Breast cancer predictive models

JW du Toit
Computer Science Department
Stellenbosch University
Stellenbosch, South Africa
22808892@sun.ac.za
22808892

***Abstract*—Breast cancer is the most common type of cancer and is a serious threat to the lives of millions of women. Thus, there is a need to effectively and accurately predict whether breast cancer tumours are malignant or benign in early diagnosis. To achieve this we can use certain descriptive features of tumours and train a machine learning model to predict the nature of the tumour. The k-nearest neighbours (k-NN) and classification tree algorithms are popular machine learning methods used for various predictive tasks. This project performs a comparative study of these two algorithms for the task of predicting breast cancer tumours as malignant or benign. These algorithms are compared with regard to their predictive performance based on a set of descriptive features related to breast cancer tumours. Furthermore, the optimal configuration for each algorithm is determined for the associated task. Both of these algorithms perform well, with accuracies of over 90% obtained on a test set. We observe that the k-NN algorithm outperforms the classification tree with an accuracy of 93.30% for the former compared to 91.62% for the latter. Thus, the k-NN algorithm is the preferred algorithm for this predictive task.**

I. Introduction

Breast cancer is a very common type of cancer and is a serious concern for women's health. Thus, in order to assist in breast cancer diagnosis there is a need to effectively and accurately predict whether breast cancer tumours are malignant or benign. To achieve this we can use certain descriptive features of the tumour to train machine learning models to predict the nature of the tumour. The k-nearest neighbours (k-NN) [1] algorithm and classification trees [2] are two well-known machine learning algorithms that have been applied to solve many different predictive tasks. This project uses these algorithms to build models for the prediction of breast cancer tumours.

The objective of this project is to compare the performance of these two algorithms with regard to the task of predicting breast cancer tumours as malignant or benign. Furthermore, this project aims to determine the optimal configuration of each algorithm for the associated task.

The two algorithms are implemented, trained and their hyperparameters are tuned to obtain the optimal configuration for the given task. The accuracies of these algorithms are then compared on a test set to see which algorithm performs best for the prediction of breast cancer tumours.

The main findings from this project are the optimal configurations for the two algorithms and the accuracies obtained by the algorithms on a test set. We observe that the k-NN algorithm achieves an accuracy of 93.30% and the classification tree an accuracy of 91.62%. Thus, the k-NN algorithm outperformed the classification tree and it is the preferred algorithm for this predictive task.

The rest of the report proceeds as follows: Section II provides a high-level discussion on the k-NN algorithm and classification trees. Section III discusses the approach used to solve data quality issues and the implementation of the two algorithms used. Section IV describes the process that was followed to obtain the empirical results. Section V presents and discusses the results obtained from the experiments and Section VI draws a conclusion for the report.

II. Background

This section provides a high-level discussion on the k-NN algorithm and classification trees.

A. k-nearest neighbours algorithm

The k-NN algorithm [1] is a simple supervised, similarity-based machine learning algorithm. The algorithm stores all of the training instances during the training phase. To predict a new query instance the distance between the query instance and each instance in the training set is computed using a distance measure based on the features of the instances. The algorithm sorts all of the distances and selects the k training instances that are closest to the query instance. These instances are referred to as the k nearest neighbours. The number of neighbours belonging to each possible class is counted. The class with the most instances from the set of neighbours is then predicted as the class for the query instance. Ties between classes are broken by randomly choosing one of the classes. The pseudo code for the k-NN algorithm is provided by Algorithm 1.

Algorithm 1 k nearest neighbours algorithm

```
Let d be an empty array
for all x_i ∈ D do
    d = DISTANCE(x_i, x)
    add d to d
end for
SORT(d)
Let S = the first k instances in d
return the majority class in S, ties are broken randomly
```

B. Classification trees

Classification trees [2] are supervised machine learning algorithms that use a tree structure to form a predictive model. The non-leaf nodes of the tree represent decisions on descriptive features and the leaf nodes represent the target class predictions. Classification trees use information theory [3] concepts such as entropy and information gain to induce a tree structure that is used for predictive tasks.

Classification trees recursively split the training set into subsets such that they have a lower level of heterogeneity, until all subsets are homogeneous. The training set is split based on conditions for a certain input feature such that all instances in a newly formed subset satisfies a condition that the instances in the other subsets do not satisfy. For example, for a continuous valued feature a threshold is chosen and the dataset is split such that all instances that are less than or equal to the threshold for that feature will be in the same subset and all instances that are greater than that threshold will form a different subset. The algorithm uses entropy [3] as a measure of homogeneity for a dataset with respect to class labels. If all instances in a dataset have the same class, *i.e.* they are homogeneous, entropy is 0. The entropy of a dataset D is given by

$$H(D) = -\sum_{m=1}^{M} p(y_m) log_2 p(y_m) \tag{1}$$

where M is the number of classes and $p(y_m)$ is the probability of a class y_m occuring in D and is given by

$$p(y_m) = \frac{\text{freq}(y_m, D)}{|D|} \tag{2}$$

where freq(y_m, D) is the number of times that class y_m occurs in D.

Suppose D is split on the input feature x with given O outcomes. The entropy obtained by the split is given by

$$H_x(D) = \sum_{o=1}^{O} p_o H(D_o) \tag{3}$$

where p_o is the probability of outcome o and is given by

$$p_o = \frac{|D_o|}{|D|} \tag{4}$$

and D_o is the subset containing all instances associated with outcome o.

The objective of the classification tree induction algorithm is to select the condition on an input feature that will maximise the reduction in heterogeneity of the dataset. Thus, the algorithm uses a concept called information gain [3] with the goal of maximising the value of the information gain in order to increase homogeneity. The information gain for the dataset D being partitioned on the feature x is given by

$$\text{gain}(x) = H(D) - H_x(D) \tag{5}$$

where $H(D)$ is the entropy of dataset D and $H_x(D)$ is the entropy obtained by splitting the dataset D on the input feature x.

The algorithm follows the following steps to induce the classification tree:

1) Compute entropy of the current dataset, $H(D)$.
2) For each input variable calculate the entropy, $H_x(D)$.
3) Calculate the information gain for D being split on x.
4) Select the input variable that results in the highest information gain.
5) Split D into O subsets, where O is the number of outcomes for x.
6) Repeat the process on each of the subsets until each subset is homogeneous.

The pseudo code for the classification tree induction algorithm is provided in Algorithm 2 where D is the dataset.

Algorithm 2 InduceTree(D)

if $|D| = 0$ **then**
  Return leaf with default class
end if
if $|D| > 0$ and $\forall \mathbf{x} \in |D|$ the class is the same, *i.e.* y_m **then**
  Return leaf with class label y_m, containing D
end if
Select a test based on a single input variable
Split D into $D_1, D_2, ..., D_O$ where O is the number of outcomes
for o = 1 to O **do**
  InduceTree(D_o)
end for

To predict a query instance the conditions in nodes of the tree are simply checked, starting at the root node, and the nodes are followed based on these conditions until a leaf node is reached. The class associated with the leaf node that is reached is then assigned as the predicted class for the query instance.

III. Methodology

This section discusses the data quality issues for the given datasets and how these issues were addressed. Furthermore, the implementations of the k-NN and classification tree algorithms are discussed.

A. Data exploration and management

The datasets used for this project include a training dataset with 390 instances and a test dataset with 179 instances. For each instance the patient ID, patient gender, whether the tumour is benign or malignant and 31 descriptive features about the tumour such as the radius mean and texture mean are provided. We perform an exploratory data analysis to observe different characteristics of the datasets and check for potential data quality issues. We observe various data quality issues such as irrelevant features, outliers, missing values, incorrect data entries, and imbalanced class distributions.

The diagnosis for the nature of the tumour is encoded from 'M' to 1 for malignant tumours and from 'B' to a 0 for benign tumours to work more easily within the implemented

algorithms. Furthermore, irrelevant features such as the patient ID and gender are removed.

We observe an incorrect data entry for the 'texture mean' feature as one of the values was negative. This negative value is changed to the median for that feature. Furthermore, we observe one clear outlier value for the 'smoothness mean' feature which is assumed to be an incorrectly entered value as it is not a realistic value. This value is also changed to the median of that associated feature. The algorithms used for this project are robust to outliers and thus it is not required to remove outliers that are not clearly incorrect entries.

We observe that 94 instances in the training set contain a missing value for at least one feature and thus it is not feasible to remove all of these instances. Furthermore, we observe that the number of missing values for each feature in the training set is at most 6. Thus, we decided to impute these values as there are so few missing for each feature. For both the k-NN and classification tree algorithms the missing values are imputed using the median of the associated feature values. The missing values in the test set are also imputed with the median of the training set for the associated feature.

We observe that the training set is imbalanced with 63% of instances belonging to the benign class and 37% belonging to the malignant class. As the k-NN and classification tree algorithms are sensitive to imbalanced class distributions, we implement strategies to address this issue.

To address the issue of the imbalanced dataset for the k-NN algorithm we implement a weighted k-NN method that changes the way each neighbour contributes to the final class prediction. The contribution of the neighbour is changed to a function of the inverse distance between the query and training instance such that the contribution of each neighbour is given by

$$\frac{1}{d(\mathbf{x}_n, \mathbf{x})} \tag{6}$$

where $\mathbf{x}_n$ is the n^{th} nearest neighbour and $\mathbf{x}$ is the query instance. The algorithm sums the contribution of each instance to the associated target class and assigns the class with the highest sum to the query instance. This ensures that the majority class does not dominate the minority class with the consequence of poor precision for the algorithm.

For classification trees, imbalanced datasets may lead to instances of minority classes with small leaves which are then likely to be pruned and thus classification trees are sensitive to skew class distributions. In order to address the issue of the imbalanced dataset, we use Tomek links [4] to undersample the majority class. A Tomek link is identified between two instances of differing classes, $\mathbf{x}$ and $\mathbf{y}$ if there does not exist a $\mathbf{z}$ such that $d(\mathbf{x}, \mathbf{z}) <= d(\mathbf{x}, \mathbf{y})$ or $d(\mathbf{y}, \mathbf{z}) <= d(\mathbf{x}, \mathbf{y})$. We find all of the Tomek links in the training set and remove the instance from the Tomek link with the majority class.

Finally, for the k-NN algorithm the feature values are normalised to a scale between [0, 1] such that each feature carries the same weight in the distance calculation. This normalisation is done for each feature of each instance by subtracting the minimum value of the feature and dividing by the difference between the maximum and minimum value for that feature. If these feature values are not normalised some features with a larger range of values would have a bigger influence in the calculation of the distance measure and thus the model would be biassed towards those features.

B. k-nearest neighbours implementation

The k-NN algorithm is a very simple algorithm and the standard algorithm used for classification problems as described in Section II is implemented with some modifications to address the issue of an imbalanced dataset as described in Section III-A.

To calculate the distance between two instances, $\mathbf{x_1}$ and $\mathbf{x_2}$ we use the Euclidean distance measure which is given by

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{n=1}^{N} (x_{1n} - x_{2n})^2} \tag{7}$$

where N is the number of descriptive features of an instance.

In order to tune the hyperparameters for the k-NN algorithm the original training set is split up into a new training set and a validation set, with 70% for training and 30% for validation. The new training set is then used as the training instances and the validation set is used to obtain the optimal configuration for the algorithm.

The only hyperparameter for this algorithm is the value of k. The value of k determines the number of nearest neighbours to consider when selecting the class of a query instance. For smaller values of k the algorithm is more likely to be influenced by noise and overfit this noise. For larger values of k the algorithm becomes less precise and is more likely to underfit the set of training instances. We experiment with different values of k on the validation set to obtain the optimal configuration for the k-NN algorithm as the performance on the validation set is an estimation of the generalisation performance of the model. Finally, the tuned algorithm is evaluated on the test set to determine its generalisation performance.

C. Classification tree implementation

We implemented the classification tree as described in Section II with some modifications to allow for the use of continuous feature values and to avoid overfitting.

This predictive task uses continuous feature values and thus a threshold is chosen within the range of values to partition the instances based on whether the values for that feature are below or above that threshold. To determine the best threshold we find the unique values of the associated feature and select these values as possible threshold points. The information gain for each of these possible thresholds is computed and the value with the highest information gain is chosen to be the threshold for that feature at the specific node. The same procedure is followed for the other features and the feature with the highest information gain is chosen with its associated threshold.

The standard classification tree algorithm splits the dataset until all subsets associated with leaf nodes have an entropy of

0, *i.e.* there are only homogeneous subsets left. This leads to overfitting and bad generalisation performance as the leaves may contain outliers and noise. In order to prevent overfitting the implemented algorithm uses different stopping criteria for a node to not be split and thus to be changed to a leaf node. These stopping criteria include the minimum number of samples and minimum information gain to split on. The minimum number of samples to split on forces the algorithm to stop splitting the dataset once the number of entries in the set it is considering is less than this threshold. This can help prevent overfitting as the tree will become less complex and less precise such that it does not capture the effect of noise or outliers. Larger values for the minimum number of samples will result in simpler trees whereas smaller values will result in more complex trees prone to overfitting. The minimum information gain to split on allows the algorithm to stop splitting the dataset when a node obtains an information gain less than a certain threshold. This prevents overfitting as only nodes that lead to an information gain that is above this threshold will be included in the decision tree and nodes that do not increase the homogeneity above this threshold will be set to be a leaf node.

In order to tune the hyperparameters for the classification tree algorithm the original training set is split up into a new training set and a validation set, with 70% for training and 30% for validation. The new training set is then used as the training set to build the classification tree and the validation set is used to obtain the optimal configuration for the algorithm.

We experiment with different values for the minimum number of samples and minimum information gain hyperparameters on the validation set to obtain the optimal configuration for the classification tree algorithm. Finally, the tuned algorithm is evaluated on the test set to determine its generalisation performance.

IV. Empirical procedure

This section describes the procedure that was followed to obtain the empirical results for this project. These experiments were conducted to see how the two algorithms perform compared to each other as well as to find optimal configurations for each of them for the given problem.

The algorithms were measured in terms of accuracy which is simply the percentage of instances in the test set that are correctly predicted by the algorithm. To obtain this accuracy we used the algorithms to predict the outcome for each instance in a test set and count the number of instances that were correctly predicted. The total count is divided by the number of instances in the test set and multiplied by 100 to get the accuracy as a percentage.

The original training set is split up into a new training set and a validation set, with 70% for training and 30% for validation.

The k-NN algorithm with the new training set was tested on the validation set using different values of k to obtain the optimal configuration of the algorithm. The values of k were chosen in increments of 2 from 1 to 25. These values provide a good range for the tradeoff between overfitting and underfitting the training instances. Lower values of k are more precise but are prone to capturing the noise in the data and larger values do not capture noise but are less precise. This range of values thus tests the tradeoffs between these two aspects to find the optimal configuration for the algorithm. For each value of k the accuracy on the validation set was recorded. The optimal configuration was then selected based on the best performancing k value on the validation set. Finally, the accuracy of the algorithm was determined with this optimal configuration on the test set.

The classification tree algorithm was tested on the validation set using different values for its hyperparameters. These hyperparameters are used to prevent the tree from overfitting the training data such that the algorithm has good generalisation performance. Different combinations of values for the number of minimum samples and minimum information gain to split on were used to obtain the optimal configuration for the algorithm. The minimum number of sample values ranged from 10 to 25 in increments of 5. These values provide a good range for the minimum number of samples to split on as lower values lead to a more complex and precise model but higher values lead to simpler and less precise models. This range is thus used to obtain the optimal tradeoffs between these two aspects. The minimum information gain values ranged from 0 to 0.25 in increments of 0.05. These values provide a good range for the minimum information gain threshold to split on for the same reasons as the minimum number of samples. For each increment of the minimum number of samples, all increments for the minimum information gain were tested. This was done to obtain the optimal combination of these two hyperparameters with the goal of preventing the model from overfitting the data. For each of these combinations the accuracy on the validation set was recorded. The optimal configuration was then selected based on the performance on the validation set. Finally, the accuracy of the algorithm was determined with this optimal configuration on the test set.

V. Research results

This section presents and discusses the results obtained through the empirical procedure described in Section IV. First, we experimented with the hyperparameters of the k-NN algorithm to obtain the optimal configuration for the algorithm, *i.e.* the optimal value for k. Then, we experimented with the hyperparameters of the classification tree to obtain the optimal configuration for the algorithm, *i.e.* the optimal combination of the minimum sample size and minimum information gain to split on. Finally, we used the optimal configurations for each algorithm to test their performance on the test set.

A. Tuning k-NN algorithm

We obtained the accuracy of the k-NN algorithm on the validation set for different values of k to find the optimal value for this hyperparameter with regard to generalisation performance. Table I provides the accuracies obtained on the validation set for the different k values. From Table I we

observed that the optimal k value for the validation set was 9 and 11 with both values obtaining accuracies of 95.73%. These values of k provide a good balance with regard to avoiding capturing the noise in the data and not underfitting the training set. This configuration of hyperparameters thus results in the best generalisation performance for the given task. Furthermore, Table I indicated that smaller k values such as 1 and 3 lead to lower accuracies. These results can be attributed to the lower values of k leading to the overfitting of the training set as the algorithm is more prone to be affected by noise in the data because it is too precise. However, we also observed that larger k values such as 13 and 15 lead to a less accurate algorithm. These results can be attributed to the higher values of k leading to the algorithm underfitting the training set and becoming less precise as the number of neighbours grows too large.

TABLE I
ACCURACIES FOR DIFFERENT k VALUES ON VALIDATION SET

k	Accuracy (%)
1	90.60
3	94.02
5	94.02
7	94.87
9	95.73
11	95.73
13	94.02
15	94.87
17	94.87
19	94.87
21	94.87
23	94.87
25	94.87

B. Tuning classification tree algorithm

We obtained the accuracy of the classification tree algorithm on the validation for different combinations of the hyperparameters, minimum sample size and minimum information gain to split on, to obtain the optimal configuration for the classification tree. Table II provides the accuracies obtained on the validation set for the different combinations of parameter values. Table III provides the optimal combinations of minimum sample size and minimum information gain which all obtained an accuracy of 94.02% on the validation set. From Table III we observe that all of the optimal configurations occurred at a minimum sample size of 20 for varying minimum information gain values. These results indicate that a minimum sample size of 20 provides a good strategy for preventing the tree from growing too deep and complex. This prevention of overfitting results in better generalisation performance for the classification tree and thus these combinations are the optimal configurations for the classification tree. From Table II we observe that lower minimum sample sizes such as 10 and 15 lead to lower accuracies. This can be attributed to the model becoming too complex and overfitting the training data. However, higher minimum sample sizes such as 25 also lead to lower accuracies. This can be attributed to the model underfitting the training data and being too simple to capture the relationship between the conditions on the features and the output classes. Furthermore, from Table II we observe that minimum information gain values smaller than 0.25 do not impact the performance of the algorithm as the values for the different minimum sample sizes remain the same for these values of minimum information gain. However, for larger values of minimum information gain such as 0.25 the performance starts to deteriorate. This is because the algorithm is prematurely stopping nodes from being split even though these nodes may lead to a significant information gain and thus the algorithm is underfitting the training set.

TABLE II
ACCURACIES FOR DIFFERENT CLASSIFICATION TREE CONFIGURATIONS ON VALIDATION SET

Minimum sample size	Minimum information gain	Accuracy (%)
10	0	93.16
10	0.05	93.16
10	0.1	93.16
10	0.15	93.16
10	0.2	93.16
10	0.25	88.89
15	0	93.16
15	0.05	93.16
15	0.1	93.16
15	0.15	93.16
15	0.2	93.16
15	0.25	88.89
20	0	94.02
20	0.05	94.02
20	0.1	94.02
20	0.15	94.02
20	0.2	94.02
20	0.25	88.89
25	0	92.31
25	0.05	92.31
25	0.1	92.31
25	0.15	92.31
25	0.2	92.31
25	0.25	88.89

TABLE III
OPTIMAL CONFIGURATIONS FOR CLASSIFICATION TREE ON VALIDATION SET

Minimum sample size	Minimum information gain
20	0
20	0.05
20	0.1
20	0.15
20	0.2

C. Comparison of k-NN and classification tree

We use the optimal configurations for the k-NN and classification tree algorithms to obtain the accuracy for each algorithm on the test set. As both of these algorithms obtained more than one optimal configuration we randomly select one of the configurations for each algorithm and use those values for the evaluation on the test set. The value of k for the k-NN algorithm is thus chosen as 9.The hyperparameter values for

the classification are chosen as 20 for the minimum split size and 0.15 for the minimum information gain. The accuracy for these algorithms on the test set is provided by Table IV. From Table IV we observe that the k-NN algorithm outperformed the classification tree where the former obtained an accuracy of 93.30% and the latter obtained an accuracy of 91.62%. The performance of the k-NN algorithm can be attributed to the optimal configuration of the algorithm which allows the algorithm to avoid being influenced by noise in the data while not underfitting the training set. Furthermore, the k-NN algorithm is able to more effectively form complex borders between classes whereas classification trees need to use complex conditions on feature values to form these borders as they can only form axis-parallel decision boundaries. Thus, true decision boundaries that are not parallel to the axes require a more complex decision tree which potentially leads to capturing noise in the data and overfitting.

TABLE IV
ACCURACIES FOR k-NN AND CLASSIFICATION TREE ON TEST SET

Algorithm	Accuracy (%)
k-NN	93.30
Classification tree	91.62

VI. CONCLUSION

The objective of this project is to compare the performance of the k-NN and classification tree algorithms with regard to the task of predicting breast cancer tumours as benign or malignant. Furthermore, this project aims to determine the optimal configurations for each algorithm for the associated predictive task.

For each of the two algorithms the hyperparameters are tuned using a validation set to obtain an optimal configuration for each algorithm for the associated task. The optimal configurations are then evaluated on a test set to measure the performance of the algorithms compared to each other.

From the results discussed in Section V we derive that the optimal configuration for the k-NN algorithm with regard to the associated task is a k value of 9 or 11. Furthermore, the optimal configuration for the classification tree for the given task is a minimum sample split size of 20 and a minimum information gain less than 0.25. Finally, we derive that the k-NN outperforms the classification tree for the task of predicting breast cancer tumours as malignant or benign. The k-NN algorithm obtained an accuracy of 93.30% whereas the classification tree obtained an accuracy of 91.62%. Thus, the k-NN algorithm is the preferred algorithm for this predictive task.

This project can be extended by using different techniques for handling the data quality issues for each of the algorithms. Furthermore, different pruning techniques for the classification tree can be implemented and experimented with.

REFERENCES

[1] E. Fix and J. Hodges, "Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties", International Statistical Review / Revue Internationale de Statistique, vol. 57, no. 3, p. 238, 1989.

[2] J. Quinlan, "Induction of decision trees", Machine Learning, vol. 1, no. 1, pp. 81-106, 1986.

[3] C. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, vol. 27, no. 3, pp. 379-423, 1948.

[4] E. Swana, W. Doorsamy and P. Bokoro, "Tomek Link and SMOTE Approaches for Machine Fault Classification with an Imbalanced Dataset", Sensors, vol. 22, no. 9, p. 3246, 2022.