Interim Report: Machine learning for classifying mushroom edibility

ZFAC037

Abstract What this project is going to achieve is to program and implement classification algorithms to test on the UCI mushroom data-set to predict, based on twenty-two categorical attributes, whether or not a mushroom species is edible or poisonous (a binary classification problem) and to produce key results metrics that are required for the comparision of the algorithms. The UCI mushroom data-set consists of information about 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Another goal will be to use an image data-set of mushrooms for this problem as well and compare the performance of the algorithms on image input and on categorical input.

Contents

1 Introduction

1.1 Aims and Objectives

Aim: To implement and compare algorithms for classifying mushroom edibility, 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.

Objectives:

1. Program k-nearest neighbor algorithm.
2. Program a classification decision tree.
3. Applying the algorithms on the hypothetical mushroom feature data (UCI mushroom repository).
4. Comparing the two algorithms on being able to classify edibility of mushroom images.
5. Getting accuracy scores from applications of both algorithms on both data-sets.
6. Generating confusion matrices to display the binary classification results.
7. Generating ROC curves for both algorithms application to both sets.
8. Comparing the results and discussing how best to improve them.

1.2 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.

1.3 Image Data-set used

The mushroom images used are sourced from the Danish Svampe Atlas [16] and split into two categories, edible and poisonous, only Agricus and Lepiota type mushrooms were used, 82 samples split half poisonous, half edible. Why this data-set was chosen is due to the fact that the data-set is relatively small and because image data contains a large number of features if too many images were selected then the algorithms run-times would be very high. When turned into .csv file format all images were compressed to 300 pixel size.

1.4 k-nearest neighbors

Introduction to 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"[5] for use in classification and regression.

1.5 CART

Introduction to 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." [15]. The term CART was created by Leo Breiman to name decision tree algorithms that can be used for regression or classification problems.

1.6 Literature survey

In the paper "Classification of Mushroom Fungi Using Machine Learning Techniques" [8] 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 project tests the findings of this paper that kNN is the best algorithm as this project will implement many algorithms and optimize them and the goal is to achieve better than 94% on a mushroom edibility image classification problem. Using the UCI Mushroom data-set as synthetic features extracted from an image to train the model and then test the model of features extracted from an image. Using the image data-set to see what can be achieved with raw pixel data.

Examples of applications of *k*-nearest neighbors

- Breast cancer diagnosis [11]
- Tea category identification on optimal wavelet entropy [19]
- Noninvasive blood pressure classification [14]
- Diabetes classification [4]
- Financial modeling [20]

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

Examples of applications of CART

- A statistical tool to investigate risk factor interactions with an example for colon cancer [3]
- Predicting stroke inpatient rehabilitation outcome [6]
- Tomato plant disease classification in digital images. [10]
- Credit risk modelling [21]
- classification of Takayasu arteritis (inflammation of the arteries) [1]

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 [10] is a standout example as it involves image classification.

2 Theory

2.1 Nearest neighbors

2.1.1 Different measures of closeness

Where $a = 1^{st}$ vector, $b = 2^{nd}$ vector, 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:

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:

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

L1 and L2 norms are p-norms where $p = 1$ and $p = 2$ respectively. All the norms can be expressed as dot products, as 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:

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

[17]

Neighbourhood function $N_k(x')$ 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:

$$D_i = dist(x', X_i)$$

$$N_0 = \{\}$$

$$N_{i+1} = N_i \cup \{min(D/N_i)\}$$

2.1.2 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:

$$\widehat{Y}(x') = mode(y_i | x_i \epsilon N_k(x')), \qquad [15]$$

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.

2.1.3 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: $\widehat{Y}(x') = y_i | x_i \epsilon N_1(x')$

2.1.4 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.

2.1.5 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 random 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.

2.1.6 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 gets higher the larger the training set becomes.

- **A Simple Approach:** $k = n^{1/2}$ where n is training set size
- **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 achieves the highest cross validation score, this doesn't involve data snooping as only the training set is used.

2.1.7 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 weakness is that building the model is fast however 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 and working out distances the algorithm is very memory intensive compared to other algorithms.

2.2 Decision tree

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.

2.2.1 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. [9]

2.2.2 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:

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

2.2.3 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.

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

[18]

2.2.4 Classification decision-tree in pseudo-code

1. " Place the best attribute of the dataset at the root of the tree.
2. Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree." [12]

2.2.5 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.

2.2.6 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." [12]

2.2.7 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.[15]

- **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.[15]

- **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.[15]

- **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.

2.2.8 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. [7]

- **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. [7]

- **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. [7]

- **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. [7]

2.3 Scoring Metrics

2.3.1 Cross validation

Definition 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, when $K = m$ this is called "leave one out cross validation".

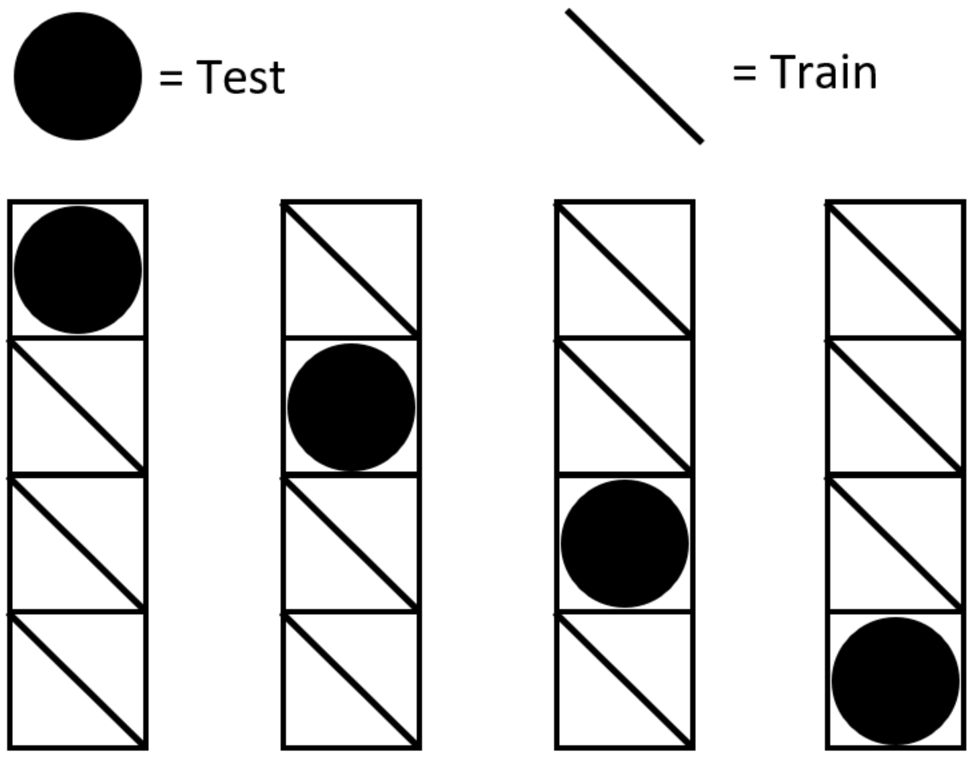
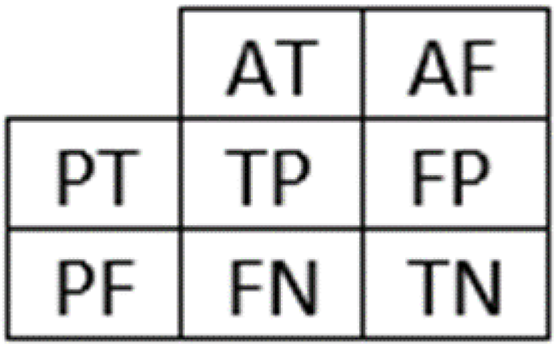


Figure 1: Four fold cross validation

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. Also can be used for parameter selection as the best performing parameter in cross validation will often be the best generalised parameter.

2.3.2 Confusion Matrices

Matrix to represent the number of true positives(TP), true negatives(TN), false positives(FP) and false negatives(FN). Below 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: Binary confusion matrix

2.3.3 ROC curve

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.

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

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

[2]

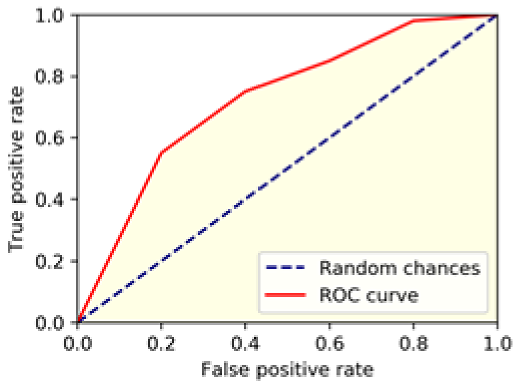


Figure 3: Example of a ROC curve

2.3.4 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.

3 Implementation

3.1 How to run the code

To see all graphs and scoring metrics obtained run the nn-knn.ipynb and decision-tree.ipynb notebooks.

3.2 Uci data-set information

"This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like “leaflets three, let it be” for Poisonous Oak and Ivy.”

- Data Set Characteristics: Multivariate
- Number of Instances: 8124
- Area: Life
- Attribute Characteristics: Categorical
- Number of Attributes: 22

- Date Donated: 1987-04-27
- Associated Tasks: Classification
- Origin: Mushroom records drawn from The Audubon
- Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf 3.5.2 Attributes of the data-set Attribute Information: "

[13]

3.3 Image data-set

Comprises of one set with 41 images of Agricus mushrooms labelled edible and 41 images of Lepiota mushrooms labelled poisonous.

3.4 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.

3.5 Modules and Notebooks

3.5.1 UML

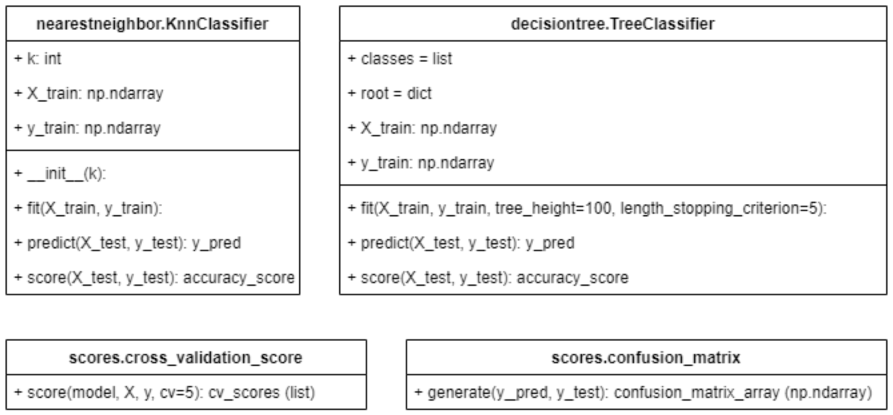

nearestneighbor.KnnClassifier
+ k: int
+ X_train: np.ndarray
+ y_train: np.ndarray
+ __init__(k):
+ fit(X_train, y_train):
+ predict(X_test, y_test): y_pred
+ score(X_test, y_test): accuracy_score
decisiontree.TreeClassifier
+ classes = list
+ root = dict
+ X_train: np.ndarray
+ y_train: np.ndarray
+ fit(X_train, y_train, tree_height=100, length_stopping_criterion=5):
+ predict(X_test, y_test): y_pred
+ score(X_test, y_test): accuracy_score
scores.cross_validation_score
+ score(model, X, y, cv=5): cv_scores (list)
scores.confusion_matrix
+ generate(y_pred, y_test): confusion_matrix_array (np.ndarray)

Figure 4: UML of classes in this project

3.5.2 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 index then trains and tests on selected folds in turn and returns an array of scores from the cross validation. 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.

"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 produces 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 also contains the fit, predict, and score methods, however this module does not work for multi-class classification as it was not needed to be implemented.

"notebooks/data-visualization-cleaning" notebook This notebook is used to load the image data from the image_data files and assigns the correct binary label depending on which file the sample came from (edible=0 or poisonous=1) also the images are compressed down from their original size down to (100,100) resolution which then gives 300 features for the data-set, and it then saves the files to mushroom_image_data.csv for the pixel-data samples and the corresponding labels are saved in mushroom_image_target.csv.

"notebooks/nn-knn" This notebook is used to train and test the k-nearest neighbor algorithm and to generate all graphs and performance results shown in this report.

"notebooks/decision-tree" This notebook is used to train and test the decision tree classification algorithm and to generate all graphs and performance results shown in this report.

3.6 Data normalisation

It is import 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 Normalizer was chosen as the Scaler of choice on the UCI mushroom repository due to the fact that both the train and the test set can be normalised at together as data-snooping does not occur due to each sample being normalised individually. The data was saved in pre-processed-uci-dataset folder .
The StandardScaler was chosen as the Scaler of choice on the image-data due to the variance controlling nature.

3.7 Testing nearest neighbor module functionality

There is a python notebook to test the results of the project's version of k-nearest neighbors against sci-kit learns implementation and although the results were similar on sci-kit learns iris data-set (Chosen as flower classification is similar to mushroom classification) there are small differences that are believed to come from different methods for tie breaking as the iris classification problem is a multi-class classification. However tie-breaking will not be a large issue on application as this is a binary classification problem, therefore if an odd number for k is chosen then there will not be ties.

3.8 Testing classification tree module functionality

There is a python notebook to test this project's version of a classification tree to make sure it operates as expected, this time on the breast cancer data-set from the sci-kit learn package as my classification tree was only programmed to work with binary classification problems.

3.9 Parameter selection

3.9.1 Choosing k for k-nearest neighbors on image data set

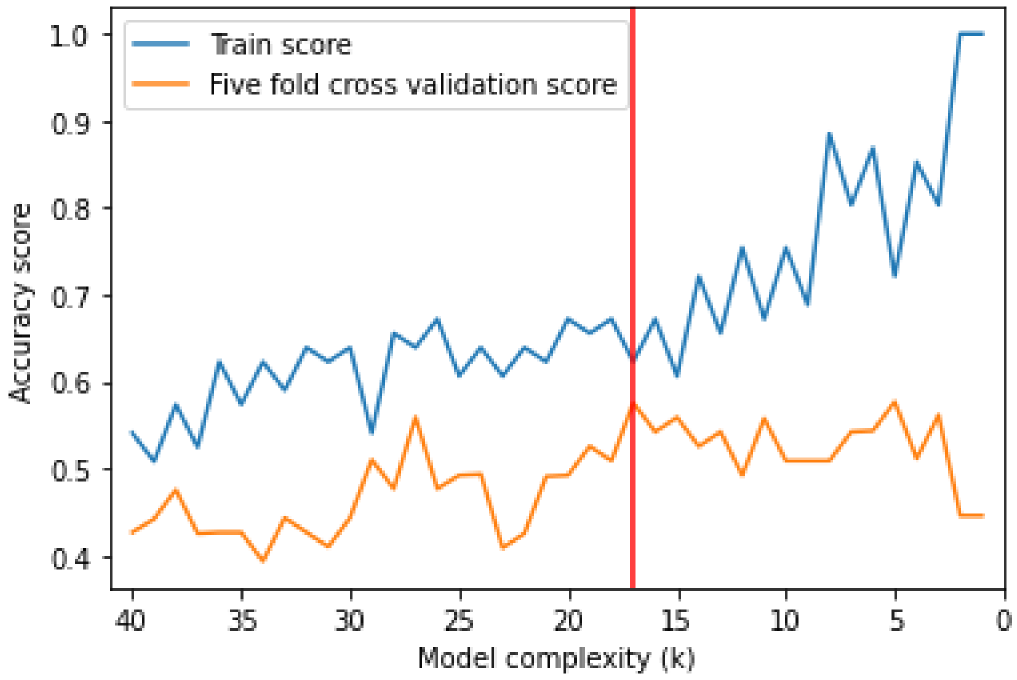


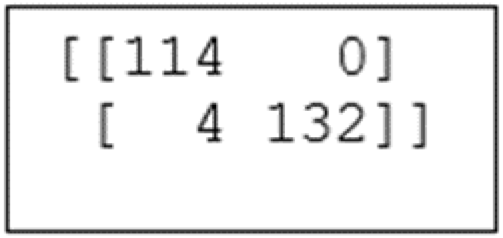
Figure 5: Knn applied to image data-set

Which k is best? This graph 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 17 denoted by the red line. Only the training set was used in the cross-validation.

4 Results

4.1 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 6: Confusion Matrix achieved when k-nearest neighbors is used on UCI mushroom data-set

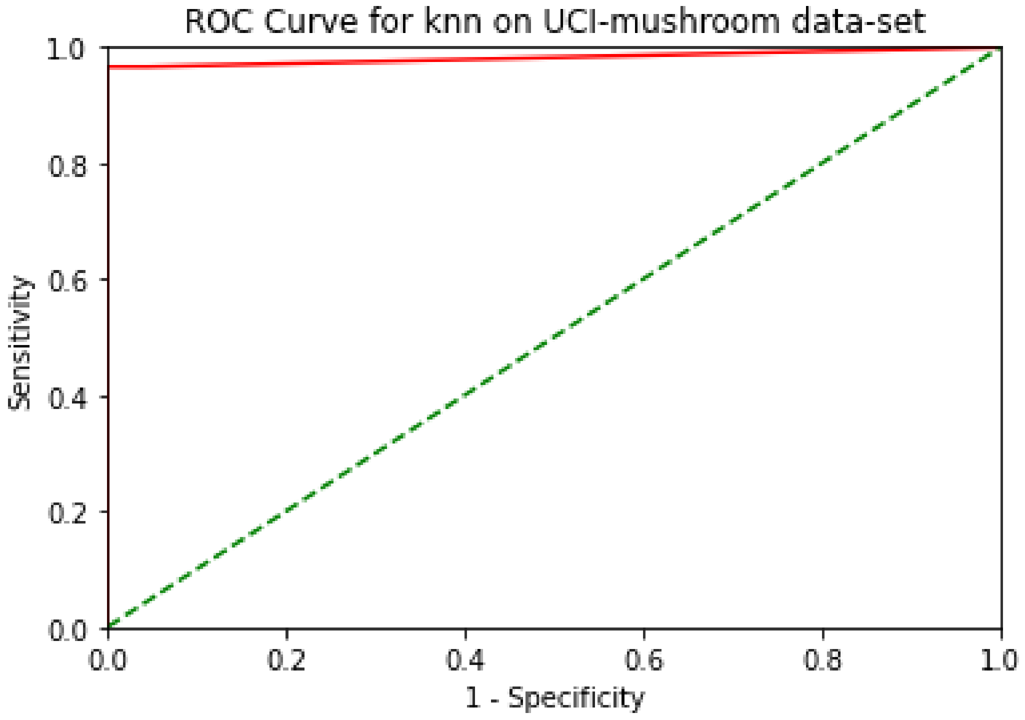
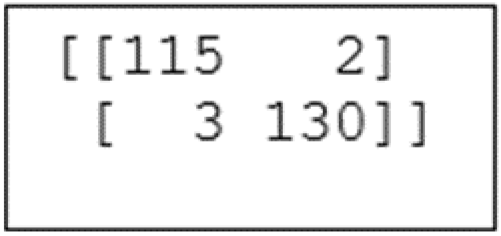


Figure 7: AUC = 0.98 (2.d.p)

4.2 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 8: Confusion Matrix achieved when decision tree is used on UCI mushroom data-set

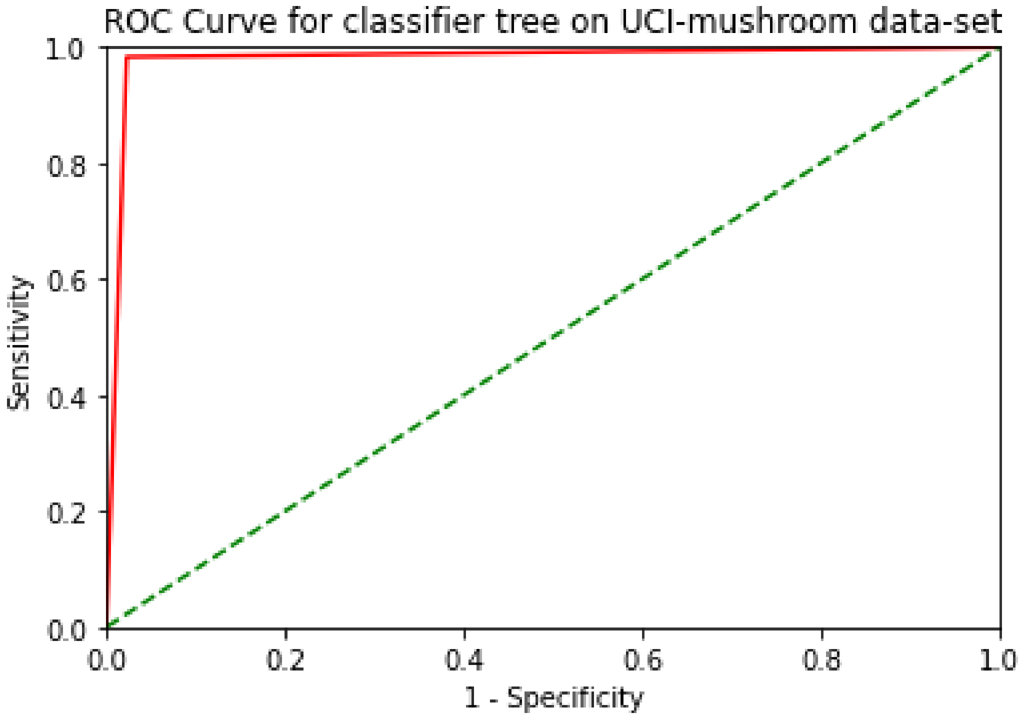
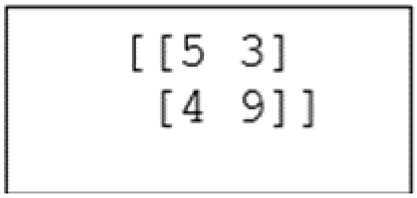


Figure 9: AUC = 0.98 (2.d.p)

4.3 kNN image data results

On the image data-set of 82 images, 300 raw pixel data features each, with a training split of 75% train, 25% test, using $k = 17$, using euclidean distance, the algorithm achieved the accuracy score of 0.667 (3.s.f) , sensitivity=0.556 (3.s.f), specificity=0.750 (3.s.f). confusion matrix:



```
[[5 3]
 [4 9]]
```

Figure 10: Confusion Matrix achieved when knn is used on image mushroom data

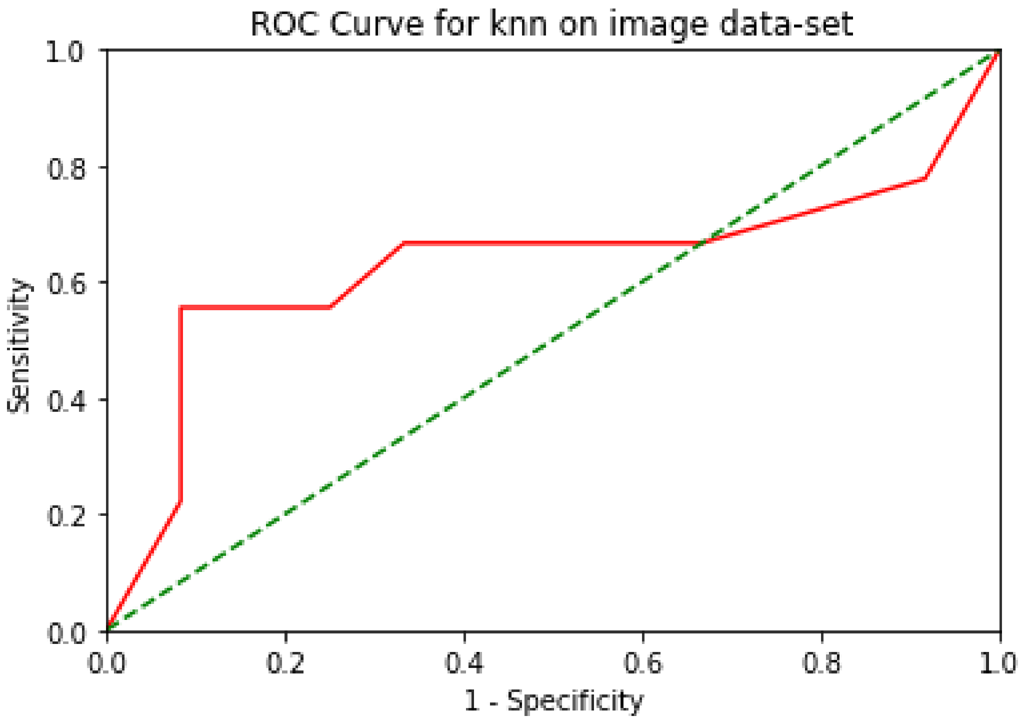
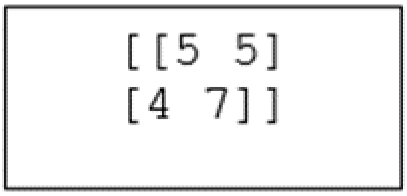


Figure 11: AUC = 0.63 (2.d.p)

4.4 CART image data results

On the image data-set of 82 images, 300 raw pixel data features each, with a training split of 75% train, 25% test, using stopping-criterion of minimum elements in leaf node as 5 , using max-depth of 100, the algorithm achieved the accuracy score of 0.571 (to 3.s.f) producing the confusion matrix in figure 6. sensitivity=0.556 (3.s.f), specificity=0.583.



```
[[5 5]
 [4 7]]
```

Figure 12: Confusion Matrix achieved when classifier tree is used on image mushroom data

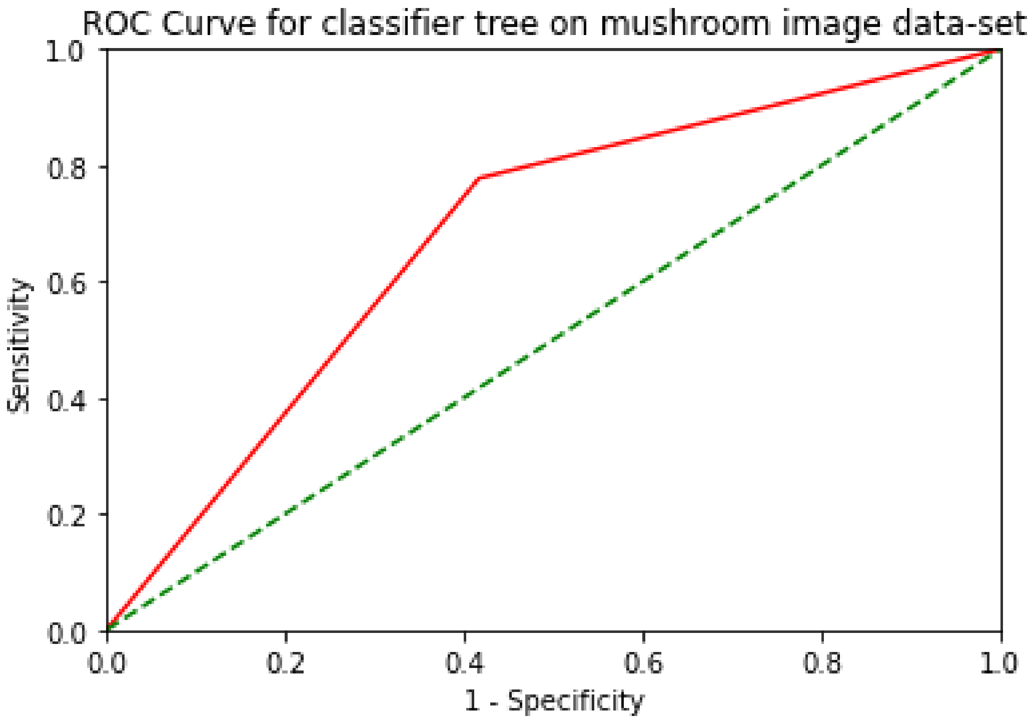


Figure 13: AUC = 0.68 (2.d.p)

4.5 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.

4.6 Comparison of image data results

Both algorithms scored extremely poorly in all metrics on the image data with most specificity and sensitivity results only being slightly above 0.5, where 0.5 would be random guessing, this is most likely due to the fact pure image data is extremely noisy and without using feature extraction the algorithms are given a large amount of useless data. Also the small size of the data-set meant that outliers affected results more than if the data-set was larger.

4.7 Comparison of results from the two data-sets

Really what the results show is a case of extremely good results for the hypothetical feature data, and extremely bad results with the raw image pixel data with both models. What this means is that if meaningful features can be extracted from the image data both algorithms used should be able to produce highly accurate models.

4.8 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.

5 Conclusion

Both algorithms performed very similarly despite their different approaches for building the model. The application of the two algorithms to the hypothetical features went successfully, however the results of the application of the two algorithms to the raw pixel data did not produce good scoring metrics, therefore going forward what this project will focus on is feature extraction of the images, noise reduction of the images and models such as neural networks and support vector machines as they tend to perform well on image data-sets. The steps taken will make all scores and graph results produced from the image data-set closer to the that of the scores and graph results produced by the UCI hypothetical features.

6 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 .

7 Bibliography

References

[1] William P Arend, Beat A Michel, Daniel A Bloch, Gene G Hunder, Leonard H Calabrese, Steven M Edworthy, Anthony S Fauci, Randi Y Leavitt, JT Lie, Robert W Lightfoot Jr, et al. The american college of rheumatology 1990 criteria for the classification of takayasu arteritis. *Arthritis & Rheumatism*, 33(8):1129–1134, 1990.

[2] Huy Bui. Roc curve transforms the way we look at a classification problem. *https://towardsdatascience.com/a-simple-explanation-of-the-roc-curve-and-auc-64db32d75541*, 2020.

[3] 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). *Cancer Causes & Control*, 13(9):813–823, 2002.

[4] Mohamed Amine Chikh, Meryem Saidi, and Nesma Settouti. Diagnosis of diabetes diseases using an artificial immune recognition system2 (airs2) with fuzzy k-nearest neighbor. *Journal of medical systems*, 36(5):2721–2729, 2012.

[5] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 1967.

[6] Judith A Falconer, Bruce J Naughton, Dorothy D Dunlop, Elliot J Roth, Dale C Strasser, and James M Sinacore. Predicting stroke inpatient rehabilitation outcome using a classification tree approach. *Archives of Physical Medicine and Rehabilitation*, 75(6):619–625, 1994.

[7] Anuja Nagpal. Decision tree ensembles bagging and boosting. *towardsdatascience.com*, 2020.

[8] Mohammad Ashraf Ottom. Classification of mushroom fungi using machine learning techniques. *International Journal of Advanced Trends in Computer Science and Engineering*, 8:2378–2385, 10 2019.

[9] Lior Rokach and Oded Maimon. Top–down induction of decision trees classifiers–a survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, page 487, 2005.

[10] 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)*, pages 1242–1246. IEEE, 2016.

[11] Manish Sarkar and Tze-Yun Leong. Application of k-nearest neighbors algorithm on breast cancer diagnosis problem. In *Proceedings of the AMIA Symposium*, page 759. American Medical Informatics Association, 2000.

[12] Rahul Saxena. How decision tree algorithm works. *https://dataaspirant.com/how-decision-tree-algorithm-works/*, 2020.

[13] Jeff Schlimmer. Uci mushroom dataset. *archive.ics.uci.edu*, 2020.

[14] Hendrana Tjahjadi and Kalamullah Ramli. Noninvasive blood pressure classification based on photoplethysmography using k-nearest neighbors algorithm: A feasibility study. *Information*, 11(2):93, 2020.

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

[16] Vispedia. Danish svampe atlas. *https://github.com/visipedia/fgvcx_fungi_comp*, 2018.

[17] Eric W Weisstein. Vector norm. *mathworld.wolfram.com*, 2020.

[18] Eibe; Hall Mark Witten, Ian; Frank. Data mining. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, pages 101–103, 2011.

[19] Xueyan Wu, Jiquan Yang, and Shuihua Wang. Tea category identification based on optimal wavelet entropy and weighted k-nearest neighbors algorithm. *Multimedia Tools and Applications*, 77(3):3745–3759, 2018.

[20] Qi Yu, Antti Sorjamaa, Yoan Miche, Amaury Lendasse, Eric Séverin, Alberto Guillén, and Fernando Mateo. Optimal pruned k-nearest neighbors: Op-knn application to financial modeling. In *2008 Eighth International Conference on Hybrid Intelligent Systems*, pages 764–769. IEEE, 2008.

[21] Junni L Zhang and Wolfgang K Härdle. The bayesian additive classification tree applied to credit risk modelling. *Computational Statistics & Data Analysis*, 54(5):1197–1205, 2010.