Multi-Class Classification of Dry Beans

Steffan Schoonbee

I. Abstract

An attempt at solving the multi-class classification problem of a dry beans dataset. This dataset has missing values, outliers, features with very different scales and a class imbalance. Two algorithms were tested on the dataset, the K-Nearest Neighbours and the Classification Decision Tree. The K-Nearest Neighbours algorithm was written from scratch while the sklearn implementation of the Classification Decision Tree was used. Data Quality issues were identified and addressed in algorithm-specific ways. A final model performance of 98% was obtained.

II. Introduction

Multi-class classification has a wide range of applications. In this study, we will explore the application of multi-class classification to the classification of dry beans. The dry beans dataset is a multi-class classification problem with 20 descriptive features. The goal is to fit a well-performing model on this training data without doing too many transformations to the data.

Data Quality is the starting point in the process. Different possible issues with data quality will be discussed, identified and addressed.

Two algorithms will be explained in detail, the K-Nearest Neighbours algorithm and Classification Decision Trees. These two algorithms are widely used and have been studied in-depth. Custom Imputer and K-Nearest Neighbours algorithms have been implemented with the latter making use of parallelization to speed up the prediction of observation significantly. The custom K-Nearest Neighbours algorithm can also take missing values as input which simplifies the training process.

The final model achieved an accuracy of 98% on testing data with very limited transformations being done the the training data.

III. Background

A. Overview

1) Supervised Learning: Supervised machine learning techniques aim to learn the relationship between descriptive features and a target variable. The techniques learn this relationship automatically from historical examples known as instances [7]. The algorithm tries to model *f(X)* in

$$Y = f(X) + e \tag{1}$$

where *f(X)* is the systematic information that *X* provides about *Y* and e is a random error term, which is independent of *X* and has a zero mean. [6].

2) Classification: Classification is a subset of Supervised Learning. Classification is when the target variable is qualitative [6]. The target variable consists of different classes. The simplest form of classification is binary classification which is when the target variable only consists of two possible classes. Multi-label classification is when multiple binary target variables are predicted. Multi-class classification is when a single target variable is predicted and the target can be any one of *k* classes where *k* ¿ 2 [1].

B. Data Quality

Data Quality in our context refers to observations that complicate the classification process. The performance of machine learning models is upper bound by data quality. Data quality is an umbrella term used for many concepts, we will only define relevant terms [5].

1) Outliers: Valid outliers are instances that are simply very different to the rest of the dataset. Valid outliers are not mistakes in the data, it simply makes the classification process more difficult. Invalid outliers are incorrect observations and should immediately be removed from the dataset. Invalid outliers are included in the dataset through error or a faulty calculation. [7].

2) Missing Values: Missing Values refer to instances for which no observation is recorded. Missing values can be imputed or dropped. Alternatively, the algorithm can be altered so that it can handle the missing values.

3) Measurements: Skewness measures the asymmetry of a dataset's distribution. Positive skewness indicates a longer right tail, while negative skewness indicates a longer left tail [9].

Formula:

$$\gamma_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^3}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\right)^{3/2}} \tag{2}$$

Kurtosis measures the "tailedness" of a distribution. High kurtosis indicates heavier tails (more outliers), while low kurtosis suggests lighter tails. Excess kurtosis compares the tailedness to a normal distribution, where a normal distribution has an excess kurtosis of 0 [9].

Formula:

$$\gamma_2 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\right)^{2}} \tag{3}$$

$$\text{Excess Kurtosis} = \gamma_2 - 3 \tag{4}$$

C. Preprocessing

Preprocessing refers to transforming the data to either address data quality issues or optimise the dataset for a specific machine learning algorithm.

1) Dummy Variables: Refers to changing categorical input features with *k* possible values into *k* different binary features where k_i indicates that an observation belongs to class *i* [4].

2) Standardisation: Standardisation is when the following transformation is applied to numeric features in a dataset

$$z_i = \frac{x_i - \mu}{\sigma} \quad \text{for } i = 1, 2, \ldots, n$$

. The resulting *zi's* have zero mean and a standard deviation of 1 [7].

3) Imputing: Imputing is a way to address missing values in a dataset. The missing values are not dropped and the algorithm is not modified instead, the values are populated with probable estimates. The most popular ways of imputing populating them with either the mean, mode or estimating the value with smaller models. [7].

4) Class Imbalance: When a categorical feature or target variable does not have an equal amount of observations in each class. This imbalance can lead to a model being based to predicting the majority class. In this case, a high accuracy but a low recall or precision can be obtained. [7].

D. Model Architectures

Two models will be considered, k-nearest Neighbors (KNN) and Classification Decision trees.

1) KNN: KNN was originally introduced by Fix and Hughes [3]. It is a form of Similarity-based learning. This is when an algorithm simply looks at what has worked well historically and predicts that. The algorithm thus does not try to model *f(X)* but looks at similar cases [7].

A similarity measure has to be defined that defines how similar one observation is to another. Four different similarity measures will be considered.

1) Euclidean Distance:

$$d_{\text{Euclidean}} d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{5}$$

2) Manhattan Distance:

$$d_{\text{Manhattan}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i| \tag{6}$$

3) Cosine Similarity:

$$\text{cosine\_similarity}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \tag{7}$$

4) Chebyshev Distance:

$$d_{\text{Chebyshev}}(\mathbf{x}, \mathbf{y}) = \max_{i=1,\ldots,n} |x_i - y_i| \tag{8}$$

The Algorithm: Given a positive integer *K* and a test case *x0*, the KNN Classifier first gets the K most similar points in the training data. The condition probability for class j is then estimated by the fraction of points in K that belong to class *j* [6].

The Parameters: Once the similarity measure has been defined the KNN algorithm only has a single parameter. The number of neighbors, *K*.

Algorithm 1: K-Nearest Neighbors (KNN)

Input: Labeled dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, query point x_q, number of neighbors k

Output: Predicted label y_q for the query point x_q

```
Initialize an empty list of distances;
foreach point (x_i, y_i) ∈ D do
    Compute the distance d(x_q, x_i);
    Add tuple (d(x_q, x_i), y_i) to the list of distances;
Sort the list of distances in ascending order;
Select the top k tuples from the sorted list;
Set y_q to the most frequent label among the top k tuples;
return y_q;
```

2) Decision Trees: Decision trees were first introduced into the field of computer science by J. Ross Quinlan [10]. In 1984 Leo Breiman *et al*. introduced the CART classification decision tree which popularised the algorithm [2]. Decision trees are forms of information-based learning, which means they determine which descriptive features provide the most information about the target variables [7].

Information gained measurement: To understand how trees work we first need to define an information gained measurement. The information gained measurement aims to define the informativeness of a descriptive feature. [7].

1) Gini Index:

$$G = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) \tag{9}$$

The Gini Index is a measurement of node purity where a small value indicates that the node has observations that belong mainly to a single class. [6].

2) Cross-entropy:

$$D = -\sum_{k=1}^{K} p_{mk} \log p_{mk} \tag{10}$$

Cross-entropy also measures node purity. Cross-entropy and the gini index are very similar numerically [6].

3) Log Loss:

$$\text{Log Loss} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log(p_{ik}) \tag{11}$$

Where y_{ik} is a binary indicator for when an observation belongs to class *i*. Log loss aims to maximise the probability of predicting the class correctly.

To address class imbalance the calculation of the entropy can be modified. The classes can be balanced by using class weights. Each class has a class weight that makes the information measurement more sensitive to the minority class and less sensitive to the majority class.

The structure of a Decision Tree:

1) Root and internal nodes: These indicate a test that needs to be carried out on the descriptive features. These tests split into two paths the predicting observation follows based on its descriptive features.
2) Leaf nodes: Indicate a predicted level of the target variable. Leaf nodes do not split up the data anymore and once an observation reaches a leaf node a prediction is made. [7].
3) Branches: All of the nodes are connected with branches that indicate the path that an observation will take when predicting.

How a Tree is built: The information a decision tree learns is where to split up the data in internal nodes in a way that leaf nodes are as pure as possible. To obtain these tests we aim to maximise the information gain with each split. This process of maximising the information works as follows:

1) The information gained measurement is computed as defined above with respect to the target variable. This is the original information value.
2) For each descriptive feature, partition the data into different sets using their feature values and sum each of the partitions' information measurements. This is the remaining information value.
3) Subtract the remaining information from the original. This is the information gain
4) Pick the test that maximises the information gain.

These splits are considered recursively. This is why the process is called recursive binary splitting. The dataset is split into two sets and the best split is chosen recursively.

Stopping Criterion: A tree stops training when one of the following criteria is met:

1) A pre-defined maximum tree depth is reached.
2) A pre-defined minimum number of observations remains in a leaf.
3) A pure node is reached. This means that all observations in a node belong to the same class.
4) The set of features left to test is empty.
5) The dataset is empty. This happens when splitting the dataset on a certain feature, which results in no more observations belonging to a certain partition.

The Parameters: Classification Decision Trees have four parameters.

1) Maximum tree depth: a stopping criteria.
2) Which Information gain measurement to use.
3) Minimum samples in a leaf node: a stopping criteria.
4) Class weights: the weights to use for the class weights defined above.

Algorithm 2: Classification Decision Tree

Input: Training dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, maximum depth D, minimum samples per leaf L

Output: Decision tree T

Function `BuildTree(`$\mathcal{D}$, $depth$`)`**:**
  if *all examples in* $\mathcal{D}$ *have the same label* **then**
    return a leaf node with that label;
  if $depth = 0$ *or number of examples in* $\mathcal{D} < L$ **then**
    return a leaf node with the most frequent label in $\mathcal{D}$;
  Find the best feature X_j and threshold t that minimize impurity;
  Split $\mathcal{D}$ into two subsets $\mathcal{D}_1$ and $\mathcal{D}_2$ based on $X_j \leq t$ and $X_j > t$;
  Create a decision node with the feature X_j and threshold t;
  Recursively build the left subtree T_L using $\mathcal{D}_1$ and $depth - 1$;
  Recursively build the right subtree T_R using $\mathcal{D}_2$ and $depth - 1$;
  return the node with left child T_L and right child T_R;
return `BuildTree(`$\mathcal{D}$, D`)`;

E. Model Performance

Evaluating model performance is an important part of the machine-learning process. It is important to not only pick the model that performs the best on training data. A model that performs well on unseen (test) data is preferred. It is thus recommended to partition a random part of the training data to be held out during the training process. This hold-out set is only used to determine final model performance. Hyperparameter tuning refers to the process of picking the optimal hyperparameters for a specific model to maximise model performance. One way to do Hyper-parameter tuning is with k-fold cross-validation.

1) k-Fold Cross Validation: This approach involves splitting the dataset into *k* different groups or *folds* of approximately equal size. The first fold is treated as a validation set of data and the model is trained on the *k* - 1 folds. The accuracy is then computed on the validation set. After this the second fold is treated as the validation set and the process is repeated until each fold has been treated as the validation set. The average of the *k* different accuracies is used as the final accuracy. [6].

2) Metrics: In the case of Multi-class classification, accuracy is defined as follows:

$$\text{Accuracy} = \frac{1}{n}\sum_{i=1}^{n} I(y_i = \hat{y}_i) \tag{12}$$

Accuracy is thus the proportion of correctly defined cases. It is important to also study the precision and recall for each class. A model can be better at predicting some classes than others

and it is important to investigate whether this is the case for your chosen model.

$$\text{Precision}_k = \frac{TP_k}{TP_k + FP_k} \tag{13}$$

$$\text{Recall}_k = \frac{TP_k}{TP_k + FN_k} \tag{14}$$

Where TP_k is the number of true positives for class k. FP_k is the number of false positives and FN_k is the number of false negatives for class k [11]

IV. Methodology

In this section, the methodology of the experiments will be described and defended. The custom KNN and imputation algorithms will be explained. The results of the experiments will also be given. The starting point, however, is always the data itself. Potential data quality issues will be identified after which the method of dealing with these issues will be discussed.

A. Data description

The data has 20 descriptive features. The target variable has five different classes. The 20 descriptive features have 19 numeric features and a single categorical feature *'Colour'*. In total, there are 13594 observations. This is quite a large dataset to predict 5 classes. The preliminary TSNE plot indicates we have well-separated classes with some overlap. It is thus possible to achieve quite a high accuracy. Although noise exists in the dataset as there is overlap in the decision boundaries, the noise is not so much that it makes classification impossible.

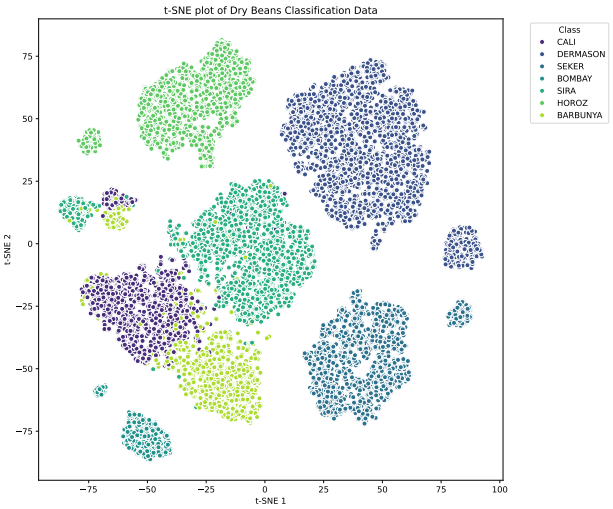

Fig. 1. Description of the image

Descriptions of the features and classes have been obtained from the original paper by Koklu and Ozkan [8]. We thus have the formulas for all the columns and a description of every class. These formulas will not be used to change the data. For the sake of the experiment, we will pretend we do not have the formulas and try to address missing values in a way that can generalize well to any dataset. These formulas were merely added to deepen our understanding of the dataset.

B. Potential Data Quality Issues

Table I gives a summary of the data. These data tables assist in identifying any possible data quality issues. They have been split into different tables for readability's sake. Before identifying potential issues it is important the remember the problem description. The problem is a supervised multiclass classification problem.

1) Outliers: As defined above we want to classify two types of outliers, valid and invalid. In table II a *ConvexArea* value of *-30* is impossible. This is clearly an invalid outlier. A mistake has been made in it's observation.

The *EquivDiameter* feature is also extremely skewed. A skewness score of 116 is far above the expected amount of variation for a feature. The feature also has a kurtosis score of 13592. There must be some mistake in this feature. The range of *EquivDiameter* is also 301 000 but only has a median of 215. One or two extremely large observations in this feature can thus be expected. The five biggest IV and smallest values V for *EquivDiameter* are very informative. There is a sudden jump for the biggest and second biggest values and a sudden drop for the smallest value. The combination of the extremely high skewness, sudden jump and drop in the values is enough evidence to define these as invalid outliers. Inspecting the formula for Equivalent Diameter confirms that the right decision was made. An Equivalent Diameter of zero only happens when the Area is also zero, which is not the case. From the formula, it can also be deduced that a significant figure error has been made in the two biggest values. Other columns also have high skewness values but they are not of the same magnitude. Thus no more observations will be defined as invalid outliers.

Area, *CovexArea* and *ShapeFactor5* specifically have high skewness and kutosis values. All three of these features have maximum and minimum values that are much different to the other values. *ShapeFactor5* has a big tail in it's distribution. *Area* and *CovexArea* have similiarly shaped distribution plots. All of the extreme values are however not outside the realm of possibility and are thus defined as valid outliers. These observations are simply much different to the rest of the observations.

2) Missing Values: Missing Values are present in the dataset. *Class* has 17 missing values. These 17 observations can not provide us with any information. Missing values are present in only four descriptive features, namely *Compactness*, *Extent*, *ShapeFactor6* and *Colour* VI. *Compactness* has the most missing values, 18. The other three features have 6 or fewer missing values. In total, we only have 52 rows that have missing values.

3) Class Imbalance: There is a class imbalance in our target variable. The classes vary much in their sizes. *DERMASON*

TABLE I
FEATURE DESCRIPTIONS AND FORMULAS

Feature	Description	Formula	NaN values	Type
Area	The area of a bean zone and the number of pixels.	$A = \sum_{r,c \in R} 1$ (where r, c is size of R)	0	int64
Perimeter	The circumference of the bean	P	17	float64
MajorAxisLength	The length of the longest line drawn through the bean.	L	0	float64
MinorAxisLength	The longest line perpendicular to the major axis.	l	0	float64
Eccentricity	Eccentricity of the ellipse with same moments as the region.	NA	0	float64
ConvexArea	Size of the convex polygon that can contain the region.	$C =$ Size of the convex hull	17	int64
Constantness	NA	NA	0	int64
EquivDiameter	The diameter of a circle having the same area as the region.	$Ed = \sqrt{\frac{4 \cdot A}{\pi}}$	0	float64
Extent	The ratio of the bounding box to the convex hull.	$Ex = \frac{A}{AB}$	6	float64
Solidity	The ratio of the pixels in the convex hull to those in the region.	$S = \frac{C}{A}$	0	float64
Roundness	Calculated with the following formula.	$R = \frac{4\pi A}{P^2}$	0	float64
Compactness	Measures the roundness of an object.	$CO = \frac{Ed}{\sqrt{A}}$	18	float64
ShapeFactor1	Ratio of the major axis length to the area.	$SF1 = \frac{L}{A}$	0	float64
ShapeFactor2	Shape factor calculated using specific formulas.	$SF2 =$ Specific formula	0	float64
ShapeFactor3	Shape factor calculated using specific formulas.	$SF3 =$ Specific formula	0	float64
ShapeFactor4	Shape factor calculated using specific formulas.	$SF4 =$ Specific formula	0	float64
ShapeFactor5	NA	NA	0	float64
ShapeFactor6	NA	NA	0	float64
Sort order	NA	NA	0	float64

TABLE II
STATISTICAL SUMMARY OF FEATURES (MIN, MAX, RANGE, SKEWNESS, KURTOSIS)

Feature	Min	Max	Range	Skewness	Kurtosis
Area	20420.0000	2.546160e+05	2.341960e+05	2.9547	10.8146
Perimeter	524.7360	1.985370e+03	1.460634e+03	1.6272	3.5943
MajorAxisLength	183.6012	7.388602e+02	5.552590e+02	1.3581	2.5337
MinorAxisLength	122.5127	4.601985e+02	3.376858e+02	2.2402	6.6641
Eccentricity	0.2190	9.114000e-01	6.925000e-01	-1.0627	1.3855
ConvexArea	-30.0000	2.632610e+05	2.632910e+05	2.9425	10.7535
Constantness	0.0000	1.000000e+00	1.000000e+00	-2.7203	5.4008
EquivDiameter	0.1614	3.014441e+06	3.014441e+06	116.5815	13592.1554
Extent	0.5553	8.662000e-01	3.109000e-01	-0.8964	0.6449
Solidity	0.9192	9.947000e-01	7.540000e-02	-2.5518	12.8160
Roundness	0.4896	9.907000e-01	5.011000e-01	-0.6365	0.3766
Compactness	0.6406	9.873000e-01	3.467000e-01	0.0374	-0.2215
ShapeFactor1	0.0028	1.050000e-02	7.700000e-03	-0.5349	0.7173
ShapeFactor2	0.0006	3.700000e-03	3.100000e-03	0.3010	-0.8599
ShapeFactor3	0.4103	9.748000e-01	5.644000e-01	0.2426	-0.1458
ShapeFactor4	0.6956	3.966100e+00	3.270500e+00	0.0079	-1.1834
ShapeFactor5	0.9477	9.997000e-01	5.200000e-02	-2.7596	13.0399
ShapeFactor6	0.0005	1.789850e+02	1.789846e+02	0.0060	-1.2009
Sort order	0.0001	1.000000e+00	9.999000e-01	-0.0107	-1.2024

has 3542 observations while *BOMBAY* only has 512 observations. Thus about 4% of the observations belong to *BOMBAY* while about 26% of observations belong to *DERMASON*. This is not an extreme class imbalance but needs to be taken into account when any algorithm is used for prediction.

C. The Algorithms

The implementations used for the model architectures are described in the Background. For the KNN model, a custom implementation was written that can handle missing values and class imbalances on its own. For the Decision Tree model, the sklearn implementation was used. This implementation can not handle missing values.

1) k-Nearest Neighbors: A custom model was implemented that can have missing values in the input. This is done by simply not including a feature that has a missing value in the calculation of the distance. In the dataset, no observation has more than one missing value so this will not hinder the accuracy of the model. The algorithm starts by computing a mask for each training observation indicating missing values. This is only done once to improve the running time of the algorithm. A mask of missing values is then computed for each testing observation. The bitwise AND operation is applied to the two masks to obtain the valid mask. The valid mask is applied to both the training and testing observation to compute the distance. In this way, a distance measurement can be

TABLE III
STATISTICAL SUMMARY OF FEATURES

Feature	Mean	Std	25%	50%	75%	Range
Area	53039.8926	29319.3987	36322.7500	44651.5000	61315.5000	2.341960e+05
Perimeter	855.2118	214.2551	703.4318	794.9405	976.9982	1.460634e+03
MajorAxisLength	320.1233	85.6939	253.2939	296.8800	376.4920	5.552590e+02
MinorAxisLength	202.2535	44.9560	175.8459	192.4269	216.9912	3.376858e+02
Eccentricity	0.7509	0.0920	0.7159	0.7645	0.8105	6.925000e-01
ConvexArea	53757.1373	29773.3641	36710.0000	45175.0000	62252.5000	2.632910e+05
Constantness	0.9028	0.2962	1.0000	1.0000	1.0000	1.000000e+00
EquivDiameter	476.5146	25853.0158	215.0525	238.4367	279.4345	3.014441e+06
Extent	0.7497	0.0491	0.7186	0.7599	0.7869	3.109000e-01
Solidity	0.9871	0.0047	0.9857	0.9883	0.9900	7.540000e-02
Roundness	0.8733	0.0595	0.8322	0.8832	0.9169	5.011000e-01
Compactness	0.7999	0.0617	0.7625	0.8013	0.8343	3.467000e-01
ShapeFactor1	0.0066	0.0011	0.0059	0.0066	0.0073	7.700000e-03
ShapeFactor2	0.0017	0.0006	0.0012	0.0017	0.0022	3.100000e-03
ShapeFactor3	0.6436	0.0990	0.5813	0.6420	0.6960	5.644000e-01
ShapeFactor4	2.3682	0.8716	1.6142	2.3688	3.1146	3.270500e+00
ShapeFactor5	0.9951	0.0044	0.9937	0.9964	0.9979	5.200000e-02
ShapeFactor6	89.3860	51.8312	45.2830	88.8112	134.2740	1.789846e+02
Sort order	0.5002	0.2880	0.2481	0.5037	0.7503	9.999000e-01

TABLE IV
FIVE BIGGEST VALUES FOR EQUIVDIAMETER

Rank	Value
1	3014441.0
2	24100.0
3	569.0
4	566.0
5	562.0

TABLE V
FIVE SMALLEST VALUES FOR EQUIVDIAMETER

Rank	Value
1	0.0
2	161.0
3	162.0
4	162.0
5	163.0

TABLE VI
COLOR DISTRIBUTION OF BEANS

Colour	Count
Brown	6098
Black	3541
Green	2023
White	1926
NaN Values	6

obtained between two observations even if there is a value missing.

To address the class imbalance when making a prediction Shepard Method was used. Shepard's Method only changes how the prediction is made with the *K* closest neighbours and does not change how the distances are calculated. In Shepard's Method, the different neighbours' votes for the predicted class are weighted based on their distance from the test observation. The predicted class is not determined simply by majority voting. The weights of the training observation are defined as one over the distance between the training and testing observation.

$$w_i = \frac{1}{d(x_i, x)} \tag{15}$$

where $d(x_1, x)$ is the distance the i-th nearest neighbour to the test observation.

Predictions are then made using Shepard's formula.

$$\hat{y} = \arg\max_{c \in C} \left(\sum_{i=1}^{k} w_i \cdot \mathbb{1}(y_i = c) \right) \tag{16}$$

where
$\hat{y}$ is the predicted class,
C is the set of possible classes,
w_i is the weight of the i-th NN,
y_i is the class of the i-th NN,
$\mathbb{1}(y_i = c)$ is an indicator function for y_i

This implementation is a weighted KNN implementation and is effective at addressing class imbalance [7]. The rest of the algorithm remains unchanged.

This implementation did work but was extremely slow. Parallelization was implemented to decrease running time. Predictions do not depend on each other and the prediction process does not change any global variables. Race conditions can therefore not occur and we do not have to implement any gates. This greatly simplifies the parallelization process and every prediction can be done in parallel. The speedup will depend on the number of cores on the machine. On the local machine, a speed-up of four minutes to 12 seconds per run was experienced.

Here follows the pseudo-code for the custom KNN algorithm.

TABLE VII
DESCRIPTIONS OF DIFFERENT TYPES OF DRY BEANS

ID	Bean Type	Count	Description
0	DERMASON	3542	This type of dry bean, which is fuller flat, is white in color, and one end is round while the other is round.
1	SIRA	2634	Its seeds are small, white in color, with a flat physical structure; one end is flat, and the other end is round.
2	SEKER	2025	Large seeds, white in color, with a round physical shape.
3	HOROZ	1927	Dry beans of this type are long, cylindrical, white in color, and generally medium in size.
4	CALI	1628	It is white in color, with slightly plump seeds that are slightly larger than typical dry beans and have a kidney shape.
5	BARBUNYA	1317	Beige-colored background with red stripes or variegated, speckled color; its seeds are large with a physical shape that is oval and close to round.
6	BOMBAY	521	It is white in color, with very big seeds and an oval and bulging physical structure.
7	NaN Values	17	N/A

Algorithm 3: Custom KNN)

Input: Labeled dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, query point x_q, number of neighbors k

Output: Predicted label y_q for the query point x_q

Initialize an empty list of distances for each point;
foreach *point* $(x_i, y_i) \in \mathcal{D}$ *in parralel* **do**
    Compute the distance $d(x_q, x_i)$;
    Add tuple $(d(x_q, x_i), y_i)$ to the list of distances;
Sort the list of distances in ascending order;
Neighbours $N =$ top k tuples from the sorted list;
The weights $(w(x_q, x_i), y_i) = \frac{1}{(d(x_q, x_i), y_i)}$;
Initialise an empty dictionary for the weighted votes;
foreach *Class* $c_i \in \mathcal{N}$ **do**
    Add weight value $(w(x_q, x_i), y_i)$ to the value in the dictionary for class y_i
Set y_q to the y_i with the highest weight value in the dictionary;
return y_q;

D. Classification Decision Tree

The sklearn implementation was used for the decision tree. Sklearn's implementation is an implementation of the CART algorithm. The Cart algorithm is described in the Background section. This implementation can not take missing values as input.

E. Imputation

A custom imputation method was implemented to populate missing values. We know that the features are related to each other through the different formulas. This can also be confirmed by a simple correlation analysis between the features. It is then possible to predict the value of a missing feature using the other features. As described in the Background section we try to populate the variables with probable values.

1) The Algorithm:

- The input is an unlabeled dataset with missing values.
- Start by creating a copy of the dataframe *C* and imputing the missing values with the mean for numerical features and the mode for categorical features.
- For each column containing missing values, fit a model on all the features excluding the column being used as the target variable. The copy dataframe *C* is used as the training data. If the target variable is numeric a Linear Regression model is fit. If the target variable is categorical a Naive Bayes model is fit. Both these model can not take categorical data as input, so one-hot encoding is used to replace any categorical features.
- Populate all missing values with the relevant model fit in the previous step
- Check for convergence.
- If convergence is not reached then the dataframe *C* is changed to the dataframe populated with the predicted values.
- Repeat the process and use the new *C* as the training data.

Convergence is when the predicted values do not change anymore. Compute the difference between the the dataframe *C* and the dataframe imputed with the fitted models. For numeric features compute the mean-squared error and for categorical features compute the proportion of prediction that differs from the copy dataframe. Although the mean-squared error is computed with the copy dataframe being used as the correct values, it is merely used as a measurement of how much the prediction change. If the predictions did not change more than a pre-defined tolerance value the iteration process stops.

Linear Regression and Naive Bayes were chosen as the models to fit as they do not require scaling. This allows us to use unaltered data as input. Both the models are also not computationally expensive and intuitive to understand.

2) Pseudo Code: :

Algorithm 4: Iterative Imputer

Input: DataFrame *df* with missing values, Max iterations *max_iter*, Tolerance *tol*
Output: DataFrame *df_filled* with imputed values

```
Initialize df_filled by imputing missing values with mean for numeric and mode for categorical;
for iteration = 1 to max_iter do
    foreach column col with missing values do
        if column is numerc then
            Fit a Linear Regression model;
        if column is categorical then
            Fit a Naive Bayes model;
        Impute missing values in col using the fitted model;
    Check for convergence;
    if converged or max iterations reached then
        break;
return df_filled;
```

F. Implementation

The implementation of the algorithms and the decision-making process will be explained here. For both implementations, the 17 observations that have missing values in the target variable were dropped. Observations with no target variable provide no information about the target variable and should be removed.

1) Preprocessing for the KNN Implementation: The missing values were not removed or dropped but left as is since our implementation can take them as input. As little bias as possible is introduced into the data by using this approach.

The four invalid outliers identified in IV-B1 are replaced with missing values. This was done to not change the dataset if it is not necessary. The implementation can take missing values as input. The valid outliers also identified in IV-B1 are not removed. The KNN algorithm is robust to outliers for small values of K and it is thus not necessary to remove these values [7].

The KNN algorithm's fundamental computation is the distance measure. The scale of a specific feature thus has a big effect on the size of the distance measure. For example, the *Area* has a range of 234 000 while *ShapeFactor5* only has a range of 0.005. This scale will skew your distance measure heavily to mostly penalise a difference in *Area* and not in *ShapeFactor5*. The input features should thus be scaled as described in III-C2.

The class imbalance is mitigated by using Shepard's method. This was specifically implemented as under-sampling reduces the size of a dataset. If each class is reduced to only have 521 observations VI only 3 584 observations would remain. This reduction in size is too severe to justify. If oversampling is used 3 021 observation of the *BOMBAY* class would be newly created or duplicates. Only one-sixth of the *BOMBAY* class would be from the original dataset. Again this is too severe to justify and it is more efficient to implement a weighted KNN.

Finally, dummy variables should be obtained for the *Colour* feature as the implementation can not compute the distance between two categorical features.

G. Preprocessing for the Decision Tree

The missing values were imputed using our custom Imputation method. The sklearn implementation method can not take missing values as input and they have to be replaced. There are not a lot of missing values and although bias is introduced it will not be a significant amount. This approach also allows us to test two different ways of handling missing values between the KNN implementation and the Decision Tree implementation. For the KNN implementation, a modification was made to the algorithm while for the Decision Tree, a modification was made to the dataset. This results in a more holistic approach when comparing the two methods.

The four invalid outliers identified in IV-B1 were also imputed using the Imputation method.

Decision Trees are robust to valid outliers so the valid outliers do not need to be removed [7]. The outliers will be in leaf nodes alone and will not have a noticeable effect on the overall structure.

Decision Trees are also robust to the scale of input features [7]. To define the best split the tree uses the information gain measurement, which is not affected by the scale of input features but rather by the purity of a node. Scaling the input feature is not necessary and the features are left as is.

The class imbalance is addressed by using weights. The purity measure of a node has an associated weight for each class. It is made more sensitive to smaller classes and less sensitive to majority classes. This method was chosen for the same reasons mentioned in IV-F1.

Finally, dummy variables should be obtained for the *Colour* feature as the sklearn implementation can not make tests for categorical features.

V. Empirical Procedure

In this section, the procedure used on the data will be discussed. Algorithm-specific transformation and hyperparameter tuning will be explained. The goal of this procedure is to predict the typing of a bean, based on its features. A high accuracy is expected since our classes have well-defined decision boundaries.

A. Adressing Data Quality Issues

The start of the procedure is the remove the invalid outliers as described previously IV-B1. The values with missing target variables are also dropped before anything is done.

B. Test set

A random sample containing 10% of the data is taken. These observations will be used as a test set to evaluate the final model performance. It will not be used for hyper-parameter tuning or for model selection. It will only be used once the final model has been chosen and will be used as truly unseen data to get an idea of how well the model generalizes.

C. K-fold Cross Validation

For both the KNN and Decision Trees algorithm we will use k-fold cross-validation III-E1 to do hyperparameter tuning. This ensures good generalization and helps prevent overfitting.

D. KNN

For the KNN four different distance measurements were tested, namely Euclidean, cosine-similarity, Manhattan and Chebyshev III-D1. For K 16 different values were tested. Every uneven number between 3 and 35 was tested. It was not needed to test any more neighbours than this as the model performance started deteriorating. The following procedure was followed to obtain the best hyper-parameters.

1) Preprocess the data IV-F1.
2) Pick a combination of distance measure and K.
3) Perform 10-fold cross-validation and obtain the average accuracy for that combination of hyper-parameters.
4) Pick the hyper-parameters that obtained the best results.

This combination resulted in 640 different fits between the hyper-parameters and the k-fold cross-validation.

E. Decision Tree

Decision Trees have more parameters to test than KNN. Every combination of the following parameters was tested:

- Information gain measurement: Gini, entropy, Log loss
- Maximum Depth: Every value between 3 and 50
- Minimum Samples for a split: Every value between 2 and 9
- Minimum Samples for in a leaf: Every value between 2 and 9

For all iterations, a balanced class weight was used. This is not a hyper-parameter but rather a way to address a data quality issue.

1) Preprocess the data IV-G.
2) Pick a combination of the hyper-parameters.
3) Perform 10-fold cross-validation and obtain the average accuracy for that combination of hyper-parameters.
4) Pick the hyper-parameters that obtained the best results.

This combination resulted in 69090 different fits between the hyper-parameters and the k-fold cross-validation.

VI. Research Results

The Results obtained from running the experiment in Empirical Procedure and using the implementations described in the Methodology will be discussed here.

A. KNN

The best-performing model obtained an average accuracy of 98.16% and used 11 nearest neighbours and the cosine similarity as the distance measure. A standard deviation of 0.003 was obtained. The worst-performing fold had an accuracy of 97.58% and the best-performing fold had an accuracy of 98.79%.

B. Decision Trees

The best-performing model obtained an average accuracy of 98% and used the Log loss criteria, a maximum depth of 9, minimum samples in a leaf of 9 and a minimum sample for a split equal to 2. A standard deviation of 0.004 was obtained. The worst-performing fold had an accuracy of 97.66% and the best-performing fold had an accuracy of 98.27%

C. Discussion

Both models performed well on the dataset. This was expected given that the target variable had well-defined decision boundaries. There was not a lot of deviation between the different folds and that indicates that both the models generalise well. Further, this also indicates that the dataset does not have a lot of deviation in it. This is also clear from the descriptions of the classes in the original paper VII. The beans are different with some being bigger or rounder. The decision boundaries of the target variables are clearly defined but are not linear. KNN and Decision Trees can predict non-linear decision boundaries. That is why these two models performed so well.

D. Final model selection and performance

The best-performing KNN model is chosen as the final model. Testing the model on the test set V-B obtained an accuracy of 98%.

VII. Conclusion

The goal of the experiment was to build a well-performing model that can do multi-class classification on the Dry Beans dataset. The data had missing values, outliers and a class imbalance. The invalid outliers were dropped, missing values were dealt with depending on the algorithm used and the class imbalance was solved by using weightings. Custom Imputer and KNN methods were implemented and the sklearn decision tree was also tested. The best-performing model was the custom KNN model and a test accuracy of 98% was obtained. For further improvement on the KNN algorithm, the hyper-parameter tuning process can be greatly simplified. Compute the distance between all training and testing points once and then find the best choice for K. In the current implementation, the distances are computed for every value of K tested.

References

[1] J. Bogatinovski, L. Todorovski, S. Džeroski, and D. Kocev. Comprehensive comparative study of multi-label classification methods. *Expert Systems with Applications*, 203:117215, 2022.

[2] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, USA, 1984.

[3] E. Fix and J. L. H. Jr. Discriminatory analysis: Consistency properties. Technical Report Report No. 21-49-004, Project No. 21-49-004, University of California, Berkeley, 1951. (Prepared at the University of California under Contract No. AF41(128)-31).

[4] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, 2 edition, 2009.

[5] A. Jain, H. Patel, L. Nagalapatti, N. Gupta, S. Mehta, S. Guttula, S. Mujumdar, S. Afzal, R. Sharma Mittal, and V. Munigala. Overview and importance of data quality for machine learning tasks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3561–3562, New York, NY, USA, 2020. Association for Computing Machinery.

[6] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Texts in Statistics. Springer, New York, NY, 2013.

[7] J. D. Kelleher, B. M. Namee, and A. D'Arcy. *The Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*. MIT Press, Cambridge, MA, 2015.

[8] M. Koklu and I. A. Ozkan. Multiclass classification of dry beans using computer vision and machine learning techniques. *Computers and Electronics in Agriculture*, 174:105507, 2020.

[9] K. V. Mardia. Measures of multivariate skewness and kurtosis with applications. *Biometrika*, 57(3):519–530, 1970. Accessed 3 Sept. 2024.

[10] J. R. Quinlan. Discovering rules by induction from large collections of examples. In *Proceedings of the International Conference on Artificial Intelligence (IJCAI)*, pages 668–677, 1979.

[11] Željko . Vujović. Classification model evaluation metrics. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 12(6):1–P1, 2021.