**Description of the selected forecasting problem:** Using data on fetal health the goal is to forecast the health of the given fetus into three categories; normal, suspect, or pathological. This is a classic classification problem where data is used to group the output into types as opposed to regression where a numerical value(s) is/are calculated.

**Description of the available data:** There are 21 attributes which can be used to forecast the output which is the 22nd field. There are 2126 samples taken in this dataset providing a healthy sample size for the classification algorithms we are looking to work with today. The 21 attributes are all numerical, even if they describe categorical data meaning that the dataset was already vectorized. This dataset is not normalized, the methods we use will have to be able to deal with unnormalized data or we will have to normalize the dataset if necessary. There is no missing data in the dataset. Every attribute and typically 80% of the samples will be used to train the model (train\_X & train\_Y). The model will be tested using typically 20% of the samples and every attribute except the output (text\_X) and compared to the true output (test\_Y). This 80/20 split will apply to both cross-validation and hold-out as shown later on in this report.

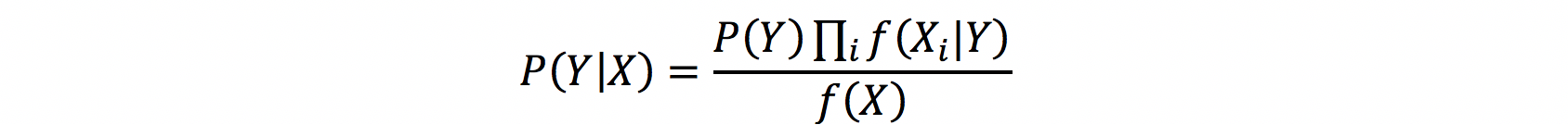
******Algorithm overview – Naïve Bayes:** Naïve Bayes for Classification relies on Bayes Theorem, shown in fig1, but makes one very naïve assumption (thus it’s name), the assumption is that all features are independent from one another, this is referred to as conditional independence thus allowing Bayes Theorem to be expressed as shown in fig2 where f is the probability density function. There are four options for probability density functions including gaussian, Bernoulli, complement and normal. In this project a gaussian distribution was selected. Naïve Bayes is computationally very inexpensive and can handle high-dimensional data more easily than other algorithms can, it especially performs relatively well on datasets with smaller sample sizes. Naïve Bayes does not compare well to other algorithms in accuracy when the sample size is large.

Figure 2

Figure 1

[[Naive Bayes Explained. Naive Bayes is a probabilistic… | by Zixuan Zhang | Towards Data Science](https://towardsdatascience.com/naive-bayes-explained-9d2b96f4a9c0)]

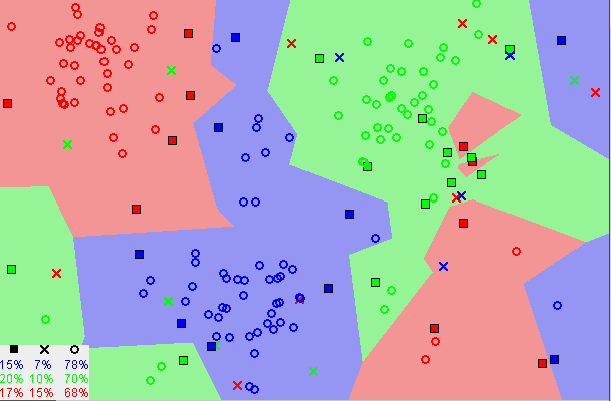
**Algorithm overview – K Nearest Neighbour:** K Nearest Neighbour attempts to group samples together by calculating the Euclidean distance between them and assumes that samples of the same type should clump up together as shown in fig3. Using Euclidean distance, the test set has each sample classified within one of the pre-drawn boundaries. This algorithm requires little to no tuning, only the k value is tuned, and is theoretically simple and very easy to apply. This algorithm struggles in terms of computational complexity when the number of attributes per sample increases.

Figure 3

[[Machine Learning Basics with the K-Nearest Neighbors Algorithm | by Onel Harrison | Towards Data Science](https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761)]

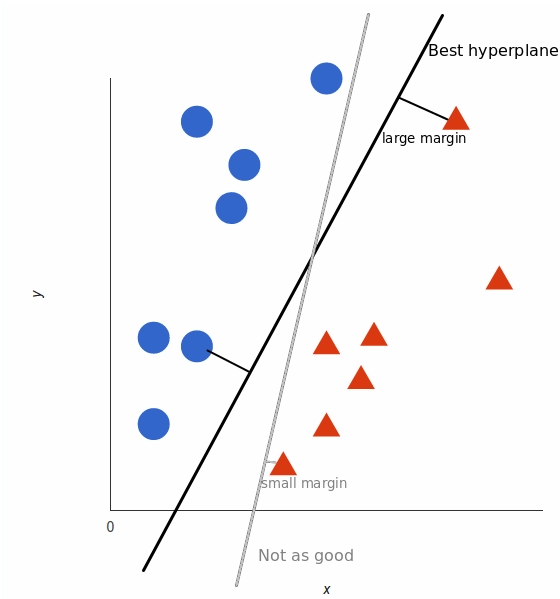
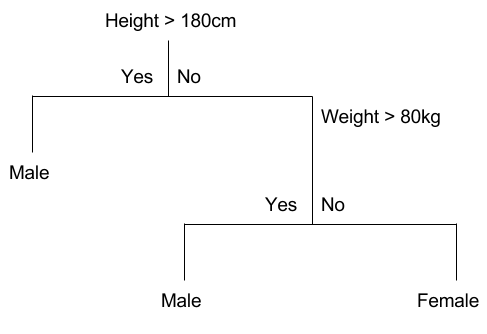
**Algorithm overview – Support Vector Machine:** The Support Vector Machine algorithm attempts to find the hyperplane which split up its training samples with as much distance between the plane and the training points as possible, essentially maximizing the margin as shown in fig4. Once the hyperplanes are selected and tuned the test data is categorized based on where it is in relation to the hyperplane. This algorithm is very good at handling data with a high number of dimensions but struggle with large sample sizes and overlapping classes which could create very complex hyperplanes, additionally tuning parameters is crucial for this algorithm and it’s very sensitive to these parameters.

Figure 4

[[Support Vector Machines (SVM) Algorithm Explained (monkeylearn.com)](https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/)]

[[Support Vector Machine Algorithm with Example – Data Science Revolution (wordpress.com)](https://heyitskoushik.wordpress.com/2017/12/25/support-vector-machine-algorithm-with-example/#:~:text=Pros%20and%20Cons%20associated%20with%20SVM%201%20Pros%3A,required%20training%20time%20is%20higher%20More%20items...%20)]

**Algorithm overview – Decision Tree:** A decision tree consists of nodes, branches and leaves. A node is simply any dividing condition, the branches connect two nodes and correspond to the answer from the previous node and the entry into the next node, the leaves predict the actual outcome and are terminal points. This algorithm essentially divides the dataset into smaller subsets recursively until the training set is neatly divided into the appropriate categories as shown in fig5. This algorithm is not resource intensive and has no tuning parameters making it quick to setup and to use. However, this algorithm has a strong bias towards overfitting and can easily fall into that trap.

Figure 5

[[Decision Tree Classification. A Decision Tree is a simple… | by Afroz Chakure | The Startup | Medium](https://medium.com/swlh/decision-tree-classification-de64fc4d5aac)]

**Algorithm overview – Random Forest:** The Random Forest algorithm builds on the idea of the Decision Tree algorithm and provides many trees for an input sample to be applied to output of each of these trees’ “votes” on the category and the majority vote decides the categorical output. This method provides a more accurate result than a Decision Tree algorithm and can efficiently handle large amounts of data without using a large amount of computation power and is not prone to overfitting (unlike the Decision Tree algorithm).

[ [Random forests - classification description (berkeley.edu)](https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm)]

**Algorithm Application & Evaluation:** The basic application and evaluation structure was identical across all five of the algorithms that were used in Assignment 1, in fact the code was identical between large sections of each algorithm.

Graphical user interface, text, application

Description automatically generated1. The first step is to import the required libraries to run the necessary algorithm. This snippet comes from the Decision Tree algorithm and thus we are importing the DecisionTreeClassifier from sklearn.tree, in all of the algorithms this is pretty much the only line we changed in terms of imports. The only exception was for SVM where we also imported a library that could be useful for tuning the parameters of the SVM.

Text

Description automatically generated2. The second step was to read the data into a dataframe (pd), the dataset was stored in a tab delimited file called ‘fetal\_health.txt’. A good practice is to print what is called the “head” this is a built in function of the dataframe, which prints feature names and the first five samples - ensure the data was read as you expected. The second part of this step is to split the data into useable chunks. Split the dataset into X and Y values and then use the train\_test\_split method which was imported to split the data into train\_X, test\_X, train\_Y, test\_Y. Note that the test size is specified as 20% and this value was used in all of the algorithms.

3. The third step is the only step with significantly different implementations between algorithms, the example shown is from the Decision Tree algorithm. SVM and KNN need parameters to be specified whereas Decision Tree, Random Forest and Naïve Bayes do not need parameters to be defined by the user.

Graphical user interface, text, application

Description automatically generated4. The fourth step is uniform across all algorithms. The model you decided on in the previous step must be fitted to the training data set and then the test input data can be used on the newly trained model to make a prediction and stored in y\_pred.

**In this assignment I chose to print the results of hold-out validation (step 5) and cross-validation (step 6), even though it was unnecesary.**

Graphical user interface, text

Description automatically generated5. This step displays how to print the accuracy output of hold-out validation. Using the metrics import there is an accuracy\_score method which compare the predicted output versus the actual output and measures how accurate the predicted output was, then it prints it out.

Text

Description automatically generated6. This step sets up cross validation and then prints out the results. In all selected algorithms 5 fold cross-validation has been selected (cv). The model is selected to be a DecisionTree and then fed to the estimator, the entire X and Y data set were used in cross-validation. This isn’t an example of training on the test data because cross\_val\_score trains 5 times independantly using 4/5’s of the data without peeking and then tests with the remaining 1/5 of the data. The mean accuracy is printed out afterwards.

**Comparison of Results and Conclusions:**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Hold-out** | **Cross-validation** |
| Naïve Bayes | 0.8051 | 0.7973 |
| KNN | 0.8967 | 0.8283 |
| SVM | 0.9131 | 0.8010 |
| Decision Tree | 0.9202 | 0.8241 |
| Random Forest | 0.9531 | 0.8608 |

For all models except Naïve Bayes there was a significant difference between hold-out accuracy and cross-validation accuracy. When I looked this curious phenomenon up on Stack Exchange, it was noted that when there is a significant difference between cross-validation and hold-out that is typically indicative of overfitting. This explanation made sense because these four had been noted in some of my literature review as having trouble with overfitting, where Naïve Bayes didn’t have such a claim in my literature review. The SVM algorithm was most affected and this could be in part due to the extremely non-linear nature of the hyperplane which can permit more overfitting. Another interesting point is how incredibly accurate the Random Forest algorithm is with no real competitor, by building on the Decision Tree algorithm and voting on the results - the accuracy was improved and potential overfitting was slightly reduced as shown by the lower holdout-cross validation difference.

[[machine learning - Why does my model consistently perform worse in cross-validation? - Cross Validated (stackexchange.com)](https://stats.stackexchange.com/questions/440256/why-does-my-model-consistently-perform-worse-in-cross-validation)]

The dataset was small enough (~ 2000 samples) that there was no noticeable delays in receiving results suggesting that the programs were not computationally intensive with such a small dataset. The algorithms are listed below in order of accuracy (cross-validation).

**Random Forest, KNN, Decision Tree, SVM, Naïve Bayes**