration would be given to the hyperparameters and structure.

Conclusion

This assignment attempted to gain an understanding into different classification methods and their peformance, as well as the relation of the real and observed data. These models rely on the data we use to train them: the attribute we use in the internal nodes of a Deicision Tree, or the weight of a neuron in a neural network all depend on the training data. If the training data is not a good representation of the testing data, the accuracy will not be as high.

The three models selected were tested with two datasets to measure the difference between accuracy in datasets with larger attributes. It was observed that the accuracy went down as the number of attributes went up in our dataset.

There are areas of data science/machine learning looking into looking at the information stored in neural networks, for example, knowledge distillation in neural networks. For complex models such as neural networks, it is worthwhile to understand what exactly is happening “under the hood”. In the future, further research could let us have a clearer understanding of how these networks make decisions. Regardless, the algorithms we have for parsing labeled data is impressive already, and represents the long way people have come from having no way of analyzing labeled data.