COMP4121 Major Project

Discussions and Implementations of Several Classification Algorithms

UNSW SYDNEY

Gary MacNamara
z5312530

Contents

1 Introduction to classification

Classification is the problem of taking known observations or data and predicting what category or class they belong to based on previous data. Classification allows us to understand the relationship between known features and the set of objects they belong to, and has importance in many fields, including statistics (in the grouping of data), biology (in identification of organisms), and computer science (in machine learning).

1.1 The problem statement

Generally, in a classification problem we are given a training set consisting of n objects $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}$ where each object is a vector with f features/observations. We are also given n labels $\mathbf{y_1}, \mathbf{y_2}, ..., \mathbf{y_n}$, where $\mathbf{y_i}$ describes the class that object $\mathbf{x_i}$ belongs to. The job of a classification algorithm is to process this training set, and then given a new test object $\mathbf{x}_{\text{test}}$, to predict the class $\mathbf{y}_{\text{test}}$ that it belongs to.

This paper will discuss six popular classification algorithms, each with different strengths, weaknesses, and use cases. It will show implementations of these algorithms and compare their performance on synthetic and real data sets. These implementations as well as the test data are available at **https://github.com/Linoed/c4121**.

2 K-nearest-neighbours

The K-nearest-neighbours algorithm (KNN) is an algorithm used in classification and it's one of the simplest, most intuitive algorithms.

2.1 How it works

KNN has no training phase, all the computation happens after it is given test data. When considering an object $\mathbf{x}_{\text{test}}$, KNN checks the k closest objects in the training set based on a chosen distance metric, and predicts that $\mathbf{y}_{\text{test}}$ is the majority class of those neighbours.

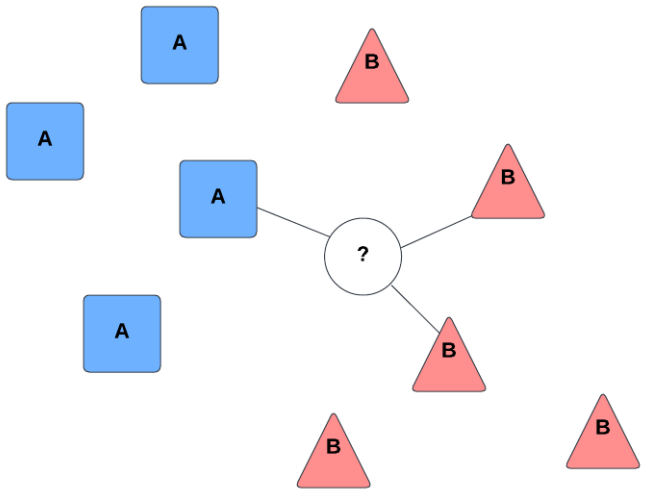

Figure 1: Showing how KNN with $k = 3$ would classify a new test object with euclidean distance as the distance metric. Here, KNN predicts that the object is class B.

2.2 Distance metrics

KNN is able to classify data sets that are clustered and have a conception of 'distance' between objects. The distance metric can be adjusted to allow KNN to classify objects with numerical features, strings, and images amongst other types of data.

2.2.1 Euclidean distance

Euclidean distance is a common metric used for KNN, and the euclidean distance between two objects $\vec{u}$ and $\vec{v}$, each with f features $D(\vec{u}, \vec{v})$ is given by the formula

$$D(\vec{u}, \vec{v}) = \sqrt{\sum_{i=1}^{f} (\vec{v_i} - \vec{u_i})^2}.$$

2.2.2 Manhattan distance

The manhattan distance represents the distance between two points on a grid, and the manhattan distance $D(\vec{u}, \vec{v})$ is given by the formula

$$D(\vec{u}, \vec{v}) = \sum_{i=1}^{f} |\vec{v_i} - \vec{u_i}|.$$

2.2.3 Hamming distance

The hamming distance is equal to the number of features in $\vec{u}$ that are different to their respective features in $\vec{v}$. This is useful for comparing strings, because one might want to consider the strings "abce" and "abcf" to be equally different from "abcd", even though

'e' is closer to 'd' than 'f' is. The hamming distance $D(\vec{u}, \vec{v})$ is given by the formula

$$D(\vec{u}, \vec{v}) = \sum_{i=1}^{f} h_i(\vec{u}, \vec{v})$$

where

$$h_i(\vec{u}, \vec{v}) = \begin{cases} 1 & \text{if } \vec{u_i} \neq \vec{v_i} \\ 0 & \text{if } \vec{u_i} = \vec{v_i} \end{cases}.$$

2.3 Choosing a value for k

The choice of k is important to consider. Too little, and the model is oversensitive to noise. Too high, and the model will have a monotonous predictions that fit the average of the data set. A good value of k is not something that can realistically be calculated, ideally it is tuned empirically in a way that minimises mislabeling.

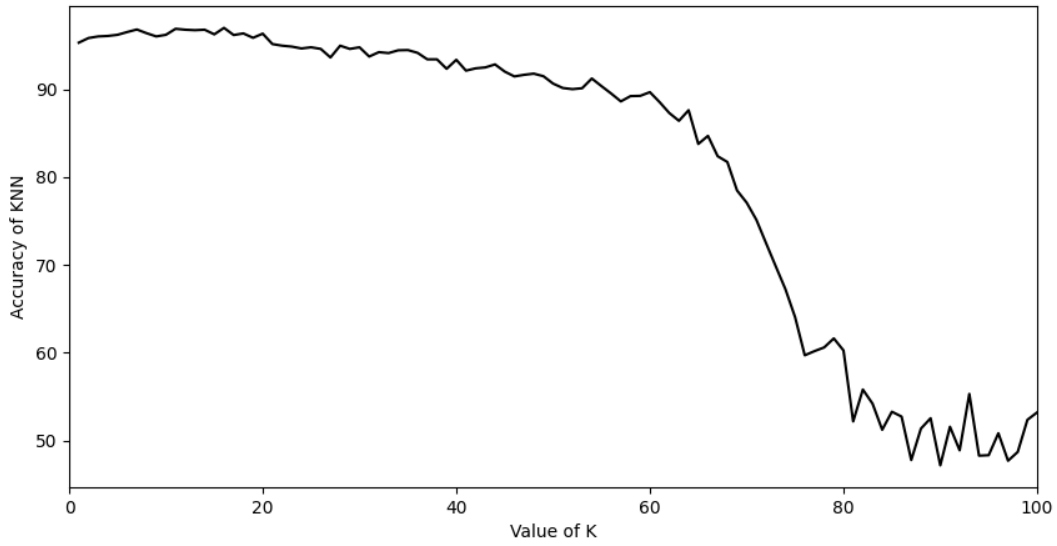

Figure 2: The accuracy of my own KNN algorithm on the Iris data set as k is varied from 1 to 100 (100 trials each, with $\frac{1}{4}$ of the data set being used to test each time). Here, the accuracy reaches a maximum at $k = 16$.

The optimal value of k also depends on the training data given. If training data heavily biases a certain class A, then a high value of k will make KNN bias towards predicting class A as it treats all k neighbours as having equal 'voting' power. Another issue is the resolution of ties. If multiple classes have the same number of votes for a certain test object, then if those votes are maximal the test object has equal likelihood of being any of the majority classes. In the case of bipartite data, one way around this is to choose k to be an odd integer, making ties impossible.

2.4 Conclusion

2.4.1 Benefits

In KNN, closer objects are considered more likely to belong to the same class, which is simple and intuitive. Additionally, KNN is able to classify both binary and multiclass problems, and its flexibility in choosing distance metrics allows for classification of data like strings and graphs. Finally, KNN uses one hyperparameter (an externally inputted parameter) k, which allows it to be easily tuned in the manner shown in Figure 2.2.

2.4.2 Drawbacks

Classifying a new test object requires calculating the distance between the object and all n training objects, and then getting the k lowest distances. The calculation of all n distances is done in $\mathbf{O}(nf)$ time. Using quickselect to get the k-th smallest distance is $\mathbf{O}(n)$, so the algorithm can be implemented optimally in $\mathbf{O}(nf)$ time, which means the algorithm scales poorly with the size of the training data and its dimensionality.

Additionally, KNN requires that distances of different features are the same. In a hypothetical data set, if one of the features was in meters, and another in light-years, it's easy to see that the euclidean distance formula wouldn't treat this data-set fairly, and another formula would have to be used. Potentially, there might not even be a distance formula for some vectors. Some features could be numerical, while at the same time others could be categorical (like red and blue). It's not clear that a 'distance' even exists in this case. Finally, as stated before, KNN is vulnerable to bias in the data set, as an imbalanced set would cause it to bias the predominant class.

3 Decision tree

Decision trees are a classification tool that use a tree data structure to predict the class of new objects.

3.1 How it works

Decision trees are a tree data structure with a root node, decision nodes with conditionals, and leaf nodes with class descriptions.

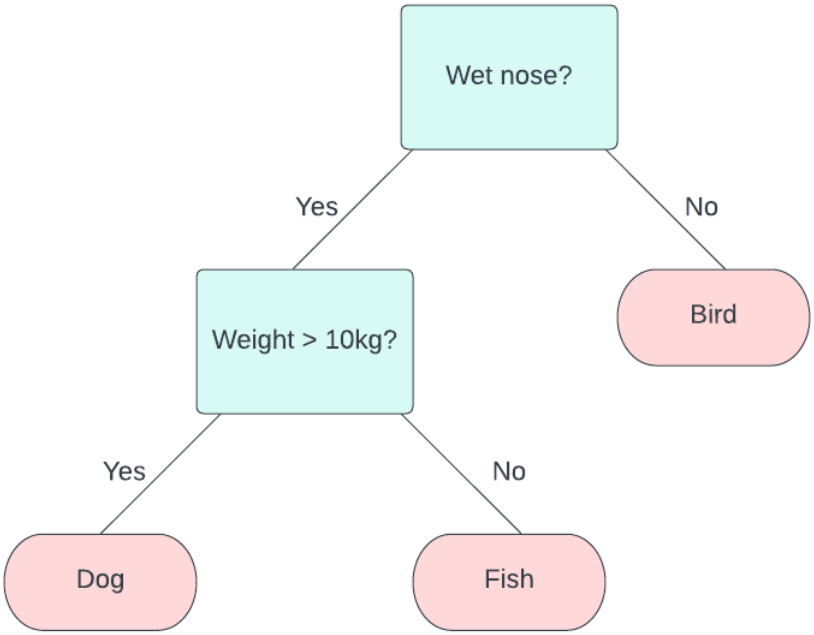

Figure 3: A decision tree for classifying between dogs, birds, and fish using their weight and if their nose is wet.

In implementation, nodes have a target feature and a conditional. These features are checked with the conditional against a test object's features, and we move down the nodes until a leaf node is reached. Once a leaf node is reached, our prediction is the class contained in the leaf node. In Figure 3, we would first check if the object had a wet nose, if not we would predict it was a bird. If it had a wet nose and its weight was greater than 10kg, we would predict a dog. Otherwise we would predict a fish.

While the prediction algorithm is fairly simple and intuitive, it is more challenging to create the tree from data. While different algorithms use different selection measures and procedures, the general idea is that at each node, we want to split the data as cleanly as possible. To do this, we check every feature of every object in the training data. We take the value of that feature v and split the data into two sets - objects with a feature value less than v and objects with a feature value greater than v. After considering all the possible splits, we choose the 'cleanest' split based on some selection measures.

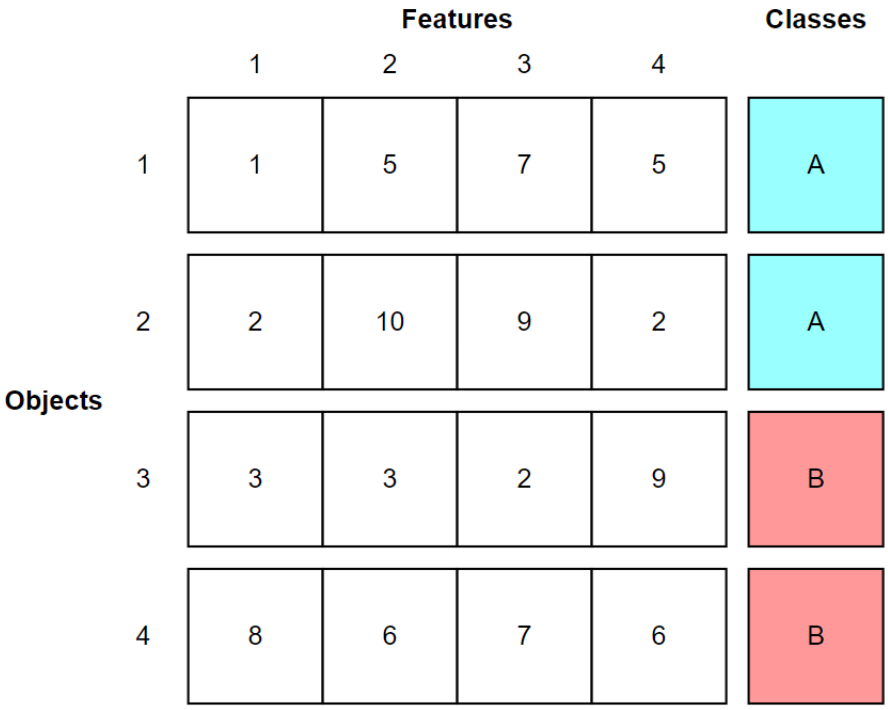

Figure 4: An example data set.

For example, in Figure 4, we could consider the split caused by considering feature 2 of object 1, which has a value of 5. Splitting the data by if feature 2 ≥ 5 would split it into two groups, the 'yes' group being objects 1, 2, and 4, and the 'no' group being object 3. A cleaner split could be made by choosing feature 4 of object 4, which would result in the tree below, a completely clean split.

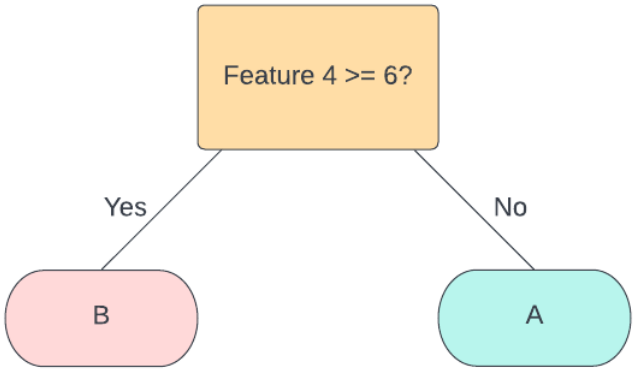

Figure 5: A decision tree made by splitting feature 4 of object 4 in Figure 4.

The whole algorithm is as follows. Starting with all the data, we consider the cleanest split, and split the group into 'yes' data and 'no' data. If a split group's classes are homogeneous, a leaf node is created on the left with its class if it's a 'yes' or on the right if it's a 'no'. If a group had mixed classes, we then perform the same algorithm on it, putting a decision node on the left for 'yes' data, or on the right for 'no' data.

Decision tree algorithms can't always get a clean split. Some algorithms split the data over and over again, potentially to the point that only a single training object is differentiated by a split. Not only is this inefficient, this can cause overfitting, where the model attempts to align to outliers, causing good performance against training data, but poor performance against unknown data. To counter this, some algorithms have a depth limit, and when their node hits the limit, instead of recursing further, the algorithm creates a leaf node with the majority class of its assigned data.

3.2 Selection measures

In section 3.1, it was shown that decision tree algorithms rely on making choices that lead to the 'cleanest' split. This section will describe the different selection measures that decision tree algorithms use to quantify how 'clean' a split is.

3.2.1 Entropy and information gain

The goal of decision tree algorithms is to split the data into homogeneous sections of data. Information content is a measure of the randomness, or how 'surprising' data is. For example, something very unlikely happening in a data set conveys much more information than something that was very likely to happen. As such, the information content $\mathbf{I}(\mathbf{E})$ of an event $\mathbf{E}$ with probability $p(\mathbf{E})$ is given by the function

$$\begin{aligned}\mathbf{I}(\mathbf{E}) &= \log\left(\frac{1}{p\left(\mathbf{E}\right)}\right)\\ &= -\log(p(\mathbf{E})).\end{aligned}$$

This is simply because the information gain of seeing an event with probability 1 should be 0, and $\log(1) = 0$. Similarly, as the probability of an event tends to 0, the inside of the log tends to infinity, and thus the total information gained is large, as expected.

Entropy is defined as the expected amount of information content, or 'surprise' of a data set. This means the entropy $\mathbf{H}(\mathbf{X})$ of a random variable $\mathbf{X}$ can be modeled by

$$\begin{aligned}\mathbf{H}(\mathbf{X}) &= \mathbb{E}[\mathbf{I}(\mathbf{X})]\\ &= \mathbb{E}[-\log p(\mathbf{X})]\\ &= -\sum_{x\in\mathbb{X}} p(x)\log p(x).\end{aligned}$$

Because decision tree algorithms attempt to homogenise data, the goal is to minimise entropy. This is where information gain is important. Information gain is defined as the difference in entropy before a split and the average entropy of the two split data sets, so higher information gain means less entropy. Hence, we can say that information gain $\mathbf{Z}(\mathbf{S})$ of splitting a data set $\mathbf{S}$ into a left set $\mathbf{L}$ and right set $\mathbf{R}$ follows the formula

$$\mathbf{Z}(\mathbf{S}) = \mathbf{H}(\mathbf{S}) - \frac{|\mathbf{L}|\mathbf{H}(\mathbf{L}) + |\mathbf{R}|\mathbf{H}(\mathbf{R})}{|\mathbf{S}|}$$

where $|\mathbf{S}|$ is the number of objects in set $\mathbf{S}$.

Thus, if a decision tree algorithm were to use information gain, it would consider each split, calculate the information gain of the split, and then greedily take the split with the highest information gain.

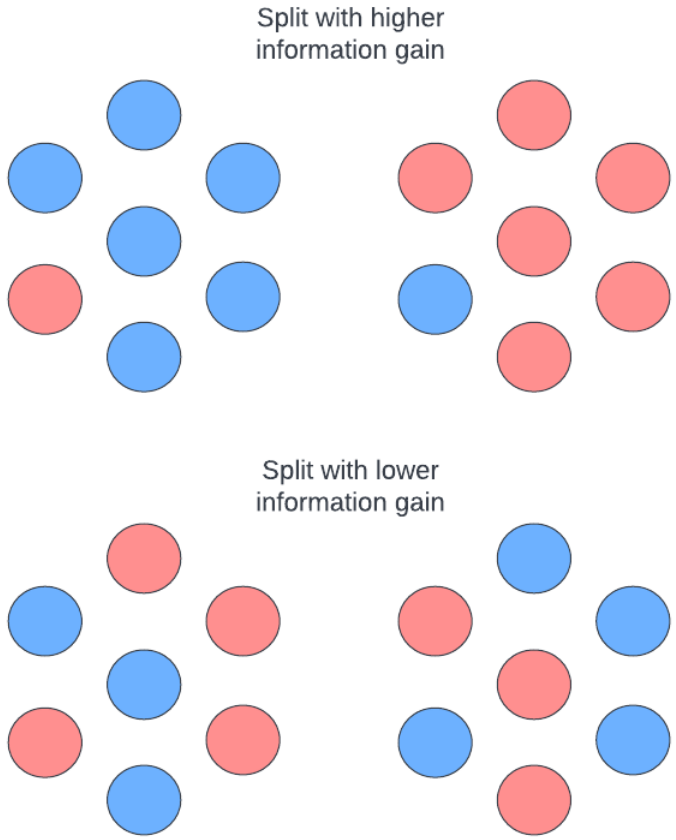

Figure 6: An example of the relative information gain from splitting into two data sets, one on the left and one on the right. A decision tree algorithm would prefer the top split.

3.2.2 Gini index

A potentially simpler selection measure is the Gini index. The Gini index measures the purity of a distribution, and ranges from 0 (denoting a homogeneous distribution) to 1 (denoting a random distribution). While the derivation is quite extensive, the formula for the Gini index $\mathbf{G}$ over C classes in a data set is given by

$$\mathbf{G} = 1 - \sum_{i=1}^{C} p(i)^2$$

where $p(i)$ is the probability of seeing class i in the data set. It is clear to see if a data set contains only one class, then this formula evaluates to $\mathbf{G} = 1 - 1 = 0$, denoting a pure data set. On the other hand, the Gini index is maximised when the summation is minimised, which is when all $p(i)$ are equally likely.

Thus, if a decision tree algorithm were to use a Gini index, it would consider each split, calculate the Gini index $\mathbf{G}$ of the left split $\mathbf{L}$ and right split $\mathbf{R}$, then take the split

which minimises the value

$$\frac{|\mathbf{L}|\mathbf{G}(\mathbf{L}) + |\mathbf{R}|\mathbf{G}(\mathbf{R})}{|\mathbf{L}| + |\mathbf{R}|}.$$

3.3 Conclusion

3.3.1 Benefits

Because they choose the best features to predict with, decision trees are able to work with data that is less preprocessed or has missing features. Decision trees also have no problem dealing with categorical and numerical data. Additionally, unlike KNN, decision trees don't require normalisation or scaling of data as they don't rely on distances. Finally, after training, the time-complexity of predictions in a decision tree is linear in the depth of the tree (so best case logarithmic, worst case linear), and doesn't suffer from the curse of dimensionality like KNN.

3.3.2 Drawbacks

Decision trees can get unnecessarily complex, based on the depth that the tree is allowed to go. Additionally, training a decision tree is a fairly expensive operation, as it checks all future state spaces at every split. Decision trees are also vulnerable to overfitting if pruning is not used.

4 Random forest

Random forest classifiers aim to be an improvement on decision trees.

4.1 How it works

Random forest classifiers take a large amount of decision trees that see sub-samples of the training data as opposed to all of it, and in predicting the class of a test object, they pass the object to all the decision trees, returning the majority voted class.

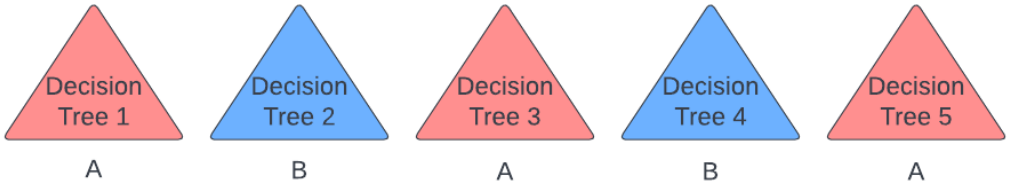

Figure 7: Depiction of a random forest of 5 trees. Here, the output would be class A.

The idea behind the random forest is rooted in the law of large numbers. That is, as the sample size grows, the expected mean tends towards the average. In the case of a single decision tree, if we assume it is more likely than not that its output is correct, then in the case of many decision trees, we should expect the total output to be correct. This is especially important for decision trees, which are prone to overfitting if a single tree considers all the training data.

4.2 Bootstrap aggregating

Bootstrap aggregating (bagging) is a procedure used in less stable classification algorithms like random forests. The first step is bootstrapping. Bootstrapping generates subsets of the training data by picking random objects with replacement (which allows duplicate objects in a bootstrap data set).

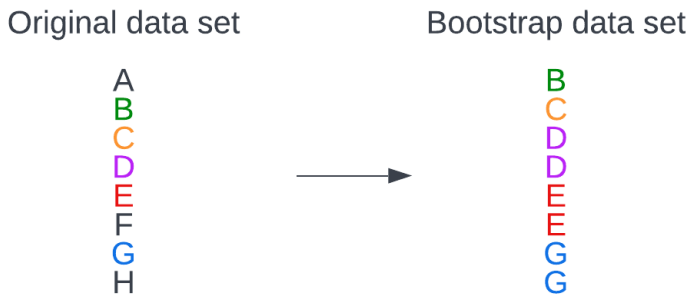

Figure 8: An illustration of how a bootstrap data set is created from the original.

Once we have accumulated a large number of trees trained on these different data sets, the next step is aggregation. The test data is fed to all of the trees and the prediction is decided by a majority vote.

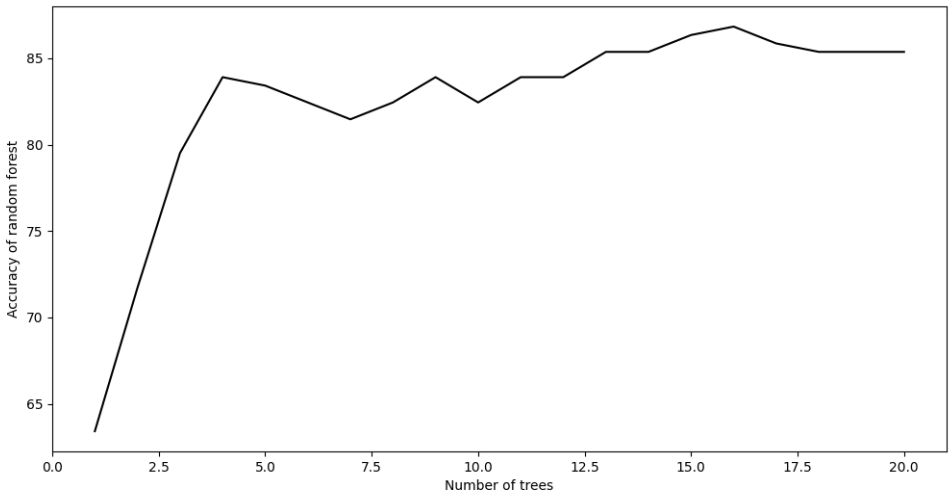

Figure 9: The accuracy of my own random forest algorithm on the Sonar, Mines vs. Rocks data set using Bootstrap data sets of size 10% of the original data set, with the number of trees varying from 1 to 20.

As shown in Figure 9, bagging and random forests have the potential to greatly improve upon the accuracy of decision trees, even when using sub-optimal bootstrap sizes like 10%.

4.3 Conclusion

4.3.1 Benefits

Random forests are more robust than decision trees, and manage to make predictions about objects in $\mathbf{O}(k * d)$ time, where k is the number of decision trees used, and d is the depth of the trees, which is best case logarithmic and worst case linear in the size of the bootstrap data sets. Random forests benefit from the advantages of decision trees that were discussed before, but are more suited to allow pruning, which reduces the risk of overfitting.

4.3.2 Drawbacks

Training decision trees is already slow, and a random forest requires training many decision trees, so the training phase is very slow. Additionally, random forests have multiple hyperparameters that must be tuned for optimal performance, like maximum depth, minimum size of node, and bootstrap size. Tuning these is mostly trial and error, and they come with a trade between speed and accuracy.

5 Naive Bayes

Naive Bayes classifiers return a probability that a test object is a class based on Bayes theorem.

5.1 How it works

Bayes theorem states that the probability $P(A|B)$ of A happening, given B has happened is given by the formula

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

The idea behind Naive Bayes is that this can be extended so if we're given an object $\vec{x}$ with f features, we can say that the probability $P(\vec{x} \in C|\vec{x})$ that $\vec{x}$ belongs to class C is given by the formula

$$P(\vec{x} \in C|\vec{x}) = \frac{P(\vec{x}|\vec{x} \in C)P(C)}{P(\vec{x})}$$

where $P(\vec{x}|\vec{x} \in C)$ is the probability of seeing features $\vec{x}$ given the object is of class C, $P(C)$ is the probability of an occurrence of class C, and $P(\vec{x})$ is the probability of seeing observations $\vec{x}$. Now we make the 'naive' assumption that the features are conditionally independent. In that case, if we say $\vec{x} = (x_1, x_2, ..., x_f)$, and we denote $y = \vec{x} \in C$, then $P(\vec{x}|y) = P(x_1|y)P(x_2|y)...P(x_f|y)$. Similarly, $P(\vec{x}) = P(x_1)P(x_2)...P(x_f)$. Thus, the formula above reduces to

$$P(y|\vec{x}) = \frac{P(x_1|y)P(x_2|y)...P(x_f|y)P(y)}{P(x_1)P(x_2)...P(x_f)}.$$

You can see that the denominator does not depend on C, so we can remove the denominator and simply consider a proportionality in order to find the best C for our observations. So we obtain

$$\begin{aligned} P(y|\vec{x}) &\propto P(y)P(x_1|y)P(x_2|y)...P(x_f|y) \\ &= P(C)\prod_{i=1}^{f} P(x_i|y). \end{aligned}$$

Thus, to find the class that $\vec{x}$ is most likely to be, we simply need to find

$$argmax_y \left(P(y)\prod_{i=1}^{f} P(x_i|y) \right)$$

which requires us to calculate $P(x_i|y)$ for all pairs i, y, where $i \in [1, f], y \in [C_1, C_{\text{total}}]$ and C_j denotes the j-th unique class in the data set. Finding $P(y)$ is quite simple, it's just the number of occurrences of class y in the training set divided by the size of the training set. Finding $P(x_i|y)$ is harder, because it's not guaranteed that we've seen the value x_i for feature i anywhere in the training set. Here, we make another assumption, that the data follows a Gaussian distribution, that is, it follows the formula

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

where μ is the mean and σ is the standard deviation of the data. Now, we can preprocess the training set so that for each class c, for each feature i, we store $\mu_{c,i}$ and $\sigma_{c,i}$. Then, we can find $P(x_i|y)$ by plugging $\mu_{y,i}$, $\sigma_{y,i}$, and x_i into the formula for the Gaussian distribution.

5.2 Conclusion

5.2.1 Benefits

Naive Bayes classifiers are able to handle binary and multiclass classifications, as well as take categorical and numerical features. It's also intuitive and works off a closed-form solution so it's easy to see why the algorithm is performing the way it is.

5.2.2 Drawbacks

In the training phase, Naive Bayes must calculate $\mu_{c,i}$ and $\sigma_{c,i}$ for C classes and f features. Calculating these values can be done in $\mathbf{O}(nf)$, where n is the number of training objects. Thus, like in answering queries, Naive Bayes suffers from the curse of dimensionality.

When predicting the class of a new object, for all classes y, Naive Bayes must find $P(x_i|y)$ for each feature i. This means the total time of answering a query is $\mathbf{O}(Cf)$. Thus, in predictions, Naive Bayes also suffers from the curse of dimensionality.

Naive Bayes also has two assumptions, one is that the features are conditionally independent, and the other is that the features are normally distributed, which are not always the case in real data sets.

6 Logistic regression

Logistic regression is a common classification algorithm used to predict binary data by inputting the hypothesis function into a logistic function.

6.1 How it works

Consider the problem of trying to determine if an email is spam or not based on the number of spelling mistakes in the email. A potentially sensible approach is to draw a line of best fit $h(x)$ through the points, set 'yes' $= 1$ and 'no' $= 0$, and if $h(x_{\text{new}}) \geq 0.5$, we consider the new email spam. Otherwise we classify it as a legitimate email.

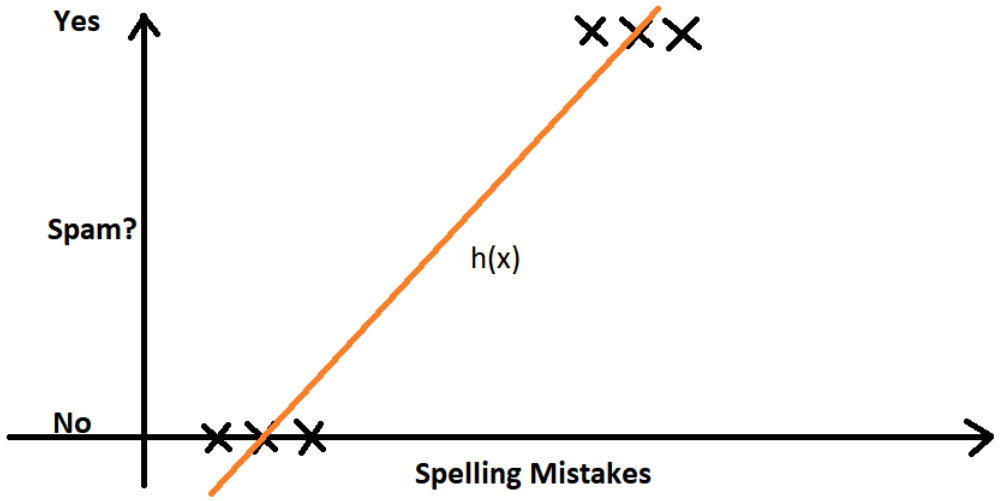

Figure 10: In this graph, it seems like our hypothesis function $h(x)$ classifies well between spam and legitimate emails.

Now consider a spam email in the training set with many more spelling mistakes. $h(x)$ would naturally shift to best fit the new points.

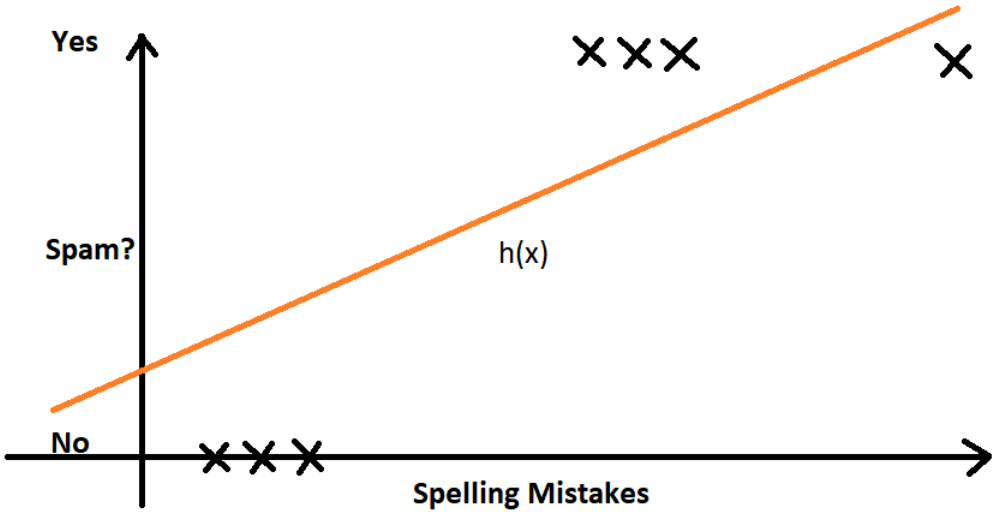

Figure 11: This graph has an additional spam email with many more spelling mistakes, invalidating linear regression as a classifier.

Of course, if we had a super spam email, our line would change again, and the line between spam and legitimate email wouldn't be at $h(x) = 0.5$. Certainly, it's a problem if our classification algorithm doesn't have a set threshold with which to classify. This should make it clear why linear regression isn't adequate for this classification problem.

In logistic regression, the idea is to pass the output of the hypothesis function $h(x)$ into the Sigmoid function $S(z) = \frac{1}{1+e^{-z}}$, which maps values to between 0 and 1.

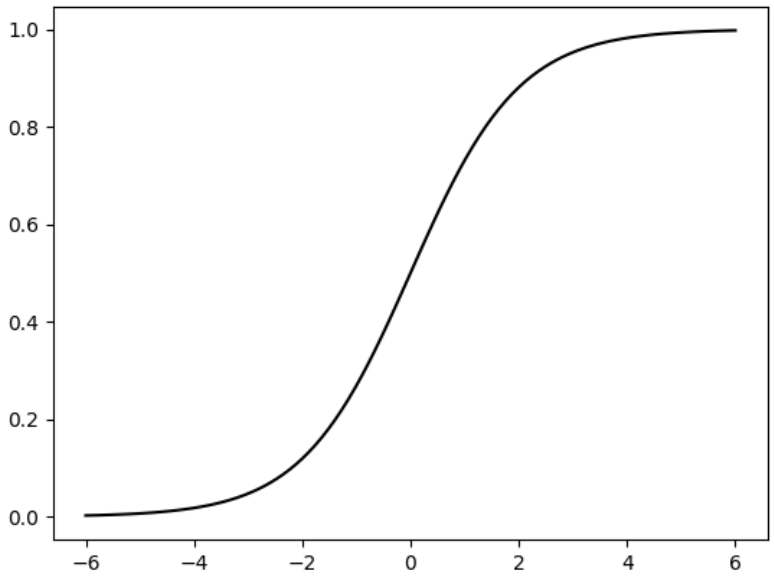

Figure 12: The Sigmoid function

As a result, we now define the function $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$, where θ_i denotes the weight of the i-th feature in $h(x)$. In order to optimise the weights θ of this new hypothesis function, we want a cost function $J(\theta)$ such that if $h_\theta(x)$ tends away from y, the error is maximised, and if $h_\theta(x)$ tends towards y, the error is minimised. If $y = 1$, $J(\theta) = -\log(h_\theta(x))$ fits the requirement, and if $y = 0$, $J(\theta) = -\log(1 - h_\theta(x))$ fits the requirement. Thus, we can combine these two cases into the cost formula

$$J(\theta) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x)).$$

As $J(\theta)$ is inversely proportional to the accuracy of $h_\theta(x)$, we want to minimise the value of $j(\theta)$. This can be done via an approximation method called gradient-descent, as this function can be differentiated with respect to the weight of feature j, θ_j. It can be proven that $\frac{\partial(J(\theta))}{\partial(\theta)}$ follows the formula

$$\frac{\partial(J(\theta))}{\partial(\theta)} = \frac{1}{f} x^T \left(h_\theta(x) - y\right).$$

To start the proof, we consider the Sigmoid function $S(x) = \frac{1}{1+e^{-x}}$

$$\begin{aligned}
\frac{d(S(x))}{dx} &= \frac{0 * (1 + e^{-x}) - (1) * ((-1) * e^{-x})}{(1 + e^{-x})^2} \\
&= \frac{e^{-x}}{(1 + e^{-x})} \\
&= \frac{1 + e^{-x}}{(1 + e^{-x})} - \frac{1}{(1 + e^{-x})^2} \\
&= \frac{1}{1 + e^{-x}} * \left(1 - \frac{1}{1 + e^{-x}}\right) \\
&= (1 - S(x))S(x)
\end{aligned}$$

Using the chain rule

$$\begin{aligned}
\frac{\partial(J(\theta))}{\partial(\theta)} = -\frac{1}{f} \sum_{i=1}^{f} &\left(y^{(i)} \frac{1}{h_\theta(x^{(i)})} \frac{\partial(h_\theta(x^{(i)}))}{\partial(\theta_j)}\right) \\
+ \sum_{i=1}^{f} &\left((1 - y^{(i)}) \frac{1}{(1 - h_\theta(x^{(i)}))} \frac{\partial(1 - h_\theta(x^{(i)}))}{\partial(\theta_j)}\right)
\end{aligned}$$

Using the differentiation of the Sigmoid function

$$
\begin{aligned}
&= -\frac{1}{f}\sum_{i=1}^{f}\left(y^{(i)}\frac{1}{h_\theta(x^{(i)})}S(z)(1-S(z))\frac{\partial(\theta_x^T)}{\partial(\theta_j)}\right)\\
&+\sum_{i=1}^{f}\left((1-y^{(i)}\frac{1}{(1-h_\theta(x^{(i)}))}(-S(z))(1-S(z))\frac{\partial(\theta_x^T)}{\partial(\theta_j)}\right)\\
&= -\frac{1}{f}\sum_{i=1}^{f}\left(y^{(i)}\frac{1}{h_\theta(x^{(i)})}h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x_j^i\right)\\
&+\sum_{i=1}^{f}\left((1-y^{(i)})\frac{1}{(1-h_\theta(x^{(i)}))}(-h_\theta(x^{(i)}))(1-h_\theta(x^{(i)}))x_j^i\right)
\end{aligned}
$$

Simplifying

$$
\begin{aligned}
&= -\frac{1}{f}\sum_{i=1}^{f}\left(y^{(i)}(1-h_\theta(x^{(i)}))x_j^i-(1-y^{(i)})h_\theta(x^{(i)})x_j^i\right)\\
&= -\frac{1}{f}\sum_{i=1}^{f}x_j^i\left(y^{(i)}-y^{(i)}h_\theta(x^{(i)})-h_\theta(x^{(i)})+y^{(i)}h_\theta(x^{(i)})\right)\\
&= -\frac{1}{f}\sum_{i=1}^{f}x_j^i\left(y^{(i)}-h_\theta(x^{(i)})\right)
\end{aligned}
$$

Finally, converting into a matrix as required

$$\frac{\partial(J(\theta))}{\partial(\theta)} = \frac{1}{f}x^T\left(h_\theta(x)-y\right).$$

This can be computed by plugging values in, and hence the cost function can be minimised via gradient descent. (The math in this section was heavily assisted by Thavanani S. 2019)

6.2 Gradient descent

Gradient descent is an iterative approximation algorithm used to find minima in functions. Given the gradient function of a convex, it's guaranteed that following right if the gradient at a point is negative, and following left if the gradient at a point is positive will find a minima. Gradient descent takes in two hyperparameters, the learning rate and the number of epochs or iterations. When we decide to move a point to the left or right, we move it proportional to the learning rate. Additionally, the number of times we check the gradient and move a point is equal to the epoch number. These require tuning, as too low of a learning rate, and no changes will be made. Too high of a learning rate, and the point will bounce past the minima, possibly increasing its y-value. Similarly, a high epoch number serves to enable a precise learning rate, but it slows down the training

phase of the algorithm. Of course, too low an epoch number, and only minuscule changes will be made to the weights.

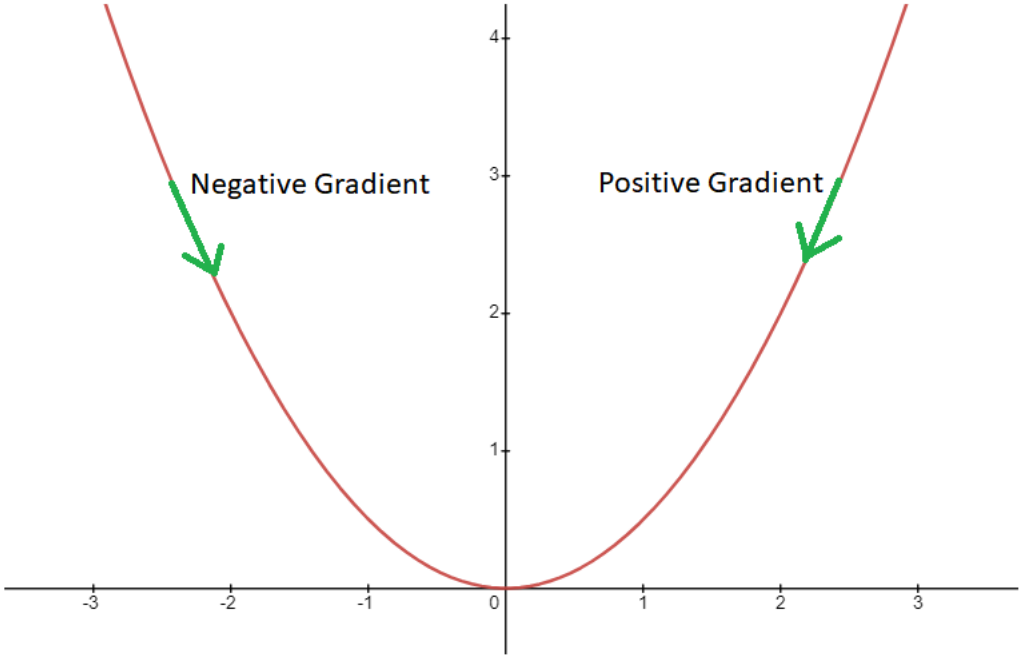

Figure 13: Two examples of gradient descent iterations on $f(x) = \frac{x^2}{2}$. The size of the green arrows are proportional to the learning rate.

6.3 Conclusion

6.3.1 Benefits

Although tricky to derive, logistic regression is mostly mathematical, and so is relatively easy to implement. The time complexity of making a prediction is $\mathbf{O}(f)$, where f is the number of features in the set, which is quite good. Finally, logistic regression has a harder time overfitting, although it is possible in high dimensionality sets.

6.3.2 Drawbacks

One of the main drawbacks of logistic regression is that it can only predict binary functions. Additionally, logistic regression relies on an assumption of linearity between features and classes. The time complexity of predictions is fairly good, but predictions will still be slower on high-dimensionality data sets than other algorithms like decision tree classifiers.

7 Multiclass logistic regression

Multiclass (or multinomial) logistic regression is an extended version of logistic regression that predicts the probability of an object belonging to all classes.

7.1 How it works

Now that we are trying to predict the probability of $\vec{x}$ being part of every single class, our $f \times 1$ weight vector $\mathbf{W}$ must become an $f \times C$ matrix, where the weight of a feature i for a certain class j is given by $\mathbf{W_{i,j}}$. In order to find the proclivity of an object towards a certain class, we multiply it by the weight matrix such that we get $\vec{Z} = \mathbf{XW}$. $\vec{Z}$ is now a $1 \times C$ vector, which is a list of numbers showing how much our model thinks object i is part of class j. In order to get the probability vector P_i for object i, we can run the softmax formula $\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$, which turns vectors into stochastic vectors, that is $\sum_{x \in \vec{v}} x = 1$. Once we have the probabilities that object i is of class j, we take the class with the largest probability and that is our prediction.

Now we need to consider how to train this model based off predictions on the training set. A reasonable loss function for $\mathbf{W}$ is

$$\begin{aligned} L(\mathbf{W}) &= -\frac{1}{n} P(Y|X, \mathbf{W}) \\ &= \frac{1}{n} \sum_{i=1}^{n} \left(X_i \mathbf{W}_{k=Y_i} + \log \sum_{k=0}^{C} e^{-X_i \mathbf{W}_k} \right) \\ &= \frac{1}{n} \sum_{i=1}^{n} X_i \left(\mathbf{W}_{k=Y_i} + \sum_{i=1}^{n} \log \sum_{k=0}^{C} e^{-X_i \mathbf{W}_k} \right). \end{aligned}$$

Adding $l2$ regularisation, the combined function $F(\mathbf{W})$ is given by

$$F(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^{n} \left(X_i \mathbf{W}_{k=Y_i} + \log \sum_{k=0}^{C} e^{-X_i \mathbf{W}_k} \right) + \mu ||\mathbf{W}||^2.$$

As such, the gradient can be calculated in this manner,

$$\begin{aligned} \nabla_{\mathbf{W}_k} F(\mathbf{W}_k) &= \frac{1}{n} \sum_{i=1}^{n} \left(X_i^T I_{[Y_i=k]} - X_i^T \frac{e^{-X_i \mathbf{W}_k}}{\sum_{k=0}^{C} e^{-X_i \mathbf{W}_k}} \right) + 2\mu \mathbf{W} \\ &= \frac{1}{n} \left(\sum_{i=1}^{n} X_i^T I_{Y_i=k} - \sum_{i=1}^{n} X_i^T P_i \right) + 2\mu \mathbf{W} \\ &= \frac{1}{n} \left(X^T Y_{\text{onehot}} - X^T P \right) + 2\mu \mathbf{W} \\ &= \frac{1}{n} \left(X^T \left(Y_{\text{onehot}} - P \right) \right) + 2\mu \mathbf{W} \end{aligned}$$

Now that we have the formula for the gradient, we can use gradient-descent to minimise the loss of the algorithm. (The math in this section was assisted heavily by Yang S. 2021)

7.2 Conclusion

7.2.1 Benefits

Multiclass logistic regression gives us the benefits of logistic regression with the ability to predict objects of multiclass data sets. Additionally, once the math is done, implementation of multiclass regression is fairly simple using python libraries.

7.2.2 Drawbacks

Like logistic regression, multiclass regression is prone to overfitting on high dimensionality data. The softmax function is prone to blow up numbers to enormous amounts because of the exponential behaviour of the formula.

8 Comparison of algorithms

I implemented all of these algorithms and tested them against 5 different test sets, Iris, Glass, Sonar, Haberman, and Fertility.

Average accuracy of different classification algorithms on different data sets over 5 trials (%)						
Algorithm	Data Set					Average
	Iris	Glass	Sonar	Haberman	Fertility	
KNN, K=3	95.79	99.63	83.08	66.15	87.2	86.37
KNN, K=6	99.47	98.15	74.62	72.4	86.4	86.21
KNN, K=9	97.37	95.93	67.31	75.58	87.2	84.68
Random Forest, 1 Tree	95.33	97.14	86.83	80.33	91.0	90.13
Random Forest, 5 Trees	98.0	98.10	99.51	85.25	94.0	94.97
Random Forest, 10 Trees	98.67	99.05	99.51	87.21	96.0	96.09
Naïve Bayes	96.84	85.56	67.31	76.10	80.8	81.32
Multinomial Logistic Regression	85.79	72.59	53.46	72.47	86.4	74.14
Logistic Regression	N/A	N/A	78.08	66.49	84.8	76.46
Average	95.91	93.27	78.86	75.78	88.2	85.60

Figure 14: The average accuracies of the different classification algorithms I implemented. The figure in the bottom right corresponds to the average of the average accuracy of the algorithms.

The Random forest algorithm outperformed all the other algorithms, although in some test sets KNN had a marginally higher accuracy than it. The regression algorithms had mediocre peformance, which is potentially indicative that I needed to insert things like bias factors into my multinomial regression. It's also worth noting that the Iris and Glass datasets are multiclass data sets, so Logistic Regression wasn't a valid classifier to run in those cases.

9 Bibliography

1. S. Thavanani 2019, The Derivative of Cost Function for Logistic Regression, accessed 4 December 2022, https://medium.com/analytics-vidhya/derivative-of-log-loss-function-for-logistic-regression-9b832f025c2d.

2. S. Yang 2021, Multiclass logistic regression from scratch, accessed 4 December 2022, https://towardsdatascience.com/multiclass-logistic-regression-from-scratch-9cc0007da372.

3. R.A. Fisher 1936, Iris Data Set, accessed 4 December 2022, https://archive.ics.uci.edu/ml/datasets/iris.

4. B. German 1987, Glass Identification Data Set, accessed 4 December 2022, https://archive.ics.uci.edu/ml/datasets/glass+identification.

5. Terry Sejnowski 1988, Connectionist Bench (Sonar, Mines vs. Rocks) Data Set, accessed 4 December 2022, http://archive.ics.uci.edu/ml/datasets/connectionist+bench+(sonar,+mines+vs.+rocks).

6. S. J. Haberman 1976, Haberman's Survival Data Set, accessed 4 December 2022, https://archive.ics.uci.edu/ml/datasets/haberman's+survival.

7. David Gil 2013, Fertility Data Set, accessed 4 December 2022, https://archive.ics.uci.edu/ml/datasets/Fertility.