Final Year Project Report

Full Unit - Final Report

Machine learning for classifying mushroom edibility

Rickus de Goede

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Nicolo Colombo

Department of Computer Science
Royal Holloway, University of London

April 7, 2021

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 11259

Student Name: Rickus de Goede

Date of Submission: 07/04/2021

Signature:

Rdg

Table of Contents

Abstract

What this project is going to achieve is to program and implement supervised classification algorithms to test on an image mushroom data-set. These different algorithmic approaches are k-nearest neighbors, a decision tree based approach, random forest and a support vector machine. The models created by these algorithms will predict, based on only features extracted from the image data-set, what the edibility of the mushroom in the image is, then the project will attempt the multi classification problem of determining the species of the mushroom. As there are no set general rules for mushroom edibility across species, this may be a better approach. Key results metrics will be generated that are required for, comparing the algorithms. The best performing algorithm when dealing with the binary classification problem of identifying poisonous mushrooms was the k-nearest neighbours with a mean sensitivity result of 0.862 (where poisonous is 1, edible is 0). Between the 6 species of mushrooms tested in a multi-class species classification, the most accurate model was a support vector machine with linear kernel, using the one vs one approach to multi-class classification, with a mean accuracy score of 0.399.

Project Specification

Aims: To implement and compare on benchmark data sets various machine learning algorithms

Background: Machine learning allows us to write computer programs to solve many complex problems: instead of solving a problem directly, the program can learn to solve a class of problems given a training set produced by a teacher. This project will involve implementing a range of machine learning algorithms, from the simplest to sophisticated, and studying their empirical performance using methods such as cross-validation and ROC analysis. In this project you will learn valuable skills prized by employers using data mining.

Early Deliverables

- You will implement simple machine learning algorithms such as nearest neighbours and decision trees.
- They will be tested using simple artificial data sets.
- Report: a description of 1-nearest neighbour and k-nearest neighbours algorithms, with different strategies for breaking the ties;
- Report: a description of decision trees using different measures of uniformity.

Final Deliverables

- May include nearest neighbours using kernels and multi-class support vector machines.
- The algorithms will be studied empirically on benchmark data sets such as those available from the UCI data repository and the Delve repository.
- For many of these data set judicious preprocessing (including normalisation of attributes or examples) will be essential.
- The performance of all algorithms will be explored using a hold-out test set and cross-validation.
- The overall program will have a full object-oriented design, with a full implementation life cycle using modern software engineering principles.
- Ideally, it will have a graphical user interface.
- The report will describe: the theory behind the algorithms.
- The report will describe: the implementation issues necessary to apply the theory.
- The report will describe: the software engineering process involved in generating your software.
- The report will describe: computational experiments with different data sets and parameters.

Suggested Extensions

- Modifications of known algorithms.
- Dealing with machine learning problems with asymmetrical errors (such as spam detection) and ROC analysis.
- Nontrivial adaptations of known algorithms to applications in a specific area, such as medical diagnosis or option pricing.
- Exploring the cross-validation procedure, in particular the leave-one out procedure. What is the optional number of folds?
- Comparative study of different strategies of reducing multi-class classifiers to binary classifiers (such as one-against-one, one-against-the-rest, coding-based).

Chapter 1: **Introduction**

1.1 Aims and Objectives

Aim: To implement and compare algorithms for classifying mushroom edibility with image data, optimize and build a model with the high accuracy generalised performance. Why the project was chosen to be about image classification is because computer vision is a field of computer science that I would like to work in, therefore comparing the algorithms in relation to computer vision makes sense. The reason mushrooms were chosen is that it is a very difficult problem to determine whether a mushroom is poisonous or not, which makes it a difficult computer vision problem; it is difficult as there are no real set rules for edibility. If given two very similar mushrooms, one can be poisonous, the other edible. This project will take a different approach to the approaches taken in the wider academic publishing which tends to use convolutional neural networks for mushroom image classification, this project will focus on using the colour information from the images as samples for the classification, the reason being is that this technique could have wider applications to image classification problems of objects that are not photographed uniformly and from the same angle.

Objectives:

1. Program k-nearest neighbor algorithm.
2. Program a classification decision tree.
3. Program a random forest
4. Program a support vector machine
5. Program cross validation, confusion matrices
6. Pre-process the image data, this is done by reducing the noise by removing the background of the images, extracting colour information about the images and extract shape information.
7. Applying the algorithms to the pre-processed data of the mushroom images with binary edible or poisonous labels.
8. Applying the algorithms to the pre-processed data of the mushroom images with species as the label.
9. Comparing the results and discussing how best to improve them.

1.2 Image Data-set used

The mushroom images used are sourced from the Danish Svampe Atlas [29] and split into six categories with one hundred images each: Agaricus Arvensis, Amanita Muscaria, Amanita Rubescens, Boletus Edulis, Cortinarius Rubellus and Galerina Marginata. Each are species of mushrooms. Where Agaricus Arvensis, Amanita Rubescens and Boletus Edulis are edible and Amanita Muscaria, Cortinarius Rubellus and Galerina Marginata are poisonous. This data-set is exactly half edible, half poisonous.

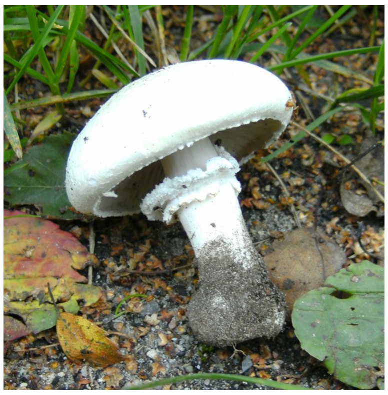


Figure 1.1: Agaricus_Arvensis (Edible)

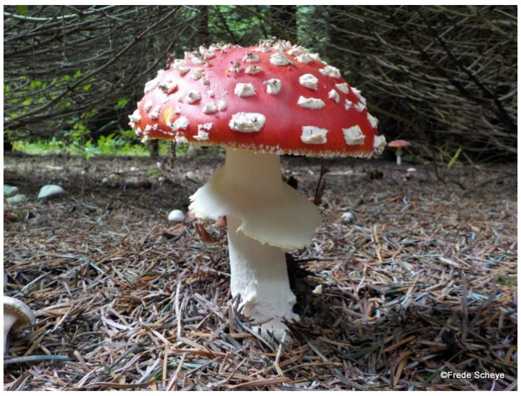


Figure 1.2: Amanita_Muscaria (Poisonous)

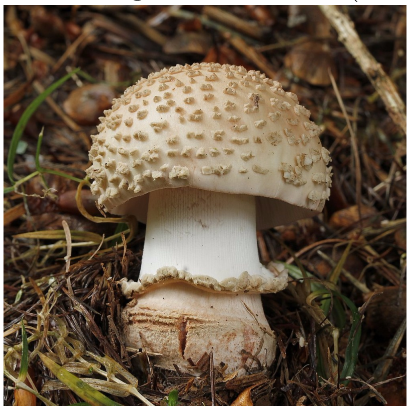


Figure 1.3: Amanita_Rubescens (Edible)

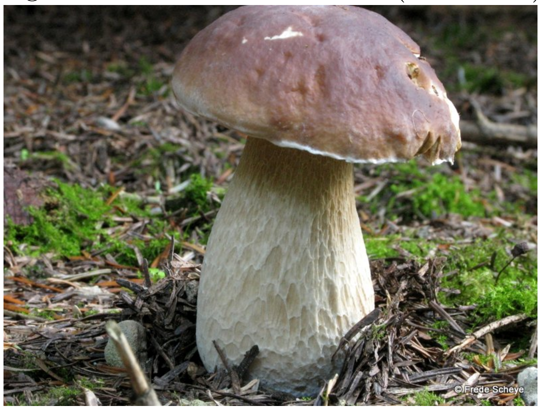


Figure 1.4: Boletus_Edulis (Edible)

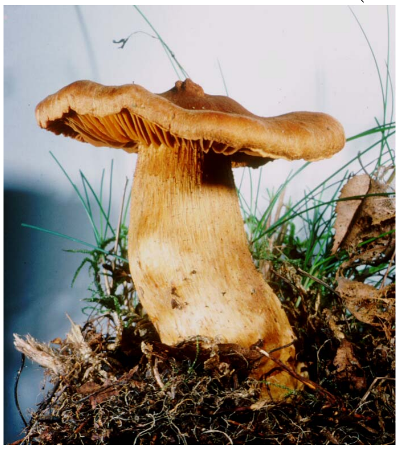


Figure 1.5: Cortinarius_Rubellus (Poisonous)

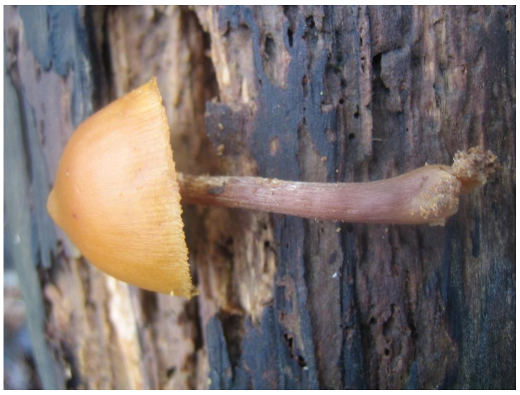


Figure 1.6: Galerina_Marginata (Poisonous)

Figure 1.7: An example of each species of mushroom used in this project

1.3 Background on mushrooms

” Mushrooms are a mold plant classified in the fungi Kingdom. The mushroom can grow and will become fiber which combines to create a structure called cap of the mushroom. The mushrooms are also classified as animals because it can synthesize food by itself [21]. Therefore, mushrooms rely on some food from various sources such as fossils, plants, organic matter, and etc. Moreover, mushrooms can grow well in the suitable weather conditions such as temperature, light, and humid environments. A life cycle of various mushrooms will start with spore; if the mushrooms are grown in the suitable weather conditions, it becomes hypha and mushroom fiber; hypha and fiber then will combine together in order to form the cap of mushrooms. If the cap of mushrooms grows older, it will produce spores which becomes hypha and then become mushrooms, respectively. Mushrooms have different sizes, shapes, colors and originations. The cap of mushroom also has several shapes and colors. Some mushrooms look similar but are different types. The structure of mushroom is illustrated in Figure 1.8

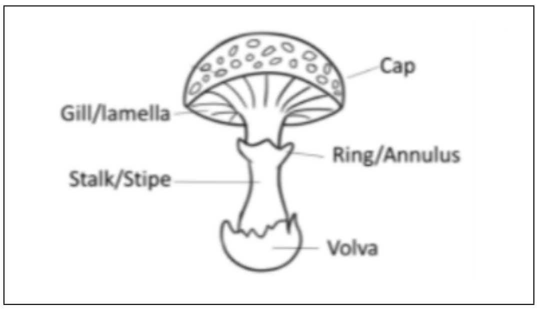


Figure 1.8: Image showing components of a mushroom

- Cap is located at the top of mushroom. If a mushroom grows properly, the cap of grown mushroom can have different shapes such as shape of inverted bowl, bell shape, pagoda shape, and parabola shape.
- Ring/Annulus is originated from thin membrane used for holding the stalk and cap of a mushroom when it is young.
- Volva is the outer part which holds the cap and the stalk of mushroom together. If a mushroom is growing up, the membrane will split out in order to stalk and the cap will be stretched out.
- Stalk/Stipe has a side hold with the cap of a mushroom and it has a different shape, size and color. The upper part holds the cap and gill.

” [22]

1.4 Algorithms used

1.4.1 Algorithm selection

k-nearest neighbors algorithm was chosen as it is a relatively simple algorithm to implement and optimise due to only having one parameter k therefore is a good first port of call in most machine learning problems.

In the mushroom kingdom representation in biology, species are organised in a tree like structure therefore there is an interesting question of whether or not the structure human experts have created to classify species results in a high performing machine learning algorithm for classification. Therefore decision tree was chosen to be used as an algorithm.

Random forest was chosen due to the fact that decision trees are prone to over-fitting and by implementing random forests could yield better performance by reducing over-fitting. Random forest is an ensemble method applied by making many decision trees trained on random sub-samples of the training set.

Support vector machines with a linear kernel can be very powerful in binary classification problems with linearly distributed data, however due to the fact that the project is implementing a multi-class classification problem of classifying species and then reducing this to the binary edibility classification problem, there are complexities involved. Overcoming these complexities will make this project more interesting. With the fact that SVMs are said to be just as powerful as one layered neural networks and much easier to train, I will be interested to see if performance rivals the results found by other studies using neural networks.

1.4.2 k-nearest neighbors

The Nearest-Neighbors algorithm is a simple, yet highly effective supervised algorithm in machine learning. It has a tendency to over-fit with small values of k, predictions are often accurate but can be unstable, however can achieve high accuracy for such a simple model. Proposed by Thomas Clover in the paper "Nearest neighbor pattern classification"[10] for use in classification and regression.

1.4.3 CART

"Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one. They are conceptually powerful. A popular method for tree-based classification and regression is called CART." [28]. The term CART was created by Leo Breiman to name decision tree algorithms that can be used for regression or classification problems.

1.4.4 Random Forest

Random forest (also called random decision forest) is an ensemble method for learning that is used for classification by constructing multiple different decision trees whilst training and outputting the class that is the mode of the classes predicted by the individual trees in the set of trees. Also proposed by Leo Brieman[5]

1.4.5 Support Vector Machine

The idea of a Support Vector Machine is to create a hyperplane that best splits the data into two classes. Support vectors are the points which are closest to the hyperplane which, if changed, will move the hyperplane. Due to the fact that the hyperplane best splits the data-set into two classes, methods are needed to apply this model to multi-class classification problems. Various approaches can be taken to apply this algorithm to a multi-class classification problem for example, one vs rest and one vs one. By changing the definition of the dot product, different kernels can be used, in this project only the linear kernel was implemented, meaning the definition of the dot product was not changed. Support vector machines were proposed by Vladimir Vapnik.

1.5 Literature survey

The paper "Classification of Mushroom Fungi Using Machine Learning Techniques" [20] found that their experimental results showed that the best technique for classifying mushroom images is kNN with accuracy of 94% based on features extracted from images with dimensions of mushroom types, and 87% extracted from images only. This study had access to the real dimensions of the mushroom (meaning actual measurements of size) therefore they could give size and shape data to their algorithms, my project does not have access to real dimensions therefore results are expected to be more in-line with the 87% achieved with features only extracted from the images.

The paper "Image Analysis of Mushroom Types Classification by Convolution Neural Networks"[22] found that with 45 species of mushrooms in their data-set 35 being edible and 10 being poisonous. Their proposed model gives the results of 0.78, 0.73, 0.74 of precision, recall and F1 score, respectively. The image dataset of 35 types has 6,000 images and the image dataset of poisonous mushrooms has 2,556 images, this is a lot higher than the 600 total images my project has. Therefore I will not be expecting to see the impressive results of this study.

A general issue I have found with the findings of the two aforementioned papers is not having the dataset that the study used available to others. This brings into question the replicability of results. If results are not replicable then they cannot be dis-proven. For this reason my dataset could be of far worse quality than the datasets used in these studies. It would have been best if I could have used an image dataset that was used in one of these two studies and compare my results with theirs however with the lack of such data I will use the dataset I put together from the Danish Svampe Atlas.

The paper "A Zoning based Feature Extraction method for Recognition of Handwritten Assamese Characters"[13] Even though not about mushroom image classification but about character recognition helped in me understanding why zoning was not applicable to the mushroom image classification problem. This strategy works well for characters because characters are always "upright" and exist in a two-dimensional space, therefore checking zones of the image for pixel density, turns out to be extremely beneficial. The mushroom image space is one of three-dimensional space where the 2-d zoning pixel density values do not provide much use.

1.5.1 Examples of applications of k-nearest neighbors

- Breast cancer diagnosis [25]
- Tea category identification on optimal wavelet entropy [33]
- Noninvasive blood pressure classification [27]
- Diabetes classification [9]

- Financial modeling [34]

This is a very versatile algorithm with application in a huge variety of fields.

1.5.2 Examples of applications of CART

- A statistical tool to investigate risk factor interactions with an example for colon cancer [7]
- Predicting stroke inpatient rehabilitation outcome [12]
- Tomato plant disease classification in digital images. [24]
- Credit risk modelling [35]
- classification of Takayasu arteritis (inflammation of the arteries) [2]

As displayed above, both have similar popular applications in biology and finance. This project lands in the domain of biology, therefore this project seems to be in the domain of the general application of these algorithms. Tomato plant disease classification in digital images [24] is a standout example as it involves image classification.

Chapter 2: **Theory**

2.1 Classification

Classification is the act of predicting the label of sample just given it's features. Labels are also called classes, targets and categories. Classification predictive modeling is the creating of a mapping function from input variables (which is commonly called X) to discrete output variables (which is commonly called y).

Classification is a type of supervised learning where the input data is given with the labels. There are two types of learners in classification, there are lazy learners and eager learners.

Lazy learners just store the training data and wait for testing data to be given, and when it is, classification is conducted based on the most closely related data in the training data. Compared to eager learners, lazy learners take less time to train but take longer to predict. k-nearest neighbors is an example of a lazy learner.

Eager learners create a classification model trained on data before receiving data for classification. It must be able to create a singular hypothesis that covers the entire instance space. Because of the fact that the model must be constructed when training, training takes a long time and the model takes less time to predict. Decision trees, random forests and support vector machines are all examples of eager learners.

[3]

2.2 Computer Vision

Computer vision is the field of trying to get computer programs to gain high level understanding from images and videos, simulating and automating tasks that the human visual system can do.

Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, e.g. in the forms of decisions.[15][16][18]

2.3 Nearest neighbors

2.3.1 Different measures of closeness

Let a and b be two n-dimensional vectors,

Dot product:

$$a \cdot b = \sum_{i=1}^{n} a_i b_i$$

L1 norm (Manhattan/Taxi-cab distance):

$$||x||_1 = \sum_{i=1}^{n} |x_i|$$

L2 norm (Euclidean distance):

$$||x||_2 = \sqrt{x \cdot x}$$

p-norm:

$$||x||_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$$

where a_i is the i^{th} component of a , b_i is the i^{th} component of b, n is dimension of vector space , x is a vector, x_i is the i^{th} component of x. L1 and L2 norms are p-norms where $p = 1$ and $p = 2$ respectively. All the norms can be expressed as dot products. L2 norm is the most popular used distance metric. Here is L2 norm expressed as a combination of dot products where x is a sample in the training set and x' is a test sample and $dist$ is the distance function:

$$dist(x, x') = \sqrt{(x \cdot x) + (x' \cdot x') - 2(x \cdot x')}$$

Due to the fact that distance can be expressed in dot products we can apply the kernel trick to the method and implement the kernel of choosing by swapping out the dot product with kernel function of choice K, so $dist$ becomes:

$$dist(x, x') = \sqrt{K(x, x) + K(x', x') - 2K(x, x')}$$

[31]

Neighbourhood function

$$D_i = dist(x', X_i)$$
$$N_0 = \{\}$$
$$N_{i+1} = N_i \cup \{min(D/N_i)\}$$

$N_k(x')$ is the neighbourhood of x', where k is number of nearest neighbors. D is the set of distances, x' is the test vector, X is the training set, $dist$ is the distance metric of choosing:

2.4 k-nearest neighbors definition

Nearest-Neighbor methods use elements in the training set closest to x' to form $\widehat{Y}$. The k-nearest neighbor fit for $\widehat{Y}$ (for classification) is defined as follows:

$$\widehat{Y}(x') = mode(y_i | x_i \epsilon N_k(x')), \qquad [28]$$

where $mode()$ is the most frequently occurring number, $N_k(x)$ is the neighbourhood of x'; the neighborhood is defined by the k closest elements in the training set. We must define a way of measuring closeness and therefore define a distance metric, often euclidean distance is used. We have now seen the two parameters of the k-nearest neighbor model, these are the value of k and the metric chosen to be used for closeness. For regression problems we do not use the mode of the labels given by $x_i \epsilon N_k(x')$, we instead use the mean.

2.4.1 1-nearest neighbor definition

The nearest neighbor method is the k-nearest neighbor model when $k = 1$ therefore the nearest neighbor fit for $\widehat{Y}$ (for regression and classification) is defined as follows: $\widehat{Y}(x') = y_i | x_i \epsilon N_1(x')$

2.4.2 k-Nearest Neighbors in pseudo-code

1. Calculate distance of test sample to all samples in training-set.
2. Arrange these distances in ascending order.
3. Select the samples that produced the first k distances.
4. Then for regression calculate the mean of the labels of this sample and this is the predicted label for the test sample or for classification calculate the mode of the labels and this would be the predicted label for the test sample.

2.4.3 Methods for tie-breaking

Especially when using k-nearest neighbors for classification problems it is possible to encounter a situation when there is a draw between labels of the nearest samples. There are a few approaches to dealing with this problem:

- **Random tie-breaking:** This method breaks ties as randomly as possible, this solves the tie-break however is inconsistent as every-time the same tie-break happens a different outcome is given; This could make the algorithm's accuracy scores less consistent.
- **Using weights:** This gives each sample an importance metric, this can often be the 1/distance to the test sample, then if there is a tie between labels the weightings of the samples are added up and the samples of the same label that have the highest sum of weights wins the tie-break. However, this creates the opportunity for another form of tie-break which can happen if the sum of weightings are equal, then you still haven't solved the tie-break.

2.4.4 Selecting k

A small value of k is more sensitive to noise of the data and is a more complex model. A larger value of k is less sensitive to noise and is a simpler model. The optimal model complexity achieves higher the larger the training set becomes.

Using Cross Validation: Using the training samples and labels calculate a cross validation score for a range of values of k, this gives an idea of generalisation. Then choose the value of k that achieves the highest cross validation score, this doesn't involve data snooping as only the training set is used.

2.4.5 Strengths and weaknesses of k-nearest neighbors

A strength of the k-nearest-neighbors model is that it is simple and easy to understand, and often gives good performance without too many adjustments of parameters. It is good to try before thinking of using more advanced techniques.

Another strength is that the algorithm could be implemented in an online format very easily as training is O(1) time complexity, so re-training with new data iteratively is possible.

A strength is that building the model is fast. A weakness however is prediction can be slow for data sets with large number of features and or a large number of samples. For k-nearest-neighbors time complexity for training is O(1) but time complexity for testing is O(N).

The model often does not perform well when a data-set has many features and when many of those features are 0 which is called a sparse data-set.

Another weakness is that it is important to pre-process your data so that distance measurements between features are scaled to the range of the features.

A further weakness is that due to storing the training-set and working out distances, the algorithm is very memory intensive compared to other algorithms.

2.5 Decision tree (CART)

Classification trees

The algorithm tries to split the training-set on all the classes contained in features in turn. Then applies an impurity measure on all the splits and selects the class that caused the purest split (not equal). Then recursively calls itself on the split data-sets until a pre-defined stopping criterion for leaf size is reached. The tree is comprised of the resulting category questions that will predict a test sample by sending it to a leaf node in the tree and the most common label in the leaf node is returned.

2.5.1 Impurity metrics

Algorithms for constructing decision trees usually work top-down, by choosing a variable at each step that best splits the set of items. [23]

2.5.2 Gini impurity

for j classes, $i \epsilon (1, 2, 3, ..., j)$, p_i is the fraction of items labelled with class i in the set:

$$gini(p) = 1 - \sum_{i=1}^{j} p_i^2$$

2.5.3 Information gain

Used by C4.5, C5.0 and ID3, tree-generation algorithms. Information gain is based on the concept of entropy and information content.

$$gain(p_1, p_2, ..., p_i) = - \sum_{i=1}^{j} p_i log_2 p_i$$

[32]

2.5.4 Classification decision-tree in pseudo-code

"

1. Assign all training instances to the root of the tree. Set current node to root node.
2. For each attribute
   (a) Partition all data instances at the node by the value of the attribute.
   (b) Compute the information gain ratio from the partitioning.
3. Identify feature that results in the greatest information gain ratio. Set this feature to be the splitting criterion at the current node.
   - If the best information gain ratio is 0, tag the current node as a leaf and return.
4. Partition all instances according to attribute value of the best feature.
5. Denote each partition as a child node of the current node.
6. For each child node:
   (a) If the child node is "pure" (has instances from only one class) tag it as a leaf and return.
   (b) If not set the child node as the current node and recurse to step 2.

" [4]

2.5.5 Pre-pruning

In pre-pruning, we manipulate parameters such as minimum leaf size, maximum depth of the tree, threshold for purity of split and use them as stopping criterion to avoid over-fitting. This can be tricky to carry out in practice as picking the optimum time to stop the tree is difficult to find out.

2.5.6 Post-pruning

”In post-pruning first, it goes deeper and deeper in the tree to build a complete tree. If the tree shows the over-fitting problem then pruning is done as a post-pruning step. We use a cross-validation data to check the effect of our pruning. Using cross-validation data, it tests whether expanding a node will make an improvement or not.

If it shows an improvement, then we can continue by expanding that node. But if it shows a reduction in accuracy then it should not be expanded i.e, the node should be converted to a leaf node.” [4]

2.5.7 Limitations of CART

- **Instability of trees:** A big issue with decision trees is that they have high variance. A very small alteration in the data-set can lead to different splits. A major reason for this instability is that the nature of the model is hierarchical, the effect of an error on a split at the top of the tree has an effect on all the splits underneath it, a way of getting round this issue could be to have a more stable split criterion however the instability does not go away. It is the cost of having a simplistic tree model.[28]
- **Lack of smoothness:** Trees do not produce a smooth prediction surface due to the partitioning of the feature space into a set of rectangles, these sharp decision boundaries, this becomes more of an issue when it comes to a regression problem as then it would be expected that the underlying function would be smooth.[28]
- **Binary splits:** We could consider using multi-way splits at each level instead of using binary splits, while this is possible it is usually not the best approach to go with as it can lead to the data getting split too quickly, which doesn’t leave enough data lower down in the hierarchy. Multi-way splits can be represented with combinations of binary split, therefore the most commonly used split is binary.[28]
- **Over-fitting:** Individual tree methods have a tendency to over-fit however this issue can be overcome by the use of ensemble methods such as bagging and random forest.

2.5.8 Ensemble methods

- **Bagging:** Also called bootstrap aggregation, is used when we need to avoid over-fitting and minimise the variance of the tree. We create multiple subsets from the training sample chosen at random. Each section of subset data is now used to train the decision trees. We end up with the resulting ensemble of multiple models each trained on a different section of the training-set. We then use the average prediction from all trees in the ensemble which results in a more robust model than using a singular tree. [19]
- **Random forest:** This method extends bagging. It goes one step further where on top of taking the random subsets of the training-set, it also takes a random selection of

features, therefore it drops certain features and only uses the selected features to grow the tree. A strength of random forest is it handles data with high-dimensionality well due to random feature selection; however this model can lack in precision in regression problems due to the fact that the mean prediction is taken from all singular trees in the random forest. [19]

- **Boosting:** This method is an ensemble technique to create multiple predictors. In this technique, models are trained in sequence with early models training simple trees on the subsets of the training-set and then scrutinising the results for errors. Another way to explain this is we fit consecutive trees (using a random subset of the training set like in the other ensemble methods) and after every tree, the aim is for the next tree to solve for the sum of error from the previous tree. [19]
- **Gradient boosting:** This method is an extension of boosting involving the use of gradient descent. The gradient descent algorithm can optimize any loss function that is differentiable. Due to the fact that the trees are built in turn trying to minimise the net loss, the gradient descent algorithm can minimise this loss. The next tree can use this algorithm to recover the loss (this loss is the difference between predicted and actual values). A strength of this method is that different loss functions can be used however it does tend to over-fit, also it requires tuning of hyper-parameters. [19]

2.6 Support Vector Machines

2.6.1 Support Vector Classification

The aim of Support Vector classification is to devise a computationally efficient way of learning 'good' separating hyperplanes in a high dimensional feature space, where by 'good' hyperplanes we will understand ones optimising generalisation. Generalisation theory gives clear guidance about how to control capacity and hence prevent overfitting by controlling the hyperplane margin measures, while optimisation theory provides the mathematical techniques necessary to find hyperplanes optimising these measures. Different generalisation bounds exist, motivating different algorithms: one can for example optimise the maximal margin, the margin distribution, the number of support vectors, etc. This theory section will consider the approach which this project will take and that will be to reduce the problem to minimising the norm of the weight vector.[11]

2.6.2 The formulation of a support vector machine

$$g(x) = w^T x + b \tag{2.1}$$

Maximise k such that:

$$-w^T x + b \geq k \quad \text{for} \quad d_i = 1 \tag{2.2}$$

$$-w^T x + b \leq k \quad \text{for} \quad d_i = -1 \tag{2.3}$$

Value of $g(x)$ depends upon $||w||$:

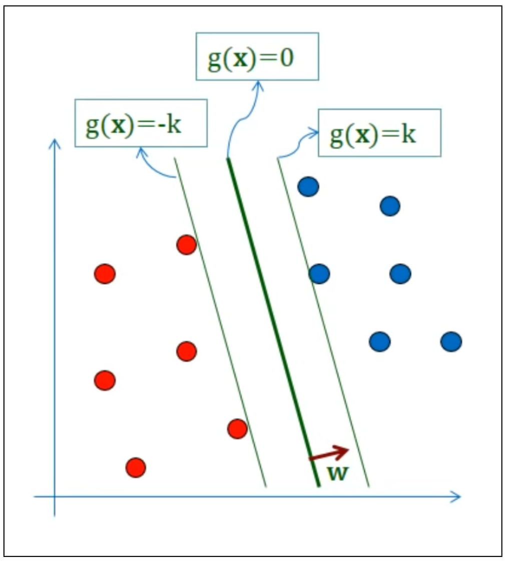


Figure 2.1: An example of a margin

1. Keep $||w|| = 1$ and maximise $g(x)$ or,
2. $g(x) \geq 1$ and minimise $||w||$

[1]
This project implements the second approach for the programmed implementation and uses gradient descent to minimise $||w||$ all while keeping $g(x) \geq 1$. Therefore, as said previously minimising the norm of the weight vector w.

2.6.3 Making the support vector machine work for multi-class classification

In its most simple type of SVMs are applied on binary classification, dividing data points either in 1 or 0. For multiclass classification, the same principle is utilized. The multiclass problem is broken down to multiple binary classification cases, which is also called one-vs-one. In scikit-learn one-vs-one is not default and needs to be selected explicitly. One-vs-rest is set as default, it is called one_v_one and one_v_rest in my implementation of my support vector python code for multiclass classification . One-vs-rest basically divides the data points in class x and rest. Consecutively a certain class is distinguished from all other classes. The number of classifiers necessary for one-vs-rest multiclass classification is only n (with n being the number of classes).The number of classifiers necessary for one-vs-one multiclass classification can be retrieved with the following formula (with n being the number of classes):

$$\frac{n(n-1)}{2} \tag{2.4}$$

Therefore is more computationally taxing than the one-vs-rest approach. In the one-vs-one approach, each classifier separates points of two different classes and comprising all one-vs-one classifiers leads to a multiclass classifier. The algorithms then all give their predicted label for a certain sample then the label with the most predictions is given to the sample. [17]

2.6.4 Limitations of Support Vector Machines

- Support Vector Machines tend to not scale very well with the number of samples increasing. Running an Support Vector Machines on data with up to 10,000 samples might work well, but working with datasets of size 100,000 or more can become challenging in terms of runtime and memory usage.
- They require careful preprocessing of the data and tuning of the parameters.
- Support Vector Machine models are hard to inspect; it can be difficult to understand why a particular prediction was made, and it might be tricky to explain the model to a nonexpert.
- They tend to work better with “homogeneous” data.

[30]

2.7 Cross validation

2.7.1 Definition

The simplest and most widely used method for estimating prediction error is cross-validation. Divide up the data into an arbitrary number of segments (called folds), use each segment in turn as the test set and the rest as the training set. The result is then summarised by adding the tallies of the correct and incorrect results. When the the number of folds is K and the sample size is m, if $K = m$ this is called ”leave one out cross validation”.

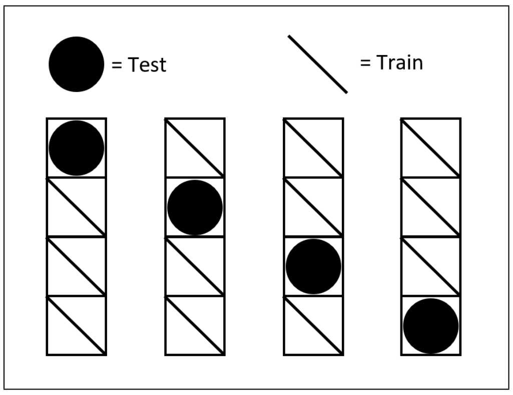


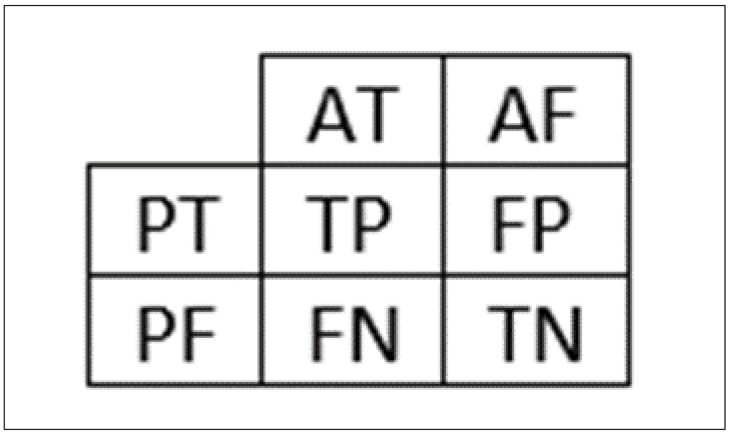
Figure 2.2: Four Fold Cross Validation

2.7.2 Why use cross validation?

The reason cross validation is used is to give a sense of generalisation only using the training-set, this gives an idea of the validity of the model. This can also be used for parameter selection as the best performing parameter in cross validation will often be the best generalised parameter.

2.8 Confusion Matrices

Matrix to represent the number of true positives(TP), true negatives(TN), false positives(FP) and false negatives(FN). In figure 2.3 is a graphical representation of matrix that would be used for a binary classification problem. Where AT stands for actual value true, AF stands for actual value false, PT stands for predicted value true, PF stands for predicted value false. This format will be used for all confusion matrices in this report.



	AT	AF
PT	TP	FP
PF	FN	TN

Figure 2.3: Binary Confusion Matrix

$$Accuracy = \frac{True\_Positives + True\_Negatives}{True\_Positives + True\_Negatives + False\_Positives + False\_Negatives}$$

The accuracy is the proportion of correct predictions (both true positives and true negatives) among the total number of cases examined. [8]

$$Precision = \frac{True\_Positives}{True\_Positives + False\_Positives}$$

Precision is what proportion of labels predicted true by the algorithm were actually true.

$$Sensitivity = \frac{True\_Positives}{True\_Positives + False\_Negatives}$$

Sensitivity (True Positive rate) measures the proportion of positives that are correctly identified. [14]

$$Specificity = \frac{True\_Negatives}{True\_Negatives + False\_Positives}$$

Specificity (True Negative rate) measures the proportion of negatives that are correctly identified. [14]

2.9 ROC curve

2.9.1 Definition

ROC stands for receiver operator characteristic. It is the curve that results in plotting 1 - specificity (false positive rate) on the x-axis and sensitivity on the y-axis.

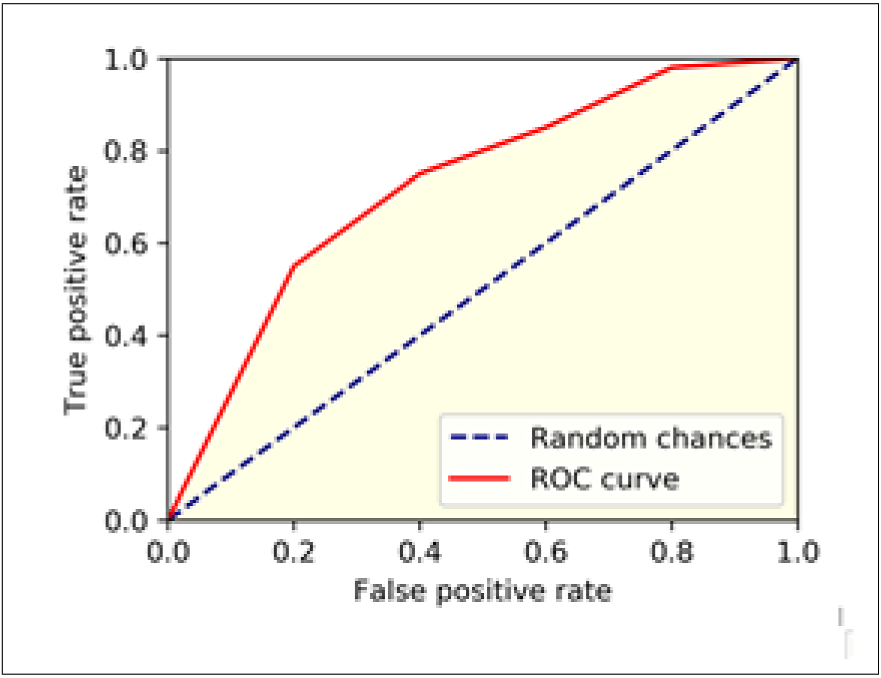


Figure 2.4: Example of a ROC curve [6]

2.9.2 AUC (area under the curve)

The AUC is the area under the ROC curve. The larger the area under the ROC curve the better the classifier is. AUC gives an aggregate measure of performance over all classification thresholds.

Chapter 3: **Implementation**

3.1 Which programming language was used?

The programming language used was python however as python is a relatively slow language the program relies on using many Numpy library functions as the functions of this library are written in C which run much faster. The program encountered heavy performance issues when the interpreter would spend too much time in the python for-loops, therefore, where possible, python for-loops were avoided. As these algorithms are relatively simple the slow performance is bearable and the results are accurate.

3.2 Modules and Notebooks

3.2.1 Description of modules and notebooks

"scoring/scores.py" module

This module was programmed and includes the classes cross_validation_score and confusion_matrix. Cross_validation_score is a correct implementation of the cross validation procedure, it takes a data-set, then partitions the data-set into folds using numpy slicing indices, then trains and tests on selected folds in turn and returns an array of scores from the cross validation. This is in the scores module because in this project cross-validation is used in parameter optimisation to give generalised scores. Confusion matrix takes as parameters two numpy arrays, one is the predicted values, and the other is actual values; it then returns a numpy array with the shape and contents of a binary classification problem's confusion matrix. The confusion_matrix class also contains the functions, accuracy, precision, sensitivity and specificity and these are used to return the corresponding statistical measure defined in their name; these functions take no parameters and they use the true positive, true negative, false positive and false negative values to calculate the accuracy, precision, sensitivity and specificity respectively.

"scoring/graphs.py" module

The module graphs.py contains the class graphScore which takes a dictionary of the results for an experiment where the keys are the names of the algorithms and the values are the corresponding scores and plots them on a bar chart to display the results.

"algorithms/nearestneighbor.py" module

This module includes the class KnnClassifier which contains three essential methods for carrying out classification problems which are: fit, predict and score. Fit takes in the training data for the model to use, predict returns an array of postulated labels produced by the algorithm given a test-set and score returns the accuracy score which is also $1-$ error_rate, where the error_rate is the number of miss-classified samples divided by the number of samples.

"algorithms/decisiontree.py" module

This module contains the class TreeClassifier which is a python implementation of the decision tree algorithm and it also contains the fit, predict, and score methods, this classifier works for multi-class classification also. This module additionally contains the class RandomForestClassifier class is an implementation of the ensemble method of random forest discussed in the theory section. This class implements bagging and random sub-set selection using a seed in the random sub-set generator so that the same random result is given every-time a certain seed is given.

"algorithms/supportvector"

This module contains the class SVC which stands for support vector classifier. This class is an implementation of the support vector machine algorithm that only works for binary classification. Contains the fit, predict, and score methods. This module also contains the class SVC_multi and this is an implementation of the support vector machine that works for multi class classification problems. SVC_multi contains fit, predict and score methods, also a definition of the dot product so that the implementation of kernels is possible. SVC_multi's fit function has a parameter that is called method and when method is set to "one_v_one" the algorithm carries out the one against one approach to getting the support vector machine to work with multi-class problems. If method is set to "one_v_rest" then the algorithm will carry out the one against rest approach to getting the support vector machine to work with a multi-class problem.

"notebooks/data_image_loading" notebook

The image data is read in from files corresponding to their respective species there is then code that takes the images and removes the backgrounds, they are then saved in the file they were previously with the same name, therefore replacing all the images in the file with images without their backgrounds, the code will delete images that produce sparse matrices after the background is removed as it will not be a good representation of the mushroom if almost all the image is background.

"notebooks/image_feature_extraction_color" notebook

The pixel data of the images with no background is read in and is then given to a function that calculates a color histogram from the pixel data. The test train split of images is done at this stage and the data is saved in different csv files.

"notebooks/main_project_notebook" notebook

In this notebook the algorithms are optimised, trained and compared. This notebook uses all the code written for this project and is the culmination of the programming.

3.3 Data normalisation

It is important to normalise the features so that they are measured on roughly the same scale. It is important to avoid data-snooping when carrying out normalisation therefore the test set should not be used. There are a few classes in the sci-kit learn package for python available for this purpose. I will outline the Scalers and explain how they function.

- **StandardScaler:** For every feature in the resulting data-set the mean is zero and the variance is one.
- **RobustScaler:** Still makes sure that the features are measured on the same scale but uses the median and quartiles to avoid using outliers in the calculations. It shifts the data down by its median and divides by the interquartile distance. Outliers can be an issue for other Scalers.
- **MinMaxScaler:** Moves data so that all features lay inbetween zero and one by subtracting the minimum value for a feature from all values for that feature and dividing by the maximum value for that feature.
- **Normalizer:** This divides each sample by the euclidean norm of the sample. This means that each sample is normalised independently therefore the training and test set can be normalized together with this approach as there won't be any data snooping.

The StandardScaler was chosen as the Scaler of choice on the image-data due to the variance controlling nature. Every step necessary was taken to avoid data-snooping, scalers fit to a training set were only applied to the validation set and test set. The scaler used was never fit using the validation set or the test set.

3.4 Preliminary testing done with nearest neighbours and decision tree algorithms

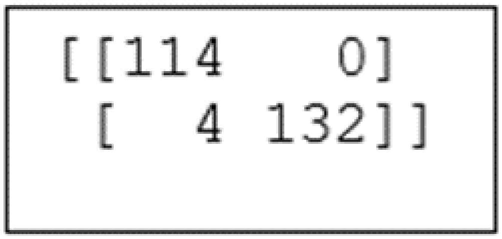
The tests carried out to assess the performance of the algorithms that were programmed are they were tested on the UCI mushroom data-set, which is a data-set of hypothetical features of mushrooms. This data-set was used to test as the data is extremely clean as it wasn't generated in a real world environment and it lies in the domain of this project.

3.4.1 UCI mushroom data set

This data set contains many physical features that could possibly be extracted using feature extraction from pictures of mushrooms. This hypothetical categorical data can give a good idea of the performance of the algorithms if the computer vision was 100% accurate and gathered all features. The data-set does contain some features which can not be extracted from pixel data, such as odor and spore print color.

3.4.2 kNN UCI mushroom data results

On the hypothetical features of the UCI dataset with a sample size of 1000, $k = 3$, with a training split of 75% train, 25% test, using euclidean distance, the algorithm achieved the accuracy score of 0.984 (3.s.f), sensitivity=0.966 (3.s.f), specificity=1.00(3.s.f); confusion matrix:



```
[[114   0]
 [  4 132]]
```

Figure 3.1: Confusion Matrix achieved when k-nearest neighbors is used on UCI mushroom data-set

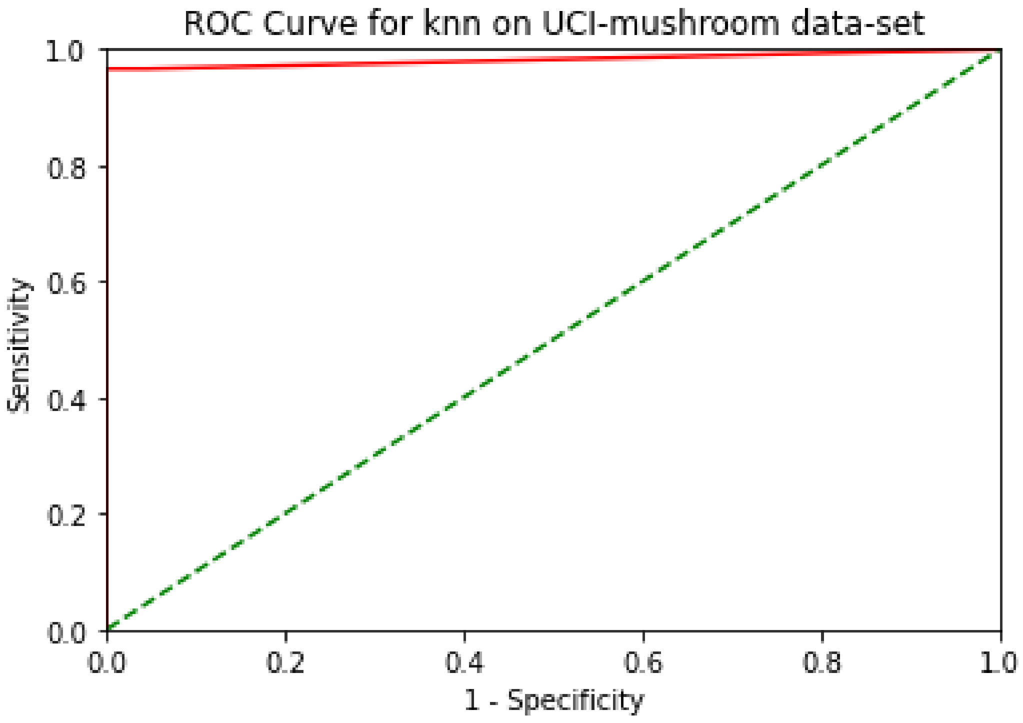
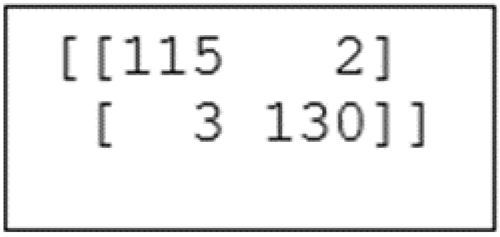


Figure 3.2: AUC = 0.98 (2.d.p)

3.4.3 CART UCI mushroom data results

On the hypothetical features of the UCI dataset with a sample size of 1000, min_stopping_criterion=5, with a training split of 75% train, 25% test, the algorithm achieved the accuracy score of 0.980 (3.s.f), sensitivity=0.975(3.s.f), specificity=0.985(3.s.f); confusion matrix:



```
[[115   2]
 [  3 130]]
```

Figure 3.3: Confusion Matrix achieved when decision tree is used on UCI mushroom data-set

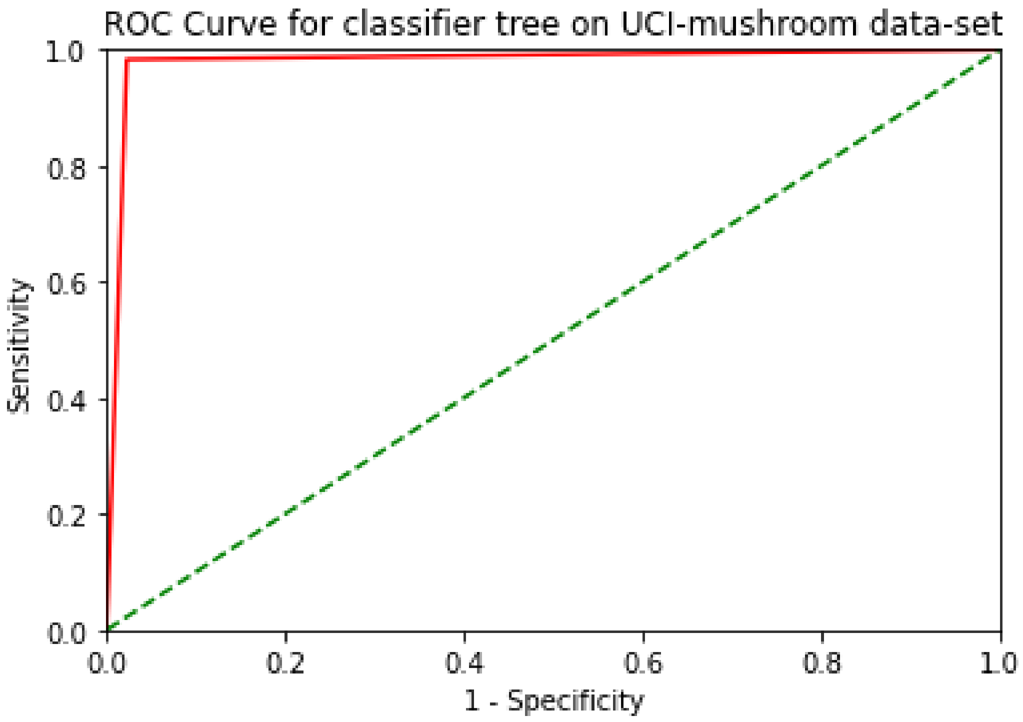


Figure 3.4: AUC = 0.98 (2.d.p)

3.4.4 Comparison of UCI data-set results

Both algorithms achieved very similar accuracy scores with k-nearest neighbors achieving 0.984 and the classification tree scoring 0.980. However, the decision tree produced 3 false negatives compared to k-nearest neighbors which produced 4, and in the context of mushroom edibility classification false negatives can be deadly if acted on (predicting not poisonous when poisonous), therefore in this regard the decision tree performs slightly better however this small difference isn't enough to say in a general case that the decision tree is better at avoiding false negatives. k-nearest neighbors performed slightly better on specificity achieving 1.00 compared to the decision tree which scored 0.985; therefore k-nearest neighbors is better at correctly identifying edible mushrooms. k-nearest neighbors performed slightly worse on specificity achieving 0.966 compared to the decision tree which scored 0.975; therefore the decision tree is better at correctly identifying poisonous mushrooms. Both achieved almost perfect ROC curves, both with AUC of 0.98, meaning both models can almost perfectly distinguish between true and false values.

3.4.5 Comparison of running-time

The decision tree algorithm took longer to train on both data-sets than k-nearest neighbors, however was faster in prediction than k-nearest neighbors. This slow tree building is most likely due to the fact that classifier tree building is a greedy algorithm and also the implementation in python for this product relies very little on numpy features and uses mostly pure python which is a-lot slower.

Chapter 4: **Experiments**

In machine learning there is a "No free lunch" theorem, basically meaning that there is no one best algorithm for every problem. The best performing algorithm changes based on the data-sets used and the parameters passed to the algorithms. This project runs these experiments in an attempt to find the best performing algorithm for mushroom classification problems with a data-set consisting of mushroom images.

The random forest algorithm and support vector classifier will now also be compared as an extension on the previous testing because ensemble methods often improve performance for the decision tree algorithm and support vector machines are interesting conceptually powerful models that have a history of performing well with image classification. Therefore making the algorithms tested: k-nearest neighbors, decision tree, random forest and support vector machine. Comparison will be done of the accuracy, precision, sensitivity and specificity of the algorithms on the binary edibility classification problem. Accuracy scores will be compared for the multi-class species application as well as comparison of the performance of the one against one and one against rest approach to using support vector classifiers for multi-class classification.

4.1 Data pre-processing

4.1.1 Why pre-process the data?

The algorithms that I'm using tend not to perform well on raw pixel data as the features are convoluted and don't hold meaning themselves, it's when a group of pixels is taken that meaningful information is given. Feeding an algorithm like nearest neighbors raw pixel data is like giving a person the constituent pixels of an image and no positional data and asking them to tell you what species they are looking at.

4.1.2 Resizing the images

All images were resized to the size 100 by 100 pixels, this allows the data of the images to be comparable and have the same number of features. What is lost in this process however are the original proportions of the image but this is not a major issue as most of the images in the data-set are of the mushrooms in different angles and rotations.

4.1.3 Removing the background

The background of the mushroom images are noise because the pixel information they give do not have an effect on what species of mushroom is in the image. therefore using the OpenCV python module, the backgrounds of all the images in the data set were removed.

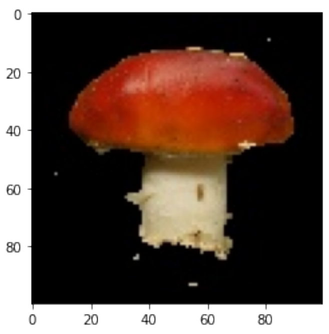


Figure 4.1: Image of Amanita Muscaria

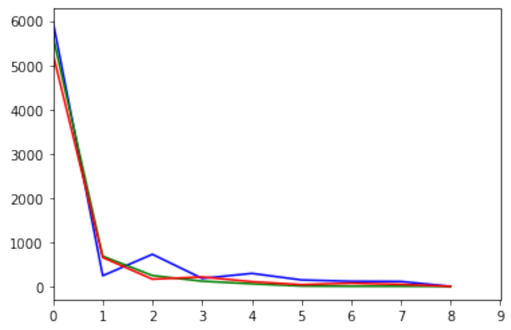


Figure 4.2: Colour histogram of image where x axis is bin number, y axis is pixel count, red line is red intensity the blue line is blue intensity and the green line is green intensity

Figure 4.3: An example of an image and its colour histogram

4.1.4 Creating a colour histogram

This process works by creating bins which each cover a color intensity interval and the bins count up every-time they get a pixel that lands in the bin's intensity interval, there are a set of bins for red intensities, a set for green and a set for blue, the more bins the smoother the graph. What is very useful about the colour information generated by the color histogram is that it does not depend at all on the positioning of the mushroom in the image. Using 8 bins for red, blue and green this process produced 512 features for each sample as $8 \times 8 \times 8 = 512$.

4.1.5 Why not use shape information?

It was attempted to use canny line detection which is an algorithm used for producing a new image with just white lines on a black background in-order to get shape data on resized forty by forty pixel, gray-scale mushroom images. The issue that was had with this data was the non-homogeneity of the angles the images were taken from as you can see in figure 4.8. The wide variety of shapes led to poor performance of the algorithms. If a dataset could be obtained where all images are taken from the same angle then the shape data would have resulted in a useful dataset. It was decided to omit the line generated data from this project as it seemed to hinder the algorithms in preliminary exploratory tests.

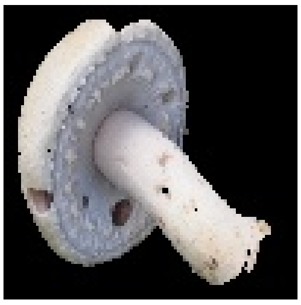


Figure 4.4:

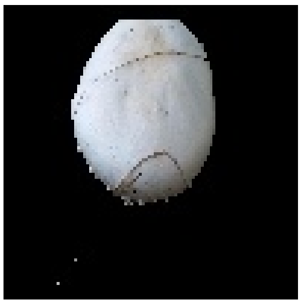


Figure 4.5:

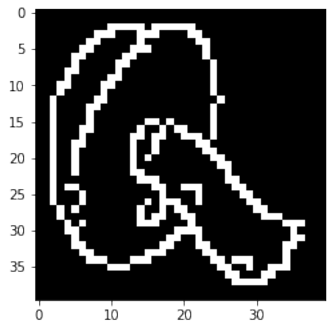


Figure 4.6:

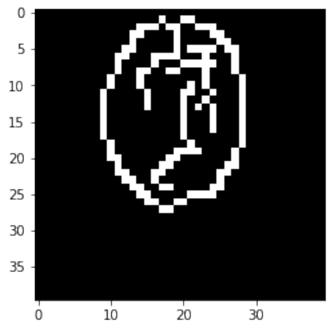


Figure 4.7:

Figure 4.8: Canny line detection run on two different pictures of agricus arvensis, highlighting the problem with the photographed angles.

4.2 Binary Edibility Problem Using Colour Histograms As Features

4.2.1 Overview of this experiment

This experiment is applying k-nearest neighbors, decision tree, random forest and support vector machine classifiers to the colour histogram samples created after pre-processing and binary labels, 0 for edible, 1 for poisonous.

4.2.2 Algorithm Optimisation

To compare the algorithms performance, they will need to be optimised. The optimisation has to be done for algorithms for all experiments carried out and how you do this is to optimize the parameters that the algorithms take in, this can be done by carrying out grid searching. Grid search relies on running the algorithms with all the possible combinations of parameter options that you have provided. This project's version of the algorithms that have been programmed in python unfortunately have extremely long runtimes. This means that this project implementations of the algorithms time taken to run increases dramatically with grid-searching involved too. K-nearest neighbours only has one parameter to optimise and this project's version has good run-time, so this project's implementation of the algorithm was used in optimisation. All other algorithms were optimised using the sci-kit learn implementation of the algorithm. However my implementation of the support vector classifier uses different parameters to the sci-kit learn implementation, therefore I chose a large number for the number of iterations as the model approaches a minimum norm of the weight vector using gradient descent.

Which k is best? To demonstrate that this project has implemented a form of optimisation, parameter optimisation was done with the one algorithm that this project has implemented that it was possible to do so with. The graph in figure 4.9 is the desired shape with the cross-validation curve looking like an upside-down U and the train score sloping up to one. The best k for the best generalised accuracy score will be at the top of the orange curve this is at k is 23 denoted by the red line. Only the training set was used in the cross-validation. The curve shown here is exactly what is expected. The generalisation curve (The cross validation curve) shows that as model complexity increases past the red line then the algorithm becomes over-fit on the training-set.

4.2.3 Selecting k for k-nearest neighbors

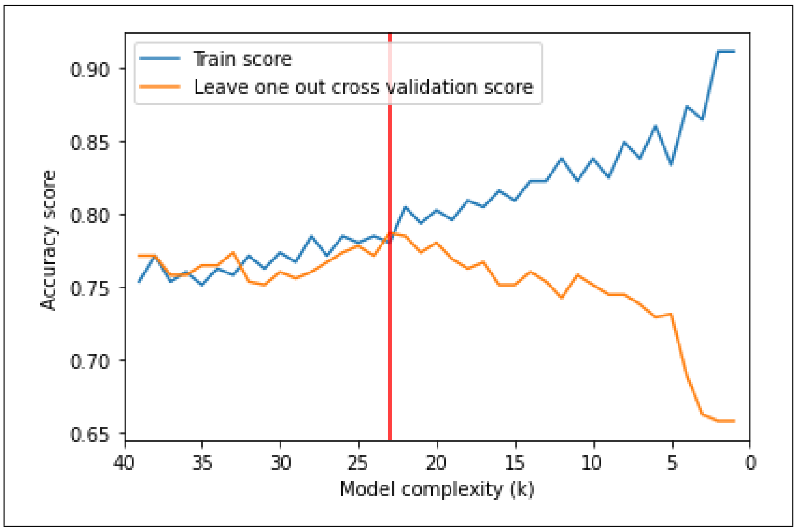


Figure 4.9: Graph showing optimisation for binary edibility problem

4.2.4 Scores

The algorithms were then trained and tested 20 times on the training set comprising of 400 samples randomly selected from the 450 available in the whole training set and tested on 100 randomly selected samples from 150 available samples from the test set. Scores were calculated for all 20 train test combinations and the scores where averaged to give the mean results. The error bars represent the standard deviation of scores achieved by the algorithms.

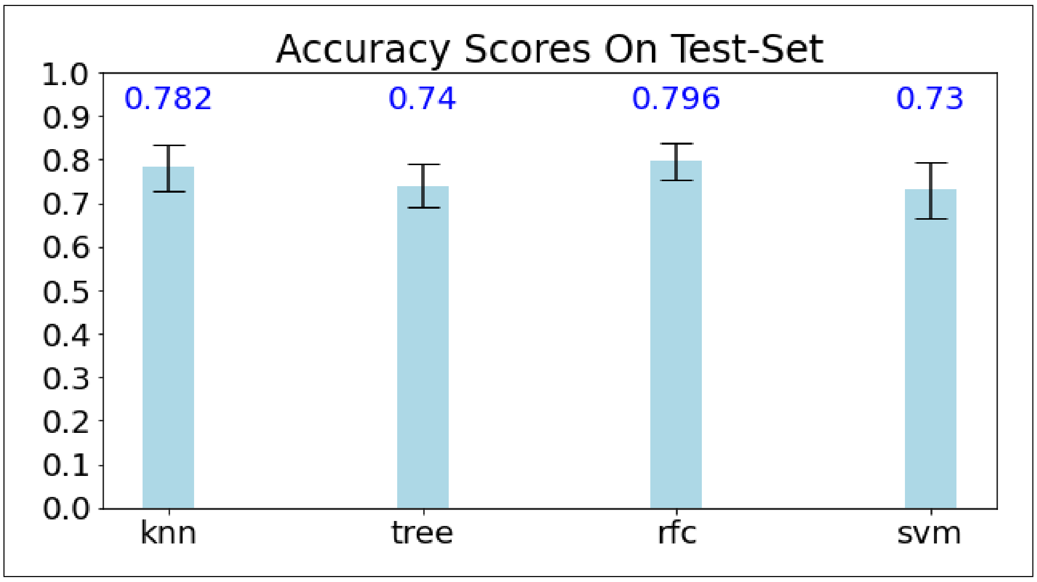


Figure 4.10: Graph showing mean accuracy scores for binary edibility problem. knn:k-nearest neighbors, tree:Decision Tree, rfc:Random Forest, svm:Support Vector Classifier

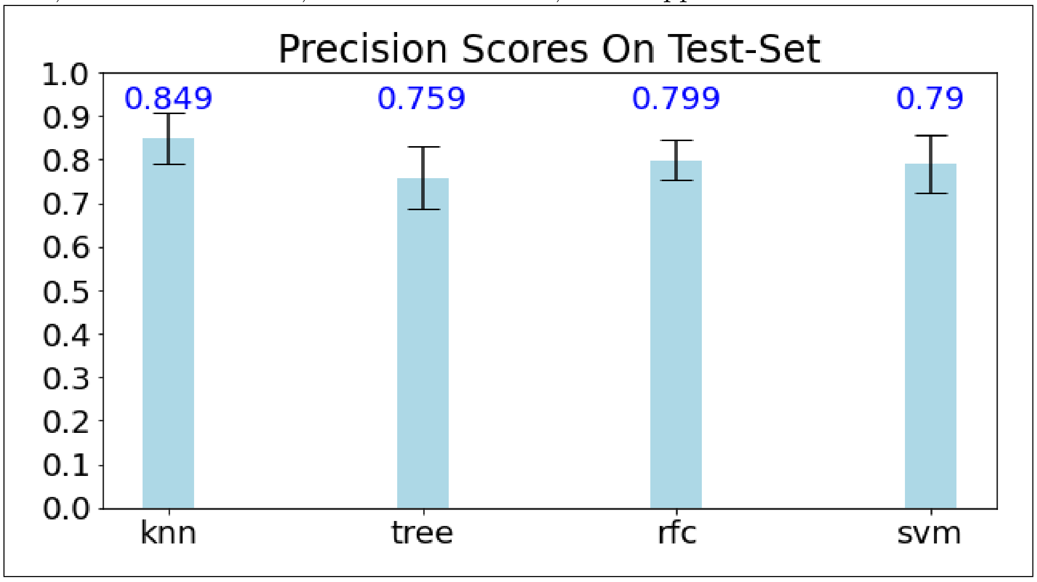


Figure 4.11: Graph showing mean precision scores for binary edibility problem. knn:k-nearest neighbors, tree:Decision Tree, rfc:Random Forest, svm:Support Vector Classifier

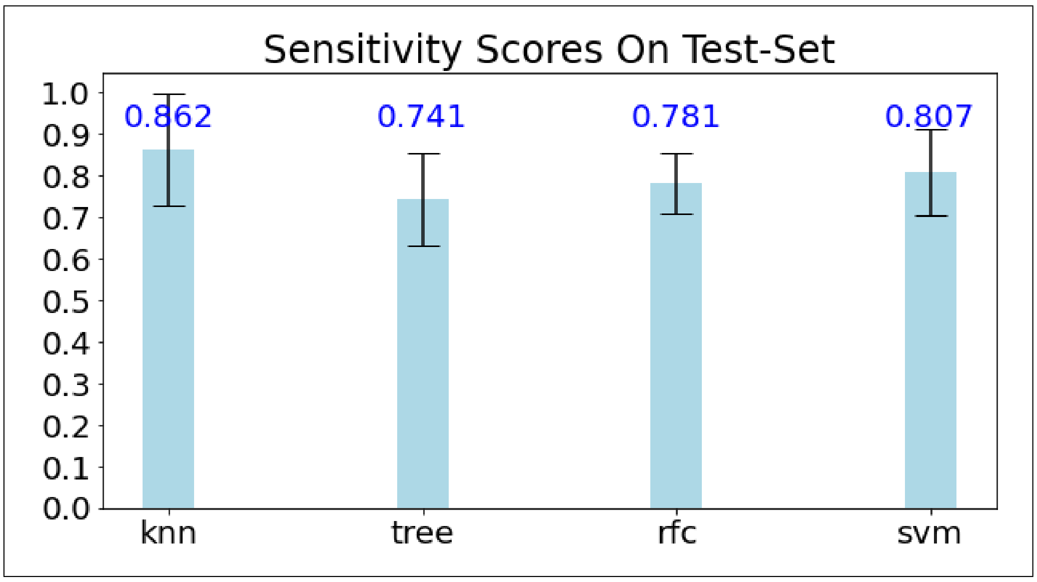


Figure 4.12: Graph showing mean sensitivity scores for binary edibility problem. knn:k-nearest neighbors, tree:Decision Tree, rfc:Random Forest, svm:Support Vector Classifier

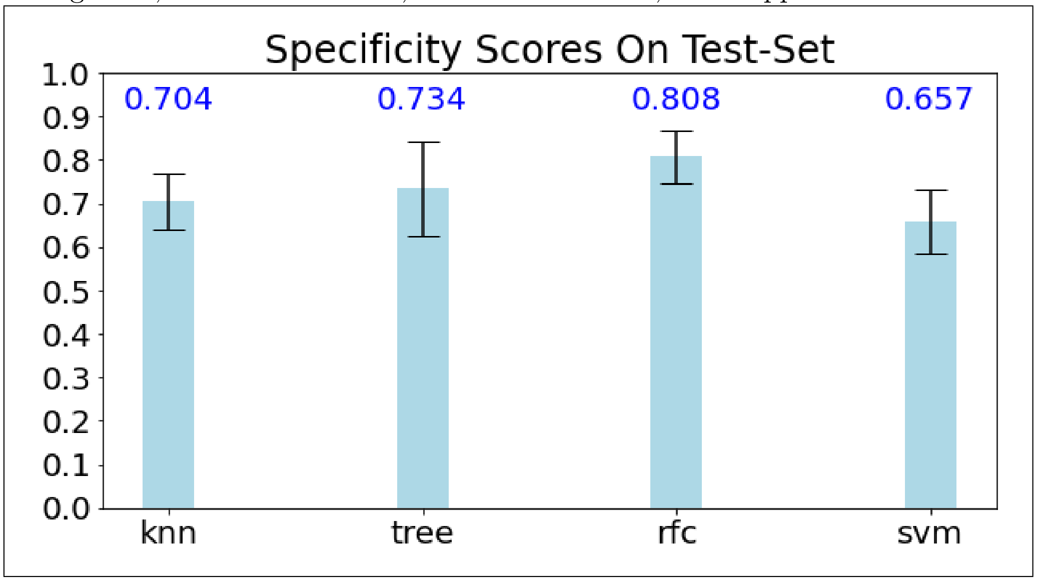


Figure 4.13: Graph showing mean specificity scores for binary edibility problem. knn:k-nearest neighbors, tree:Decision Tree, rfc:Random Forest, svm:Support Vector Classifier

4.2.5 Discussion

Seeing as this problem was a binary classification problem and was distinguishing between two different classes any accuracy score above 0.5 would be better than random guessing. All algorithms scored above this amount.

The support vector machine with linear kernel and the decision tree were very similar in accuracy scores, this is most likely due to the fact that both the algorithms make linear decision boundaries. The optimal parameters given for the decision tree was a very small tree height (a tree height of 2), meaning a not so complex decision boundary. This therefore makes sense that they resulted in very similar modelling as the linear kernel support vector machine gives straight boundaries.

Random forest performed the best in the accuracy measure, scoring 79.6% accuracy on average on the test sets as seen in 4.15. However k-nearest neighbors scored the best in sensitivity with 86.2% which in the context of this project measures the proportion of the poisonous mushrooms that where identified as poisonous. It is very important to identify the poisonous mushrooms as poisonous as it is more harmful to identify a poisonous mushroom as edible.

For these reasons k-nearest neighbors is the best performing algorithm for practical application for this problem as sensitivity is crucial in this safety critical application of machine learning. The paper "Classification of Mushroom Fungi Using Machine Learning Techniques" [20] found that their experimental results showed that the best technique for classifying mushroom images is kNN with accuracy 87% extracted from images only. This is a similar result to the 78% found by this project taking into account that this project uses a different data-set and all accuracy scores for other algorithms were generally around 10% lower than their paper's results for the same algorithms. Their project also used color histograms to extract features.

4.3 Multi-class Species Problem Using Colour Histograms As Features

4.3.1 Overview of this experiment

This experiment is applying k-nearest neighbors, decision tree, random forest and support vector machine classifiers to the colour histogram samples created after pre-processing, the labels being predicted correspond to the six different species of mushroom used in this project.

4.3.2 Algorithm Optimisation

The same methods were used as with the binary edibility problem, exhaustively selecting k, using grid search for the two tree based methods and selecting a large number of iterations for the support vector classifiers with linear kernel, one using the one vs one method and the other using the one vs rest method.

Which k is best? The graph in figure 4.14 is the desired shape with the cross-validation curve looking like an upside-down U and the train score sloping up to one. The best k for the best generalised accuracy score will be at the top of the orange curve, this is at k is 20 denoted by the red line. Only the training set was used in the cross-validation. The curve shown here is exactly what is expected. The generalisation curve (The cross validation curve) shows that as model complexity increases past the red line the algorithm becomes over-fit on the training-set.

4.3.3 Selecting k for k-nearest neighbors

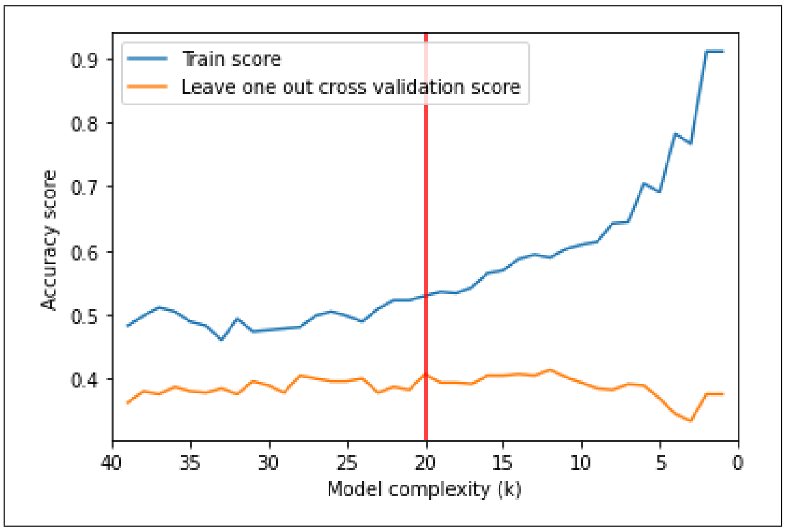


Figure 4.14: Graph showing optimisation for binary edibility problem

4.3.4 Scores

The algorithms were then trained and tested 20 times on the training set comprising of 400 samples randomly selected from the 450 available in the whole training set and tested on 100 randomly selected samples from 150 available samples from the test set. Scores where calculate for all 20 train test combinations and the scores where averaged to give the results. The error bars represent the standard deviation of scores achieved by the algorithms.

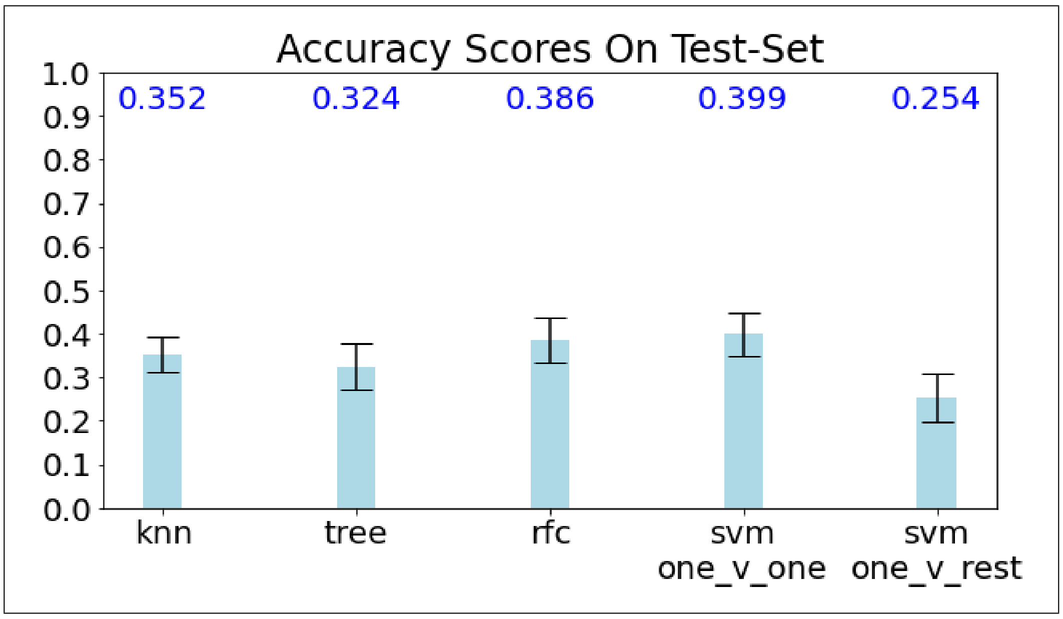


Figure 4.15: Graph showing mean accuracy scores for the six species classification problem. knn:k-nearest neighbors, tree:Decision Tree, rfc:Random Forest, svm oneVSone:Support Vector Classifier using one against one approach, svm oneVSrest:Support Vector Classifier using one against rest approach

4.3.5 Discussion

Seeing as this problem was distinguishing between six different species any accuracy score above 0.167 (3.s.f) would be better than random guessing, all algorithms scored above this amount.

The support vector machine with linear kernel, using the one vs one approach performed the best here with accuracy score of 0.399. What is interesting is that the support vector classifier using the one vs rest approach performed the worst in this experiment with accuracy score of 0.254 therefore showing that one vs one was the best approach for turning the support vector classifier into a multi-class classifier. This may be because there were only six classes which is a relatively small number of classes and if these classes are all quite similar (Which is the case as the mushrooms are quite similar looking). The distinction of a class from the other data needed for the one vs rest procedure may not have been present and the more subtle differences were picked up better by the one vs one procedure.

It is surprising that the support vector machine model did the best in the multi-class species classification as it did not perform very well in the binary problem. The impressive results that convolutional neural net-works produced in the paper "Image Analysis of Mushroom Types Classification by Convolution Neural Net-works"[22] that was 78% accuracy in a multi-class species problem, can help explain why a support vector model out-performed others. The reason the support vector machine model may have out-performed the other models could be because the support vector machine is equivalent to a neural network with one layer.

Chapter 5: **Conclusion**

In conclusion the aim of this project was reached, which was to create and compare machine learning models that can successfully distinguish between images of edible and poisonous mushrooms with good generalizable performance. Getting almost 80% accuracy in the binary edibility problem, is a success for this project, especially when considering that the data-set being used was extremely noisy and wasn't very homogenus.

In respects to previous research on this topic, the findings of "Classification of Mushroom Fungi Using Machine Learning Techniques" [20] with their experiments using colour histograms, (even though they used a different data-set) were very similar, both finding k-nearest neighbours to be the best performing algorithm within the context of mushroom image classification with binary edible or poisonous labels.

The use of colour histograms and colour analysis when it comes to domain specific supervised learning problems, where you know that you have images of objects that are of a certain type, can be a simple way to abstract the problem and reduce dimensionality of the data. With the tandem use of shape information obtained via canny line detection and zoning, this approach can be simple, effective and powerful. It is a shame that the angles that the images of the mushrooms were taken from in the dataset used by this project were not homogenus, otherwise the shape information could have given an extremely descriptive, abstracted idea of what is happening in the image.

When comparing the run-times of the algorithms, decision tree, random forest and support vector machines took a long time to train, however prediction was fast this is because they are eager learning algorithms that do their "learning" in the training stage. k-nearest neighbors however was very fast to train but slower than the other algorithms to predict. In a real deployable model you would rather have predictions be faster than training. In the context of this project as no online learning is being carried out, all training can be done before making the model available for use, this pre-trained model can be saved and used without re-training. Therefore this a caveat to k-nearest neighbors being the chosen algorithm, random forest performed better in accuracy scores so could be a more practical choice for deployment.

Chapter 6: **Reflection**

This project struggled to get results before it was figured out how to pre-process the data properly, this non-trivial pre-processing was very time consuming and often prone to making mistakes, like accidentally turning the whole image into a black background. If this project was attempted again, I think pre-processing should have taken first priority as it was essential for everything else in the project, therefore it acted like a type of blocker, because as soon as the pre-processing used changed, the results became wildy different.

A big issue that was encountered in this project was that the runtime of the algorithms written in python became a limitation as it prevented the number of times the algorithms were run. If it was to be programmed again it would have been better to program the algorithms in C, C++ or java due to these software languages being complied and having faster runtimes. Another option would have been to write the algorithms in Cython which is a compiled language that simulates the syntax of python with C like performance however libraries like numpy are harder to use in the context of a Cython project. The sci-kit learn implementations are all written in C therefore it would have been a better option. This became a larger issue when cross validation multiplied the number of times the algorithms were trained and tested by the number of folds used. Implementing grid search with multiple parameters became impossible because the runtime would often become hours long. This wasn't a problem with k-nearest neighbors as there was only one parameter that was changed; this issue was exaggerated by the fact that image data is very large. This is partly why I didn't implement a convolutional neural network because this would require the algorithm to be trained on raw pixel data. The memory limit of numpy arrays were also encountered when trying to put all of the mushrooms' pixel information into arrays.

Implementing convolutional neural networks to improve performance is motivated by the performance of the algorithms in the paper "Image Analysis of Mushroom Types Classification by Convolution Neural Net-works"[22] where precision for models differentiating between 35 different species was 78%, my models only achieved in 39.9% accuracy as the best result between 6 species. Distributed computing or better computational tools for this project would have been able to improve the results because then using all the image pixel data would have possible in turn making convolutional neural networks a more viable option. Also with more computation available, more species could have been used, therefore making the models much more generalizable.

A further extension of this project could be that within the field of biology a hierarchical mushroom taxonomy tree is used to classify the species of mushroom. A speculation is, if a machine learning decision tree was trained using the predefined structure of the domain specific tree whether or not this would improve results. Having predefined predicates that are able to be applied to the images via a machine learning model, may improve results.

What would also help is if there was a benchmark, relatively homogenus mushroom image data-set that was available for comparing approaches to this problem (Like the UCI mushroom data-set but for images not categorical features) then it would be much easier to compare the results of this project to the results of the wider research being done on this topic. It is not true to the scientific method to not have replicable results. This is a problem for the field of practical machine learning in general, because if someone finds a new unique way of applying techniques and there are no benchmark standard data-sets, the results of these new techniques are not comparable with other techniques.

A further extension to this project can be to try different kernels for the support vector machine, as the relationship between the colour histograms and the labels may not have been a linear relationship. Different kernels for k-nearest neighbours could have also been attempted. The scope of the project didn't allow this and I think it might have confused the goal of this project a bit, however as a further stand alone extension could be insightful.

Chapter 7: **Professional issues**

7.1 Safety

This project highlights the moral dilemma of how much trust to put into an algorithm, if a poisonous mushroom is classified as edible by this projects model and someone acts on this information and ingests the mushroom there could be fatal consequences. If this algorithm could perform with higher accuracy than a foraging expert, then maybe there could be grounds to using the model instead. However, since there are features that cannot be extracted from an image, such as spore colour, smell and size, there is almost no chance that this algorithm will perform better than a human expert. Meaning it needs to be disclaimed to not act on the output of this statistical model. This model can be used as a tool to aid in classification however should be checked by a mushroom foraging expert. A real world example in a different domain is the introduction of driverless cars, and at what point is a driverless car safer than a human driving, is it when it has less accidents? less fatalities? the driverless car makes moral decisions? The current rule in United States of America is that a driverless car can be in full control of the vehicle if there is a person at the wheel of the car ready to override the system. I think the same approach is necessary with any safety critical machine learning application, to have an override and a “expert” human making the final call. The United Kingdom government has started consultation on allowing autonomous driving, therefore it seems likely that the approach of safety critical machine learning model use will become widespread.

7.2 Licensing and use of dataset

This project uses image data from the Danish Svampe Atlas and these rules are stated on the github download page for the dataset:

”

1. You will abide by the Danish Svampe Atlas Terms of Service
2. You will use the data only for non-commercial research and educational purposes.
3. You will NOT distribute the above images.
4. The Danish Svampe Atlas makes no representations or warranties regarding the data, including but not limited to warranties of non-infringement or fitness for a particular purpose.
5. You accept full responsibility for your use of the data and shall defend and indemnify the Danish Svampe Atlas, including its employees, officers and agents, against any and all claims arising from your use of the data, including but not limited to your use of any copies of copyrighted images that you may create from the data.

” [26]
therefore during this project cannot infringe on these rules otherwise it will be breaking the terms of use of the data the Danish Svampe Atlas requires. therefore there will be no commercial applications of these models and full responsibility will be taken for any claims

arising from the use of this data, in the project. There were more open-source options for mushroom images that didn't restrict commercial applications such as the images from mushroomworld.com, however they did not have anywhere close to the number of images available from the Danish Svampe Atlas, therefore the decision was made to go with this dataset of images and except the Danish Svampe Atlas Terms of Service. The Danish Svampe Atlas Terms of Service is written entirely in danish and google translate was used to understand the text, therefore it might be a good idea to get someone who can read danish to check I have understood the terms correctly.

7.3 Plagiarism

Plagiarism is the act of taking someone's ideas and passing them off as your own, therefore here I will state some examples where it could have been plagiarism if it was not made clear that the work was not mine. This project uses quite a few python modules which counts as code that has not been produced for this project, therefore every python module that is used must be referenced to give acknowledgement of the use of others code. This project's implementation of the decision tree algorithm was heavily inspired by the work of Jason Brownlee, therefore it is stated at the top of the decisiontree.py, as it is implemented with python dictionaries much like the aforementioned implementation. If this was not stated, then this would be plagiarism. It is very important to carry out correct citation because otherwise you could be carrying out accidental plagiarism is you fail to cite your sources, therefore I created a References.bib file where I've keep all references as I've gone along and can cite anything I've read in relation to the project in my report.

7.4 Management

An issue with time management of the project is at the start of the project it was not foreseen how time-consuming the data-pre-processing stage of the images was going to be, it was thought this step was going to be relatively straight forward however this did not turn out to be the case, this in turn was exaggerated by having lots of outliers in my data set that had to be discarded by eye, such as a picture of just a hand or of the spores of the mushroom under a microscope. There should have been more time allocated in the plan to data pre-processing.

7.5 Privacy and data-protection

Data protection means keeping safe the data about someone that they do not want in the public domain. In some of the images of mushrooms there are also people standing and holding the mushrooms and the person is clearly visible in frame. The person who uploads the image to the Danish Svampe Atlas gives consent for the image to be used but the person in the image may not be the same person as who is uploading and the person in the image may not be aware that an image of them is being used in this dataset without them having given consent. There are data privacy concerns related to this. Due to concerns any images containing other people displayed in them were removed from the dataset that was used for this project.

Chapter 8: **Diary**

October 9, 2020
I have been preparing the project format to push to SVN and trying to set up connection with pycharm, and doing the correct installs to allow subversion compatibility, I have also started work on my data report, where I go in-depth and study my data-set and pull key information from it and attempt to visualize it.

October 10, 2020
In my meeting with my supervisor we went over my draft project plan and I changed my project idea (to create an application to apply any machine learning algorithm you wanted to a data-set and to see the results) as my supervisor and I agreed it was a bit general. Therefore, I changed my project to be specifically about comparing algorithms for mushroom edibility classification and decided to use Jupyter Notebooks to code in and to display my findings.

October 18, 2020
I created a notebook to clean my data, encode the labels, standardize, and save the resulting data-set. I also programmed the nearest neighbour algorithm.

October 25, 2020
I programmed the K-nearest-neighbour algorithm and created a test folder for test driven development. I also converted the jupyter notebook code of nn-knn into a python module called nearestneighbor.py and created a folder, algorithms where all the python versions of the algorithms I program will reside.

October 31, 2020
I created my nearest neighbour report as a .tex file and completed 3 sections of this, the introduction, the description of k-nearest neighbour and the description of nearest-neighbour. I also downloaded a large image data-set of mushroom images , due to the fact that the data-set file was about 12GB I decided not to upload this to the repository. In my meeting with my supervisor we decided that my algorithms should be self optimizing for parameters, and the final product will be a comparison of an optimised k-nearest neighbours algorithm against an optimised decision tree. We also decided that the data-cleaning report was of little relevance and to focus more on the machine learning.

November 8, 2020
Collected resources for finishing k-nearest neighbors report, made progress on processing image data-set that is too large to be kept on the repository.

November 15, 2020
Implemented k-nearest neighbor on uci mushroom repository, achieved 100% used sci kit learn implementation in areas as I haven't programmed my cross validation function yet so put the sci-kit learn classifier temporarily in place in sections, this will be replaced.

November 21, 2020
Included sections in my knn report of who came up with the algorithms and explained different distance metrics. I programmed my own implementation of cross validation, removed sci-kit learn implementation from knn notebook.

November 29, 2020
Added decision tree section in report, programmed a confusion matrix class, programmed the scoring for the decision tree in decision-tree notebook generated all graphs and metrics

for report .

January 17, 2021
This week worked on planning out what I was going to pursue this term, finding resources about neural networks and image feature extraction.

January 24, 2021
Added a python notebook on image-feature extraction in-order to get meaningful features extracted from my image data-set

January 31, 2021
Removed background of mushroom images created file format for multi-class classification.

February 7, 2021
Created an image data set of line only mushroom images using cannyline detection and using zoning created a data set of 16 features and created a data set than is produced from colour analysis of the images.

February 14, 2021
Added both colour data and shape data into one data-set ready for the comparisons of performance of the algorithms achieving approximately 75% accuracy for both nearest neighbors and decision tree.

February 22, 2021
I added final touches to my draft report to submit, programmed an implementation of a random forest algorithm. Started work on programming a support vector machine.

February 28, 2021
Finished implementation of support vector classifier and tests associated.

March 8, 2021
Generated comparison graphs for the binary classification problem and added section on the uci mushroom data back into report to be the tests for the algorithms.

March 15, 2021
Added results into the final report added svm section about multiclass classification. Added multiple runs to testing and average of the scores calculated.

March 20, 2021
Put results from notebook into the final report, finishing write up, doing conclusion and reflection parts of my report.

Chapter 9: **Bibliography**

[1] Atul Agarwal. "Support Vector Machine — Formulation and Derivation". In: *https://towardsdatascienc vector-machine-formulation-and-derivation-b146ce89f28* (Sept. 2019).

[2] William P Arend et al. "The American College of Rheumatology 1990 criteria for the classification of Takayasu arteritis". In: *Arthritis & Rheumatism* 33.8 (1990), pp. 1129–1134.

[3] Sidath Asiri. "Support Vector Machine — Formulation and Derivation". In: *https://towardsdatascience learning-classifiers-a5cc4e1b0623* (June 2018).

[4] bhiksha. "Building decision trees". In: *https://www.cs.cmu.edu/ bhiksha/courses/10-601/decisiontrees/* (2021).

[5] L Breiman. "Random Forests". In: *Machine Learning* (2001).

[6] Huy Bui. "ROC Curve Transforms the Way We Look at a Classification Problem". In: *https://towardsdatascience.com/a-simple-explanation-of-the-roc-curve-and-auc-64db32d75541* (2020).

[7] Nicola J Camp and Martha L Slattery. "Classification tree analysis: a statistical tool to investigate risk factor interactions with an example for colon cancer (United States)". In: *Cancer Causes & Control* 13.9 (2002), pp. 813–823.

[8] Metz CE. "Basic principles of ROC analysis". In: *Semin Nucl Med* (1978).

[9] Mohamed Amine Chikh, Meryem Saidi, and Nesma Settouti. "Diagnosis of diabetes diseases using an Artificial Immune Recognition System2 (AIRS2) with fuzzy K-nearest neighbor". In: *Journal of medical systems* 36.5 (2012), pp. 2721–2729.

[10] T. Cover and P. Hart. "Nearest neighbor pattern classification". In: *Information Theory, IEEE Transactions on* (1967).

[11] Nello Cristianini and John Shawe-Taylor. "Support Vector Machines". In: *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000, pp. 93–124. DOI: 10.1017/CBO9780511801389.008.

[12] Judith A Falconer et al. "Predicting stroke inpatient rehabilitation outcome using a classification tree approach". In: *Archives of Physical Medicine and Rehabilitation* 75.6 (1994), pp. 619–625.

[13] Elima Hussain, Abdul Hannan, and Kishore Kashyap. "A Zoning based Feature Extraction method for Recognition of Handwritten Assamese Characters". In: *www.ijcst.com* 6 (Jan. 2015).

[14] Yerushalmy J. "Statistical problems in assessing methods of medical diagnosis, with special reference to X-ray techniques". In: *SPublic Health Rep* (1947).

[15] Reinhard Klette. "Concise Computer Vision". In: *ISBN 978-1-4471-6320-6* (2014).

[16] George C. Stockman Linda G. Shapiro. "Computer Vision". In: *ISBN 978-0-13-030796-5* (2001).

[17] Hucker Marius. "Multiclass Classification with Support Vector Machines (SVM), Dual Problem and Kernel Functions". In: *https://towardsdatascience.com/multiclass-classification-with-support-vector-machines-svm-kernel-trick-kernel-functions-f9d5377d6f02* (June 2020).

[18] Tim Morris. "Computer Vision and Image Processing". In: *ISBN 978-0-333-99451-1* (2004).

[19] Anuja Nagpal. "Decision Tree Ensembles Bagging and Boosting". In: *towardsdata-science.com* (2020).

[20] Mohammad Ashraf Ottom. "Classification of Mushroom Fungi Using Machine Learning Techniques". In: *International Journal of Advanced Trends in Computer Science and Engineering* 8 (Oct. 2019), pp. 2378–2385. DOI: 10.30534/ijatcse/2019/78852019.

[21] B. S. Patowary. “”Mushroom Poisoning-an overview””. In: *Journal of college of Medical Sciences-Nepal, vol. 6* (2010), pp. 56–61.

[22] Jitdumrong Preechasuk et al. “Image Analysis of Mushroom Types Classification by Convolution Neural Networks”. In: Dec. 2019, pp. 82–88. DOI: `10.1145/3375959.3375982`.

[23] Lior Rokach and Oded Maimon. “Top–Down Induction of Decision Trees Classifiers–A survey”. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* (2005), p. 487.

[24] H Sabrol and K Satish. “Tomato plant disease classification in digital images using classification tree”. In: *2016 International Conference on Communication and Signal Processing (ICCSP)*. IEEE. 2016, pp. 1242–1246.

[25] Manish Sarkar and Tze-Yun Leong. “Application of K-nearest neighbors algorithm on breast cancer diagnosis problem.” In: *Proceedings of the AMIA Symposium*. American Medical Informatics Association. 2000, p. 759.

[26] Brigit Schroeder. “fgvcx$_f$*ungi*$_c$*omp*”. In: *https://github.com/visipedia/fgvcx*$_f$*ungi*$_c$*omp* (2021).

[27] Hendrana Tjahjadi and Kalamullah Ramli. “Noninvasive Blood Pressure Classification Based on Photoplethysmography Using K-Nearest Neighbors Algorithm: A Feasibility Study”. In: *Information* 11.2 (2020), p. 93.

[28] Robert Tibshirani Trevor Hastie and Jerome Friedman. *The Elements of Statistical Learning.* Second edition. Springer, New York, 2009.

[29] Vispedia. “Danish Svampe Atlas”. In: *https://github.com/visipedia/fgvcx_fungi_comp* (2018).

[30] Vladimir Vovk. “Neural networks and support vector machines”. In: *CS3920/CS4920/CS5920 Machine Learning* (Nov. 2020).

[31] Eric W Weisstein. “Vector Norm”. In: *mathworld.wolfram.com* (2020).

[32] Hall Mark Witten Ian Frank Eibe. “Data Mining”. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* (2011), pp. 101–103.

[33] Xueyan Wu, Jiquan Yang, and Shuihua Wang. “Tea category identification based on optimal wavelet entropy and weighted k-Nearest Neighbors algorithm”. In: *Multimedia Tools and Applications* 77.3 (2018), pp. 3745–3759.

[34] Qi Yu et al. “Optimal pruned K-nearest neighbors: OP-KNN application to financial modeling”. In: *2008 Eighth International Conference on Hybrid Intelligent Systems.* IEEE. 2008, pp. 764–769.

[35] Junni L Zhang and Wolfgang K Härdle. “The Bayesian additive classification tree applied to credit risk modelling”. In: *Computational Statistics & Data Analysis* 54.5 (2010), pp. 1197–1205.