Comparison of Machine Learning Classification Techniques

Alexander Crawford (100569102)

Jeremy Kwok (100341977)

Devin Westbye (100522109)

Artificial Intelligence

University of Ontario Institute of Technology

April 6th, 2017

One common problem in machine learning is that of statistical classification. Statistical classification is the act of identifying and classifying data points into distinct categories. For example, we might wish to classify cars into classes given their price, horsepower, and other features, or emails into spam or not spam given the presence of certain keywords. Classification is a type of supervised learning; algorithms take a set of training data that it uses to infer the classes of new points of data. There are many types of classification algorithms, each with their own advantages and disadvantages.

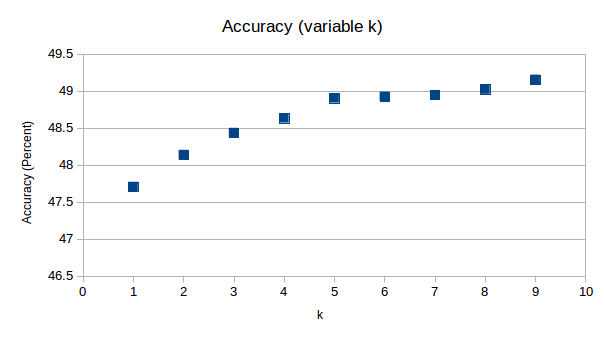
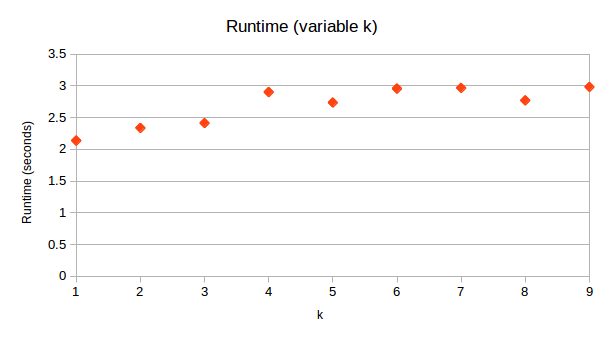
For this project, we have selected three different classification algorithms to examine: k-Nearest Neighbour, CART, and the Information Fuzzy Network. We have included, along with the analysis in this report, implementations of those three algorithms. These programs were written in Python. Python was selected as the coding language for the algorithms, given its high functionality, large scientific calculation libraries, clean, readable code, and integrated graphing technology. To test our implementations, we have compared them using a singular dataset. The dataset selected for this project related to the quality of red wine, based on many chemical factors: <https://archive.ics.uci.edu/ml/datasets/Wine+Quality>.

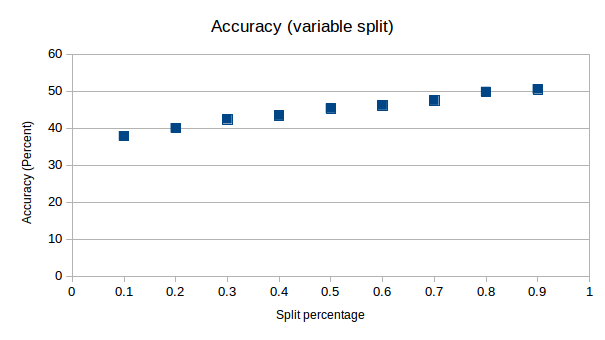
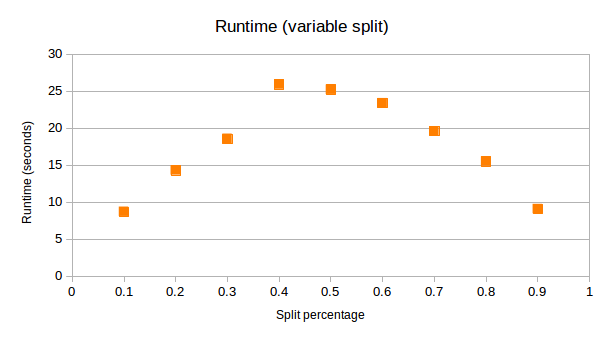
The first algorithm we will look at is k-Nearest Neighbour. k-Nearest Neighbour is a supervised learning method that can be used to classify data, or for regression analysis. k-Nearest Neighbour is often used as an introductory algorithm to machine learning, and is one of the first machine learning algorithms taught to programmers and students.

The underlying principle in k-Nearest Neighbour is that objects or data points will have similar properties to other data points in close proximity. Whenever we obtain a new data point we want to classify, we take a look at its closest k number of neighbours, defining closest as points with the smallest Euclidian distance to our new data point. The algorithm will then examine the neighbours and attempt to discern which class it belongs to. It does this by setting the new data point to be in the same class as the most number of its neighbours. For example, we can examine how the algorithm wants to classify sex based on height and weight . The algorithm would take a new data point, such as 185 cm tall and 70 kg heavy. It then looks at the k (a number specified by the user) closest points, sees that they are all male, and is able to say with certainty that the new data point is male. If it looks at multiple points and sees that some are male and some are female, then it picks whichever class is more populated amongst the neighbours. There are some extensions to the k nearest neighbours algorithm that uses a certainty value when all the neighbours do not agree, but this implement for the project simply tracks the majority class of the neighbours.

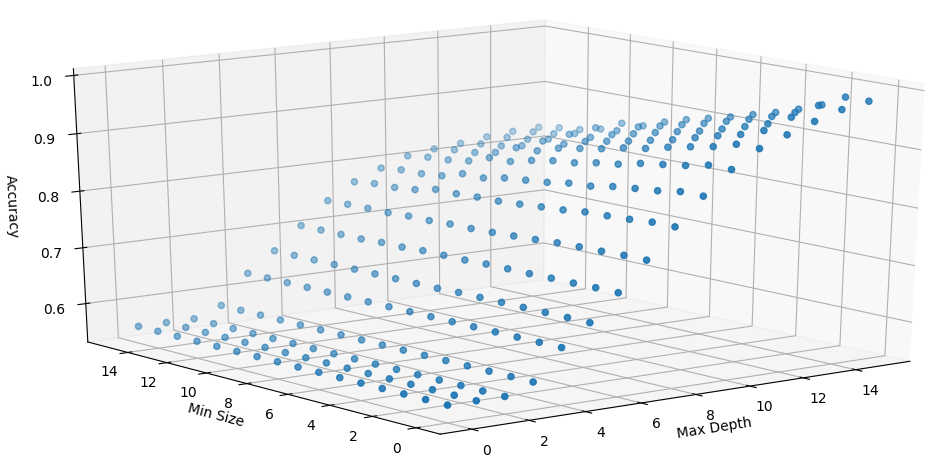
For the implementation, the data set is split randomly into a training set and a test set. One by one, the test set is read into the program, and we perform the classification based on the nearest elements of the training set. Each of those neighbours are counted and classified, and the resultant class of the test set entry is noted. The first experiment that was done with this algorithm is determining the effect of an increasing size of the training set. As noted in the graph below, we can see that the accuracy of the algorithm does increase as the training set increases. In addition, the runtime of the algorithm increases as we add to the training set, and starts to decrease once we hit the peak of 40% of our data being in our training set. This can be explained since we need to calculate the distance of each member of the training set to each member of the test set, which simplifies to n \* m, where n is the size of the training set and m is the size of the test set, so it stands to reason that the maximum runtime would occur when the data set is split in half. The second experiment that was done was the effect of increasing values of k. We can see that the runtime increases for increasing values of k. This is in line with our hypothesis, as greater values of k means more neighbours need to be checked for each member of the test set. The accuracy does increase with increasing values of k, but that increase tapers off as checking against more neighbours eventually reaches diminishing returns.

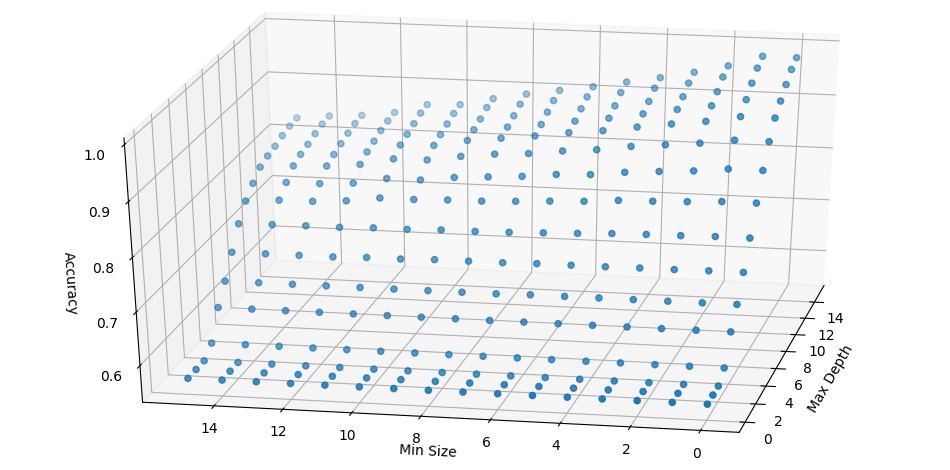
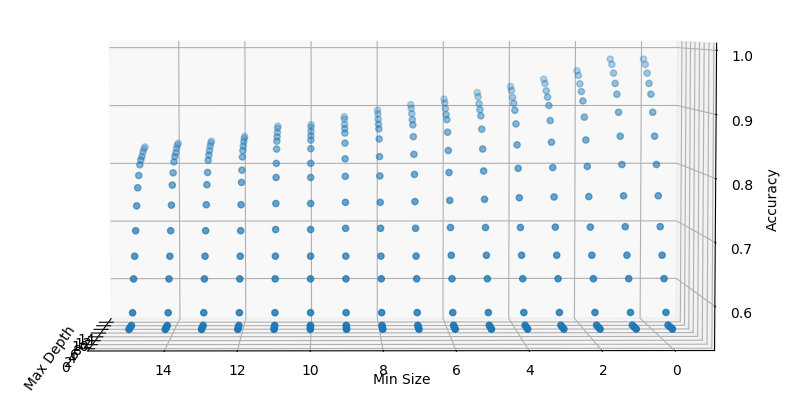
As k-Nearest Neighbours is a fairly basic and simplistic algorithm with little optimizations, the accuracy of this algorithm is quite low compared to the other algorithms that we will look at. The advantages are that it is easy to understand and easier to implement than the alternatives.

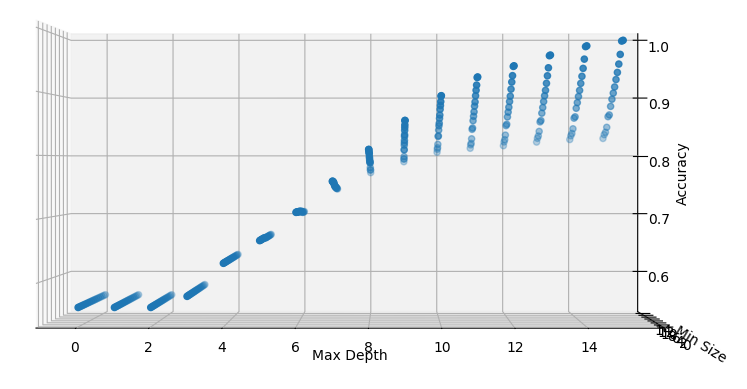




The CART algorithm is a supervised machine-learning algorithm that constructs a binary decision tree from some input data. The algorithm works by iteratively selecting a node in the input data to split on, by comparing the calculated gini impurity with the current best. The data is split by comparing the values of the available data to the value of the currently selected node. Any nodes with a smaller value fall into the “less” list, while everything else goes into the “greater” list. This process is repeated until stopping conditions are met, such as exceeding the maximum depth from the root node, or when the size of the children of a node fall beneath a certain value. In either case, a value is selected to be the terminal node, rather than it leading to more child nodes. These values, or terminal nodes, represent the actual classification value of the input data. When the decision tree is applied to new data, the nodes of the tree will compare certain values in the data to itself, and this will determine which path, less or greater, the next iteration should take. Finally, it will lead to the terminal node which will return the class value the data has been placed under. A well trained tree will classify data with a high degree of accuracy, while poorly trained trees will do the opposite.

Aside from the programming of the CART algorithm itself, a few scripts were also created to construct decision trees at increasing values of maxDepth and minSize, which determine the stopping criteria for the algorithm. The trees were then tested against the original training data, to compare how accurately each value was classified. This information was plotted in a 3D grid, as accuracy depends on both maxDepth and minSize.





From these plots it can be seen that as maxDepth increases, so too does accuracy.Inversely, as minSize increases, accuracy decreases. Furthermore, variations in maxDepth have a larger effect on accuracy than minSize does.

The overall performance of the CART algorithm to construct the decision tree on the given dataset is quite slow. It has a slow run time starting with small values of maxDepth, and large values of minSize and only gets worse as the values increase. Runtime for maxDepth of 1 and minSize of 16 takes 17.209 seconds, while runtime of maxDepth of 16 and minSize of 1 takes 42.097 seconds. However, once the tree has been constructed, the runtime of applying the tree to new data is quite fast. For example, runtime of testing all 1599 values in the dataset on the constructed tree for maxDepth 16 and minSize 1 takes 0.008651 seconds.

The Information-Fuzzy Network (or IFN) is a collection of supervised machine learning algorithms that, like many other such algorithms, work to predictively classify a single attribute for a number of objects based on the other attributes of those objects [3]. However, the common theme between all such IFN algorithms is that they are based on the concept of “fuzzy logic” - that is, an item’s membership in a given set is not simply “true or false”, and is instead a decimal value between 0 and 1, where 0 is a non-member and 1 is a member. While multiple such algorithms exist, this report focuses on the one outlined in the Data Mining and Knowledge Discovery Handbook [3].

The first part of the algorithm involves “fuzzification” of the input data. Any attributes from the input data that are not already represented as mutually exclusive groups are quantified into several groups by using fuzzy mean clustering; in this method, random means are set and then iterated upon, slightly changing the center value of the means based on the locations of all set members. At the end, the best mean positions provide a fuzzy partition coefficient, which is a numerical value between 1 and 0 that represents how well the items are grouped, given the number of means and how distinct the means are (that is, with a high fuzzy partition coefficient, if an item is a strong member of group A, it’s likely to be a weak member or a non-member of groups B, C, D, etc.). This algorithm is run multiple times for different target numbers of means to determine what the ideal number of means are, and a ‘fuzzified’ data set is then produced where each object’s attribute is replaced by how much of a member it is of the given mean group. This is repeated for each attribute until the data set is completely fuzzified.

When all training data is fuzzified, the second and final stage of the training stage begins: the construction of the network. This algorithm begins by selecting which of the data’s attributes’ fuzzy partitioning had the highest fuzzy partition coefficient (i.e. creating the most distinctive split in values). For each group in that fuzzy partition, another node is added to the tree, representing membership in that group, and new subsets of the data are constructed based on whether each data point is a member of that group; because of the nature of fuzzy sets, this may mean that a single data point may appear in multiple nodes at the same level in the tree, if it has a sufficiently high membership in those fuzzy groups. Finally, in each of these subset groups, the target attribute is examined, to determine whether this subset conclusively represents a single “truth value” (that is, whether a sufficient amount the data falls into one category, such as True or False). If it is, a path is constructed to the target node; if not, the algorithm runs recursively on this subset, further splitting it based on the next-most-clear attribute, and so on. A node is classified as a leaf of the tree when either the data conclusively represents one category (in which case it is connected to the node of that category), there are no more attributes to possibly classify the data on (in which case it is connected to the node of the category it most closely represents, even if it is not a perfect match), or if there are no nodes in that subcategory (in which case it is classified using the closest category match of the operation directly preceding it).

When the tree is constructed, new data can be tested by fuzzifying its input and following appropriate nodes on the tree. As the system saves the means previously generated during the training phase, the fuzzification of the input is a linear operation, and following the nodes in a tree can be as simple as a linear operation as well (if you only test based upon the groups a piece of test data most closely matches). Alternatively, in exchange for taking a longer operating time and exploring all nodes of the tree that sufficiently match groups of the test data, you may reach multiple conclusion nodes, and be able to also output a percentage certainty of your system’s match. Using the naive approach (only following one node per choice), the simple approach of the system reached an accuracy of over 80% on a sample set of test data, with a runtime of roughly 6 seconds, regardless of input parameters (beyond data to fuzzify).

Throughout this report, we have examined three different major classification algorithms: k-Nearest Neighbour, CART, and the Information Fuzzy Network family, each with their own unique advantages and disadvantages. While there is no perfect algorithm, each has its place depending on the goals of your classification and the attributes of your dataset. Because of the varying strengths and weaknesses of each approach, it is important for programmers and engineers to be aware and understand many different algorithms to be as efficient as possible.   
  
  
[3] O. Maimon et al. Data Mining and Knowledge Discovery Handbook. Springer: 2005. Print.