Using Supervised Machine-learning Techniques to Identify Objects Classes in Images with Depth Data

Alan Lau

MComp (hons) Computer Science
University of Bath
April 2016

Using Supervised Machine-learning Techniques to Identify Objects Classes in Images with Depth Data

Submitted by: Alan Lau

COPYRIGHT

DECLARATION

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:

Abstract

This project is an investigation of whether depth information captured using an RGB-D camera was useful in classifying classes of objects in a scene, and how classification models perform in a complex, real-world problem. We will look into three supervised classifiers - support vector machines (SVM), random forest and AdaBoost.

Basing on the *NYU Depth Dataset*, feature engineering was performed using methods such as K-means to obtain our training dataset. Then, tests were conducted to evaluate how these algorithms perform with depth data, and whether they were suitable for our problem.

It was found that random forest was best at dealing with a complex and noisy datasets, achieving an accuracy of 43.8%, with precision at 51% and recall at 49% with the training and testing datasets. The accuracy and precision-recall rates demonstrate that depth can be made useful in prediction classes of objects in a scene. In fact, the classifier was able to resemble the key parts of an image despite an expectataion of lower performance than the approximate accuracy scores may suggest. We then discuss the future work that can be performed to build on the findings of this project.

Contents

List of Figures

List of Tables

Acknowledgement

This project is completed under the guidance of my supervisor, Dr Neill Campbell. Thank you for your kind support and advice.

Special thanks goes to the Data Science team at Jagex Games Studio, Cambridge. The exceptional team provided me an invaluable experience on learning more about data during my placement year. Thank you for inspiring me to take on a project of this kind, in particular to my managers, Chris Smith and Simon Worgan. Paul Wilson, a dear friend and ex-colleague at Jagex, provided unequivocal support with queries about the project. Thank you!

Also, to Stephen Daly, my dearest friend and partner, for supporting me throughout the project. Thank you!

Lastly, for all those who have given me a helping hand through the years, directly and indirectly, thank you.

Chapter 1

Introduction

Machine learning has been an interest since the early days of the invention of computers. One active area of research of the application of machine learning is computer vision. It aims to learn from images by finding distinctive characteristics in the underlying numerical or categorical values through methods like curve fitting and splitting up the data. For humans, we understand the context of an image so that we can still recognise objects in it even if they are of different colour or made from different materials, but it is a challenging problem for computers.

Images are inherently difficult to model as they have complicated distribution in the data level. An orange in different images may appear different due to the environment it is in, the angle of which it is taken from or colour temperature of the scene. Hence, simply using pixel (colour) information may not make the cut.

Nowadays, some cameras are capable of capturing not only colours, but also depth information to show where an object sits in a 3-dimensional space. One particular interesting project is Kinect Fusion, where it fuses images together to create a 3-dimensional reconstruction of the scene and represent it in a 3-dimensional space.

In fact, research, such as SemanticPaint (Valentin et al., 2015), made use of this new source of information to create a virtual/ augmented reality experience. Valentin et al. (2015) provided a way to label objects and applied the same effect on other similar objects in a scene in real-time. However, these recognised objects are not stored. Can we find a generalised model using depth information to correctly predict objects in a scene? This way, we could use this information to enable more interesting virtual/ augmented reality experience.

A good use-case would be a real-time system that suggests what can be placed on top of a given recognised object. For instance, the system might suggest that 'bowls' and 'cups' can be placed on top of a table when a table top is recognised, and 'pots' and 'pans' when a stove is recognised.

Such recognition is a classification problem. We want to find an optimised decision boundary that splits the data into regions in order to obtain a model that can predict unseen objects. Together with a complex distribution, this optimisation problem becomes even more complicated and requires careful consideration. This contributes to an accuracy and speed trade-off which we will look into in greater detail.

1.1 Aim

There are several aims with this project. For one, we want to find out how popular classification algorithms perform in real-world applications. Formally, these models are very robust with techniques to avoid common issues such as overfitting (taking noise in the training data as features for the whole datasets). But, it is hard to know how they perform with large and complex datasets. The raw dataset that we are basing this project on will contain many images hence many possible datapoints. In chapter 4, we will discuss our approach in dealing with such situation.

We also want to investigate the usefulness of depth data alone in predicting objects in a scene. Could we obtain a classifier with high enough accuracy that can predict objects in an image to good quality? With a high accuracy classifier, we can integrate it into the aforementioned use-case in conjunction with other computer vision techniques such as segmentation.

To limit the scope of the project, we will focus on training supervised classification models, which we will discuss briefly in section 2.3 and in detail in chapter 3.

1.2 Evaluating Success

In order to evaluate how well our classifiers perform, we use various quantitative approaches, including accuracy and precision-recall rates. With these values, we can examine whether we have trained a useful classifier with good prediction power.

Also, we want to ensure that we find the best decision boundary fit while avoiding overfitting. An overfitted classifier does not possess any meaningful prediction power, even though it might show impressive results in hindsight. We want to avoid being too specific by using every detail of the training dataset without compromise, otherwise we will end up with a classifier that is tailored only to this dataset.

We will discuss all these concepts in detail in the upcoming chapters.

Chapter 2

Literature Review

This project aims to facilitate an augmented reality experience, where it could provide suggestions of the classes of objects that can be placed on top of a recognised object in the real-time scene using a Microsoft Kinect Camera. To achieve such a goal, classification, a form of machine-learning, is proposed to be used to recognise classes of objects in images. We can train different statistical classification algorithms with many similar objects, which then create generalised classifiers that can be used to predict unseen objects.

Being an RGB-D camera, not only does the Kinect Camera capture colour (RGB) images, it also captures depth information via infra-red combined with a monochrome camera (Microsoft, 2013). This information enables a detailed 3-dimensional reconstruction. We will look more into the functionality of these cameras in section 2.1.

A group of New York University researchers have created a depth dataset called the *NYU Depth Dataset* captured by a Kinect camera (Silberman et al., 2012), which is freely available online [1]. A variety of objects are scanned, segmented and labelled from a scene. The variety and the number of scans makes it an ideal candidate as training data for our classification problem. We will look into the available datasets in more detail in section 2.2.

This Review attempts to explore in greater detail about the NYU Dataset and its applications, how depth maps are useful to 3-dimensional reconstruction and how the depth dataset might be useful for training classification models. We will also look at the different classification algorithms and find out which of them we should try.

[1] The NYU Depth Dataset V2 is available for download at `http://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html`

2.1 Kinect Fusion

We start off by looking at Kinect Fusion, where our depth dataset is created from.

The Kinect Camera was first released for the company's gaming console, XBox 360. It was designed to recognise gestures, faces and voices, providing a more physical way and new dimension to interact with the interface and games than a conventional controller. A Windows-compatible version of the Camera, an SDK and Kinect Fusion were released later, enabling research into the usage of depth information and the development of commercial products (Microsoft, 2013).

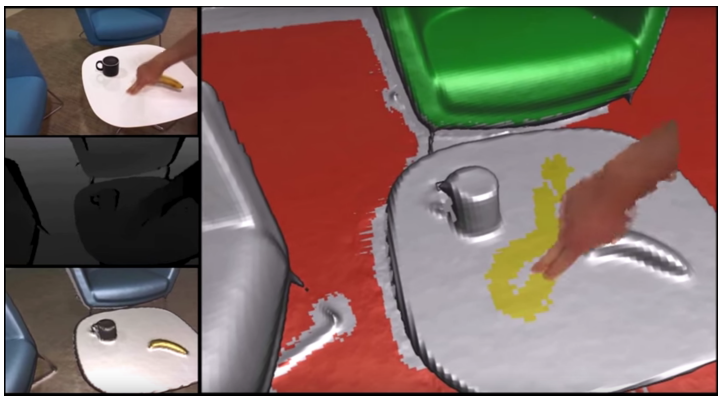

Figure 2.1: *A screenshot to show how SemanticPaint utilises depth information from a Kinect camera and Kinect Fusion to colour user-defined areas in real-time. (Valentin et al., 2015)*

Augmented reality (AR) and real-time reconstructions are some of the most popular research area with Kinect Fusion. SemanticPaint (Valentin et al., 2015) demonstrates the possibility of Kinect Fusion in AR in real-time scenes. It allows parts of the scene to be 'painted' by the user. It also uses segmentation and object recognition to paint similar objects in the same colour. Although this project does not involve real-time processes, it provides some required knowledge about how to create or use an existing segmentation algorithm and classification approaches to label individual items from a scene.

2.1.1 Kinect Camera Technologies

There are two generations of Kinect Camera, where they use different 3-dimensional camera technologies to obtain depth information about a scene. Each of these technologies has its pros and cons, which is discussed below. However, the common problem of these technologies is that they do not deal with very bright light, where detail will be lost (Shao, Han, Kohli and Zhang, 2014).

Structured Light Structured light is used in the first generation Kinect Camera (2010). A sequence of known infra-red pattern is projected onto the scene. A deformed pattern is formed when objects are 'in the way' of the pattern. The object

is then observed from another angle by the monochrome camera. Through analysing the deformed patterns and observations, the depth information about the scene can be obtained (Shao et al., 2014)(Sarbolandi et al., 2015). It is worth noting that the NYU Depth Dataset (Silberman et al., 2012) uses information captured by the first generation of the Kinect Camera, which is discussed in section 2.1.3.

Time-of-Flight Rather than looking at the deformed pattern, the second-generation Kinect Camera (2013) estimates depth information based on the time the infra-red beam takes to travel to and back from an object. The difference between the reference signal and the returned signal allows the calculation of a time difference, which helps estimating the required depth information (Shao et al., 2014).

2.1.2 Comparing Structured Light and Time-of-Flight

Structured light cameras obtain more robust depth data than time-of-flight cameras, because they observe the pattern formed rather than being estimated using the time taken for the infra-red ray to be reflected off the object.

On the other hand, time-of-flight cameras have more advantages than structured light cameras. For instance, they can handle a much brighter situation - 1W power of light versus 1 μW (Li, 2014). Also, they are capable of higher frame rate, which is especially useful when capturing videos and for real-time purposes, as no software is required to interpret the observed pattern (Sarbolandi et al., 2015).

Although the depth data captured by time-of-flight cameras is less precise, they provide a better real-time experience due to their higher frame rate capability and fewer environmental requirements.

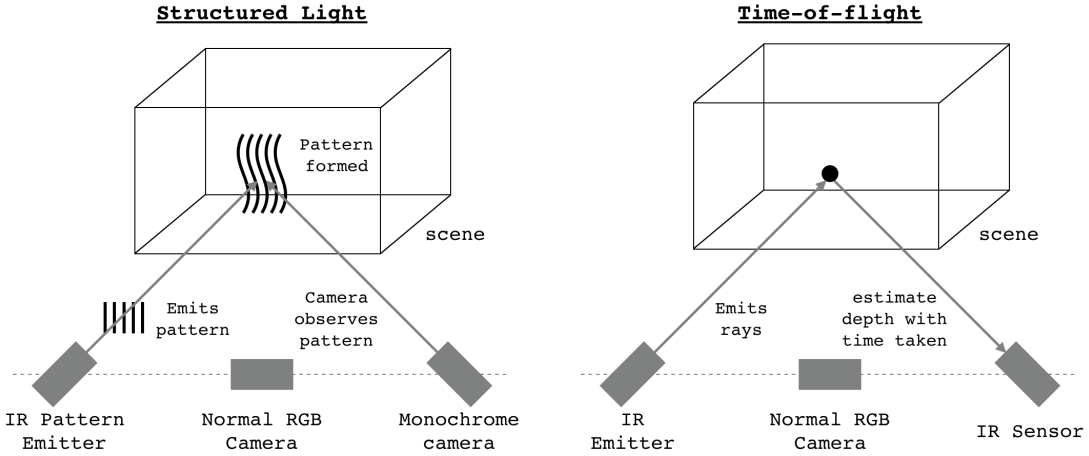

Figure 2.2: *Graphical illustration of the differences between structured light and time-of-flight cameras.*

2.1.3 Data Representation

2.1.3.1 Point Cloud and Depth

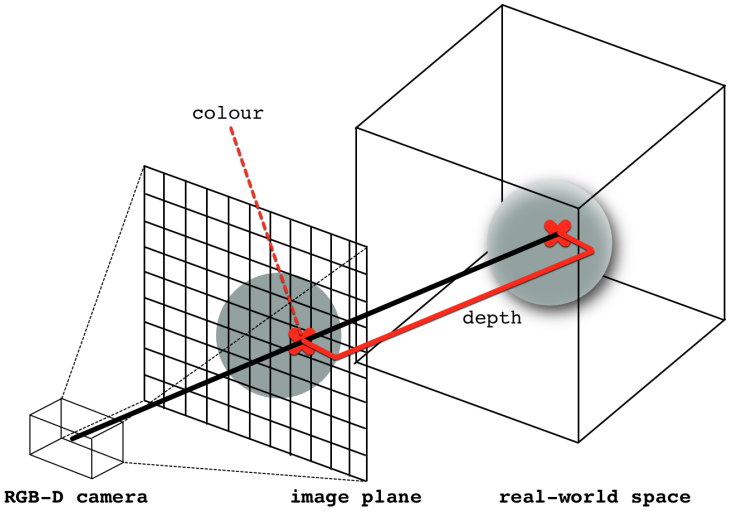

Figure 2.3: *Representing depth in a 3-dimensional space.*

A point cloud stores data points in a coordinate system (Shao et al., 2014). In a real-world case with captured images, a 3-dimensional space is used. This can be a coordinate system or some other unit such as distances from the camera. In the context of depth from the view of a camera (illustrated in Figure 2.3), it can live in some coordinate system which can then be used to infer the distance between the formed image and the point in that space. A point cloud can also be used to generate a mesh so that it can be used to render a visual image of the reconstruction volume.

Figure 2.3 demonstrates the information captured by an RGB-D camera like the Kinect. The image plane shows the normal colour image captured just like any other camera. The 2-dimensional plane is composed of a grid of spots known as pixels, with each representing some form of colour, created using a mixture of different levels of red, green and blue. Depth is the distance between a pixel in the image plane and the actual position of that pixel in a real-world, 3-dimensional space.

2.1.4 Reconstruction

2.1.4.1 Pipeline

A single raw capture with the Camera does not provide much information about a scene. Combining the depth information of many images together enables a super-resolution reconstruction (Izadi et al., 2011), creating a detailed and high quality reconstruction. The following is the full pipeline of how a reconstruction is created using raw depth data:

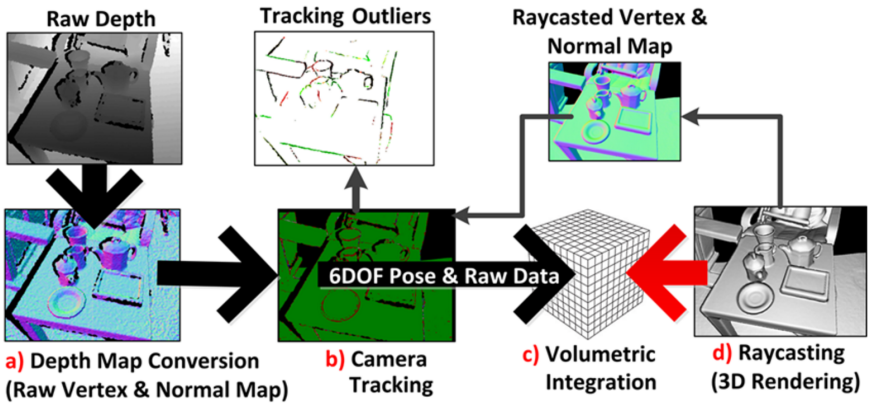

Figure 2.4: *Steps to create a 3D reconstruction with many images (taken from (Microsoft, 2013)).*

1. Raw Input Conversion
   - The raw *depth map* is captured by the infrared-monochrome subsystem of the Camera. This information is not very detailed. A depth map stores the distance between the RGB image plane and the real world location in a 3-dimensional coordinate system (recall Figure 2.3).
   - It needs to be combined with the *normal map*, which is the surface normals associated with each vertex (Szeliski, 2010). It provides a more detailed but still noisy reconstruction at this stage (Microsoft, 2013).
   - Further conversion and reconstruction is needed to retain detail, remove noise, and fill in holes missed by the Camera, possibly due to bright light.
   - This information is stored as a point cloud and will be combined into one representation later on.
2. Camera Pose Tracking
   - The location and orientation of each frame (videos are created using multiple images called frames), often referred to as the world pose, are tracked as the camera moves around.
   - This alignment is constantly traced, allowing all the point clouds to be aligned together (Microsoft, 2013).
   - The frames captured from different poses, even the smallest movement (e.g. caused by a hand-shake), will allow further quality improvements to the scene, achieving more than what a single raw capture is capable of (Izadi et al., 2011).
3. Fusing
   - The depth data converted from the raw input is combined into a single 3-dimensional space per frame.
   - A running average of depth is kept. This reduces noise, and creates a refined reconstruction by combining all the information in one place (Microsoft, 2013) (Izadi et al., 2011).

4. Volumetric Integration

    - The resultant point cloud will be of a highly detailed reconstruction of the scene. For example, grills of a millimetre can be reconstructed properly (Izadi et al., 2011).
    - A rendered image of the 3-dimensional reconstruction volume is possible by using methods such as ray-casting to provide a visual feedback of the scene, and allows for many possibilities, such as augmented reality applications.

2.1.4.2 Volumetric Representation

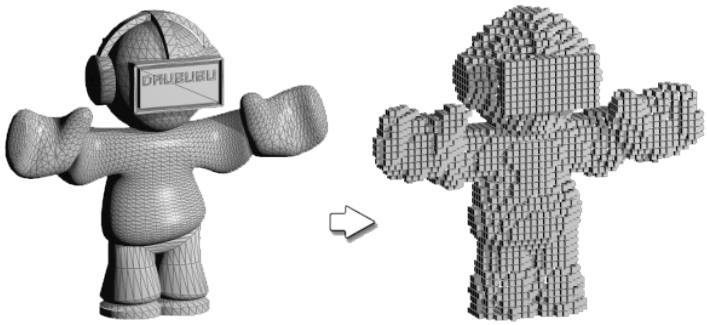

Figure 2.5: *A character represented using a voxel volume*[2]

By averaging the surface models and depth data from multiple viewpoints into one voxel volume, the scene appears to live in a 3-dimensional 'box' constructed in voxels. As illustrated in Figure 2.5, a voxel can be imagined as a 3-dimensional pixel that has some parts 'carved away' to represent the curvatures as perceived in the real world (Szeliski, 2010).

If this high quality reconstruction can be done fast enough, it can be used for interesting applications, such as an augmented reality experience where virtual particles appear to live and follow the cantors of the objects in the scene, as demonstrated by Izadi et al. (2011).

2.1.5 Segmentation

In order to perform classification of objects in a scene with many objects, we need to single them out individually so that operations such as labelling can be done. In computer vision, this is called *segmentation*. This is not a simple problem. As illustrated in Figure 2.6, segmentation is a subjective task even for human. The image can be segmented in many 'correct' ways.

[2]Image taken from `http://www.gamersnexus.net/gg/762-voxels-vs-vertexes-in-games`

Figure 2.6: Showing some posibilities of segmentation done on the same image (Taken from lecture slides created by Chinery (2016)).

There have been many studies on the topic of image segmentation. Results produced using unsupervised classifiers such as K-means and mean-shift are often used as benchmarks to test against their research (we will discussed about unsupervised classification in section 2.3).

Many approaches have been tried on a wide variety of segmentation problems. For example, Alpert et al. (2012) used a probabilistic approach to segment between foreground and background, Rother et al. (2004) utilised both colour and edge information to 'perfectly' remove the background and obtain the foreground with smoothed edges.

Semantic segmentation is another interesting segmentation problem. It attempts to segment a complex multi-object scene meaningfully. Silberman et al. (2012) used depth information and support inferences to enhance segmentation and labelling of these areas. As a side note, support inferences tell the relationship between objects - e.g., 'if a cup is supported by the book, then the cup must be lifted first' (Silberman et al., 2012).

More advanced research often introduce new datasets enabling further research. In fact, Silberman et al. (2012)'s research resulted in the creation of an extensive depth information dataset known as the *NYU Depth Dataset*. Further research done by Ren, Bo and Fox (2012) improved its labelling accuracy by 20%. We are going to base our classification dataset on this dataset, which we will now discuss.

2.2 Depth Data

As the objective of the project is on real-world application, we require a dataset that contains many image scans of a variety of scenes. Firman (2016) has compiled a list of RGB-D datasets available for research use. Many of the datasets are images of objects in lab setups. While these datasets may enable impressive results more easily, they lose the real-world scenario we are aiming for.

2.2.1 NYU Depth Dataset V2 (Silberman et al., 2012)

Quality training data is required to achieve high precision and accuracy. The NYU Depth Dataset provides a quality set of depth data which can be used for training a classifier with the aim of achieving high precision and accuracy. There are 1449 densely labelled images from a wide range of scenes in this version, containing 895 classes of objects. A large, high quality dataset from a wide range of scenarios provides a good base for trying to achieve a high performance, generalised classifier.

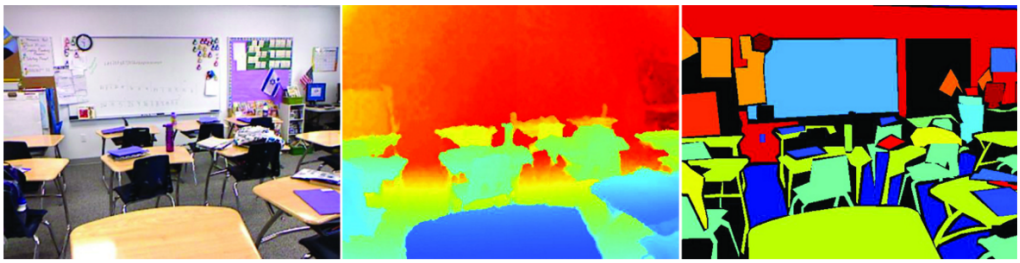

Figure 2.7: *(From left to right) original RGB image, raw depth map, processed depth map (taken from Silberman et al. (2012)).*

The raw depth information captured by the camera would lose finer, sometimes important, detail. Silberman et al. (2012) provides processed depth maps for each of these images, which contain well-defined depth data for the objects. The dataset also contains accelerometer metrics (roll, yaw, pitch and tilt) which could be used to normalise the images to point at the same direction.

It also contains labels for each instance of the objects in a scene. For example, if there are three chairs, they are labelled as *chair1*, *chair2* and *chair3*. In computer vision, we might want to match objects between two images so that we can create a panoramic image (this is beyond the scope of this project).

2.3 Classification

The idea of classification is to identify the group an object belongs to. This relies heavily on a good algorithm that is able to group similar objects together in the first place. In more formal words, a classifier groups objects that has some semantic similarity enough to be classed as the same type. Each class has a label in which these objects are identified as. For example, round objects that can hold liquid can be classed as 'bowls', despite being different in colour and of slightly different shapes (Hall, 2015).

2.3.1 Classification Types

There are many classification models that are fit for different purposes. They are broadly split into two groups - *supervised* and *unsupervised (clustering)*. The following explains what they are. We will look at some classification algorithms that could be used for this project.

2.3.1.1 Supervised Classification

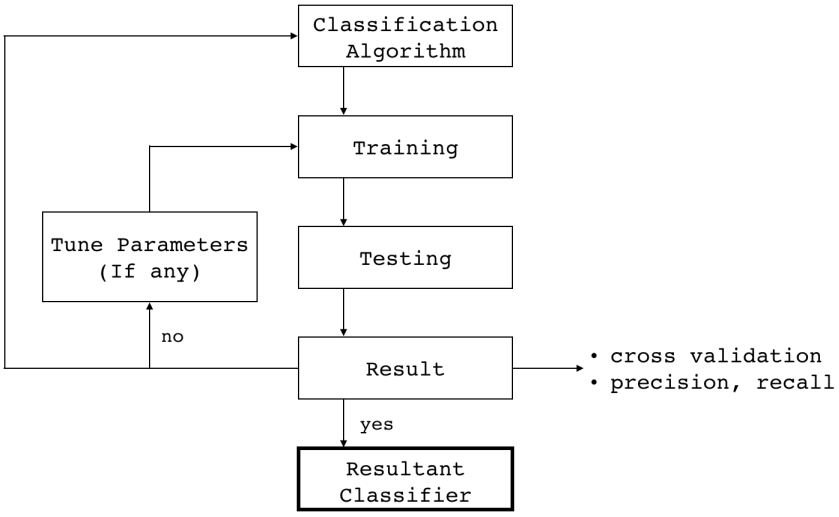

Figure 2.8: *An overview of the process for getting a supervised classifier (Sahoo et al., 2012).*

Supervised learning uses pre-labelled data to train a classifier. The labels and the number of them are pre-defined so that the classifier has a limited number of choices. The classifier has to aggregate and understand the similarities so to be able to say which class an unknown object is, based on what is already known (Sahoo et al., 2012).

A supervised classifier can either be *probabilistic-based* or *geometric-based*. Probabilistic-based algorithms involve a probabilistic density function (Gaussian distribution being one of the most well-known ones), and are sub-divided into *parametric* and *non-parametric*. For a parametric classifier, the statistical probability distribution of each class is known and used, whereas the purpose of a non-parametric classifier is to esti-

mate the distribution, as the number of parameters is not known, or does not matter (Sahoo et al., 2012) (Hall, 2015).

Supervised learning is ideal when there are an abundance of correctly labelled data for the classifier to learn from. Figure 2.8 describes the process of performing supervised classification graphically. A *training set* contains a subset of the pre-labelled data, which is used to train a selected classification algorithm. Some of the pre-labelled data is set aside and used as the *test set* to see how well the trained classifier performs.

We should emphasise the importance of the *validation* and *test* sets in evaluating the performance of a classifier. At training time, we can use the validation set to evalute the performance and rectify any mistake in classification quickly. The test set should be kept blind until the final classifier is produced.

Overfitting is a common issue when the classifier becomes too specific to the training data. In other words, it is treating noise as true signals by taking every minor variation in the training data into account (Murphy, 2012). If a classifier is overfitted by the training data, its ability to predict unseen information is hampered. In such case, we expect a great difference between the training error and testing error. We will further discuss this issue in chapters 3 and 4.

2.3.1.2 Unsupervised Classification

Unsupervised classification is also known as clustering. Items with similar properties in the data level will be grouped together by the algorithm itself. Training is still required, but the label is assigned by the classifier rather than manually. The training data should be unlabelled, to allow the algorithm decide on the best way to group the data as classes. Ultimately, the goal with unsupervised learning is to avoid any intervention and let the algorithm do its job (Hall, 2015).

In computer vision, unsupervised classification is mostly used for object recognition in moving scenes and image segmentation. As mentioned, we are going to focus on supervised algorithms to manage the scope of this project, so that we shall only look into this briefly.

This should not be confused with *online learning*. Online learning updates the classifier as new datapoints come in (Murphy, 2012). Quality new data should improve the classifier over time. This can be used with both unsupervised and supervised algorithms. For example, a streaming version of random forest is used to learn about newly labelled areas in real-time by Silberman et al. (2012).

2.3.2 Classifiers

We now look at some of popular and notable classification algorithms for both supervised and unsupervised methodologies briefly. We will discuss some of them in detail in chapter 3.

2.3.2.1 Notable Supervised Classifiers

Broadly speaking, there are four groups of supervised classification algorithms, namely, *Bayesian*, *Trees*, *Lazy* and *Functions*. Table 2.1 shows the algorithms and their associated groups that we are going to talk briefly about.

Group	Model to be discussed
Bayesian	Naive Bayes (NB)
Trees	Decision tree (DT) Random forest (RF) AdaBoost (ADA)
Lazy	K-nearest neighbours (KNN)
Functions	Support vector machine (SVM)

Table 2.1: *Groups of supervised classification algorithms and their notable examples.*

- **Naive Bayes (NB)** uses the Bayes' Theorem of probability reasoning (the probability of an event happening given some condition). It also assumes that features are conditionally independent given by the Bayes' Theorem, although this may not be true in reality (Murphy, 2012). The algorithm obtains its probability by representing the data as distributions and extracting the class that gives a higher probability. Theoretically, it is robust to overfitting and is a simple algorithm with speedy performance.

- **Decision Tree (DT)** is a multiclass classification algorithm that splits the dataset in the input space and gives local models for each region. This creates a tree with each node containing some conditions for deciding which children node to go next. By counting the number of datapoints going from the starting node to a leaf for each possible path, we obtain a distribution in the end for each leaf showing the most probable class this path leads to. This is a simplistic and efficient way in finding different groups within the input space. However, it can be overfitted easily, making it difficult to use with complex datasets (Murphy, 2012).

- **Random Forest (RF)** is an ensemble method. It obtains a prediction by averaging the results of many decision trees, which was found to be more effective than a single decision tree (Caruana and Niculescu-Mizil, 2006). It attempts to overcome the problem of overfitting by randomising both the features and data used to build each tree.

- **AdaBoost (ADA)** uses boosting to achieve a high quality classifier. Over some specified number of iterations, weak learners are trained and combined to form the final classifier. A weak learner is one that does not give much information

nor very accurate by itself, but is always be better than random (Murphy, 2012). These weak learners are usually decision trees.

At each iteration, a weight is added to misclassified datapoints, which is then used by the next iteration to train the next weak classifier. Additively, the resultant classifier becomes a robust classifier that provides strong classification power (Schapire and Freund, 2012).

- **K-Nearest Neighbours (KNN)** uses K datapoints around each datapoint to obtain the class it belongs to. Despite being a non-parametric model means that it is fast at training, it assumes strongly about the data distribution. Each datapoint simply takes the count of each class in the surrounding datapoints to decide on its class. Other downsides include complex decision boundaries and does not work well with high dimensional datasets, known as 'the curse of dimensionality' (Murphy, 2012).

- **Support Vector Machines (SVM)** aims to maximise the margins which run parallel to the decision boundary and are defined by points of each class. Data points forming the margins are called support vectors. It is a popular algorithm because of its versatility, with many tunable parameters and kernels for different distributions of data. It started off being a binary classifier, but is extended for multiclass classification by building multiple classifiers and comparing between them – one-versus-one and one-versus-the-rest are the two methods in achieving multiclass SVM (Bishop, 2006).

Note that from now on, we will use the acronyms in brackets stated next to a term for clarity. For instance, we will refer to Support Vector Machine as SVM.

Model	Training Speed	Classification Speed	Mean Accuracy
BST-DT	Fast	Fast	89.6%
RF	Rather fast	Rather fast	89.2%
SVM	Slowest	Slowest	86.2%
KNN	Rather fast	Rather fast	81.5%
DT	Very fast	Very fast	70.9%
NB	Fastest	Fastest	65.4%

Table 2.2: *Comparing properties of different classification models (adapted from Caruana and Niculescu-Mizil (2006))*

Caruana and Niculescu-Mizil (2006) performed 11 tests on widely available benchmark datasets on some supervised classifiers. Table 2.2 shows the properties of some of these classifiers and their average results on the 11 tests. Note that this can only be used as a benchmark as our dataset will be much more complicated and larger than the ones used in their paper.

Tests done by Amancio et al. (2014) and the descriptions in the `scikit-learn` documentation (we will discuss more about `scikit-learn` in section 2.5) both suggest

that SVM can outperform Random Forest with proper parameter tuning and kernel settings, but the trade-off is speed.

SVM is known to not scale well and could require a long training time compared to other models (Caruana and Niculescu-Mizil, 2006). However, it is also one of the popular machine-learning algorithms used for classification and regression problems, due to its ability to be customised with parameters, and support for different kernels to cater for different shapes and density of the input data. We shall discuss more about this in section 3.1.

Tree methods (Boosted Decision Tree (BST-DT), DT and RF) and SVM have different algorithms underneath. Tree methods obtain their good performance through an appropriate amount of randomisation of the training dataset, whereas SVM does so by choosing the right kernel and tweaking its parameters. In chapter 3, we shall discuss in depth their properties.

Despite KNN seems to perform relatively well compared to DT and NB, our dataset will be highly dimension (discussed in chapter 4), meaning that it would not be the best choice of classifier in our case. And, we could most likely disregard DT and NB due to their relatively poor accuracy even with benchmark data.

This leaves us with BST-DT, RF and SVM. In fact, they are widely used to solve classification problems. For instance, (Valentin et al., 2015) uses an online learning varient of RF to train newly labelled items so to paint that class of objects in the scene with the same colour. AdaBoost is a popular boosted tree algorithm, which we should use as our 'boosted decision tree' algorithm. In chapter 5, we are going to find out if these algorithms produce respectable results.

2.3.2.2 Notable Unsupervised Classifiers

As mentioned in the introduction, we are going to focus on supervised classification to limit the scope of this project. Nonetheless, we should briefly look at some popular unsupervised algorithms. In fact, we will be using K-means clustering for a different reason than training a classifier or for image segmentation. Instead, we will use it to engineer our dataset. We will discuss this in chapter 4.

Unsupervised classification models are also known as *clustering*. It aims to group similar objects together (Murphy, 2012). Broadly speaking, we can split clustering algorithms into three categories, namely, *centroid-based*, *distribution-based* and *density-based*. Let us discuss briefly about the models mentioned in Table 2.3.

Group	Model to be discussed
Centroid-based	K-means
Distribution-based	Gaussian Mixture Models with Expectation Maximisation (GMM with EM)
Density-based	DBSCAN

Table 2.3: *Groups of unsupervised classification algorithms and their notable examples.*

- **K-means clustering** takes K random datapoints to be the starting point. Different points are picked until the groups become stable and produces groups with minimal distances between each point and their surroundings. These points would end up being the mean of their corresponding cluster (μ_k for a cluster k), also known as centroids (Bishop, 2006).
- **Gaussian mixture model** combines multiple Gaussian distributions by imposing them to capture more information about the dataset (Bishop, 2006). Similar to K-means, each Gaussian k centres around the mean μ_k, but it also introduces covariance to allow it to work with high dimensionality the data. However, the amount of tunable parameters make it rather unscalable (scikit-learn, 2016).

  Finding parameters are simple when complete data is available, but usually not the case. With the help of the Expectation Maximisation algorithm (often referred to the EM algorithm), it iteratively finds missing values by fixing some parameters (the E step), and then optimising the other parameters with these new values (the M step) (Murphy, 2012). In fact, this method can be applied to many clustering models, including K-means. We will see this more formally in section 3.4.
- **DBSCAN** is a model that finds clusters of any shape by separating out high density as clusters. It is particularly useful for datasets with a widely separated spiral shape. It looks at the number of neighbouring points there are in its surrounding to decide if it belongs to a particular cluster. This is effective in distinguishing between noise and cluster membership, but not in locating closed-by clusters. On the up side, it is even more autonomous than other clustering algorithms such as K-means in the sense that it does not require a specified number of clusters for instance.

2.4 Performance

After we trained a classifier, we have to evaluate its performance. As mentioned in the Introduction, we can use various methods to find this out. In this section, we will look into precision-recall rates and the cross validation.

2.4.1 Precision and Recall

The precision and recall rates (accuracy) can be calculated to analyse the performance of the classifier (Davis and Goadrich, 2006). They are calculated by comparing predic-

tions against the actual values.

	Actual Positive	Actual Negative
Predicted Positive	TP	FP
Predicted Negative	FN	TN

Table 2.4: *Confusion matrix (taken from Davis and Goadrich (2006))*

In order to calculate these rates, a confusion matrix is computed. It is a table that puts the correctness of predictions against the actual values of the underlying data (Table 2.4). This shows the number of correct and incorrect recognition, allowing a couple of metrics to be calculated, notably precision and recall (Table 2.5).

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{True Positive Rate} = \frac{TP}{TP+FN}$$

$$\text{False Positive Rate} = \frac{FP}{FP+TN}$$

Table 2.5: *Common machine-learning evaluation metrics (taken from Davis and Goadrich (2006)).*

Precision and recall takes into account all the true-false positives and negatives. Precision is the positive predictive value – *'how many selected items are relevant'*, whereas recall explains the sensitivity – *'how many relevant items are selected'*, as seen in Table 2.5 and Figure 2.9. The former tells us how well the model can predict correctly; the latter tells us the *quantity* of which the relevant items is picked (Powers, 2011).

To say that a model is *good for use*, it has to be both precise and has a high recall rate. If a model is very precise, but does not pick up much of the relevant items, the model might look as if it was performing well, but in fact, it has hardly picked out enough of the right items; vice versa.

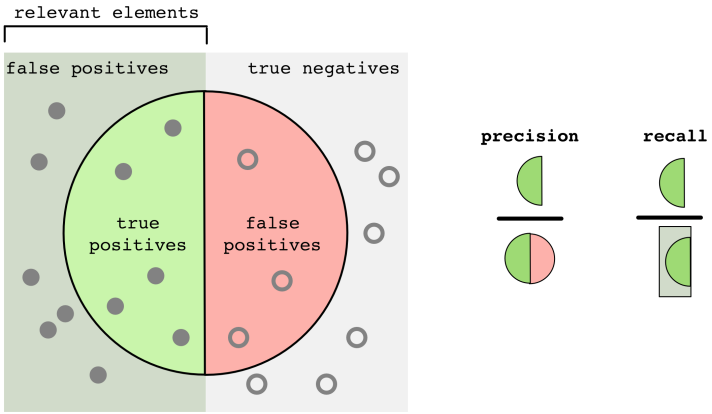

Figure 2.9: *Diagram representation of precision and recall (adapted from Powers (2011)).*

It is important to ensure that the training and testing sets are different and that there is no overlap between the two sets, otherwise overfitting will happen. Overfitting is evident when a classifier appears to be functioning perfectly during the train-and-test stage, but turns out to be unable to classify new objects when put into use. On top of the training and test sets, a third validation subset can be introduced. This enables an extra check to ensure that the classifier actually works. However, the precision and accuracy of the classifier could suffer as a result of a smaller training set (Pedregosa et al., 2011).

2.4.2 Cross-validation

It could be problematic to hold out data for training, validation and testing when there is a lack of a sizeable dataset. On one hand, we know that it is important to perform testing to find out the real performance. On the other hand, we want to learn from as much information as possible.

Cross-validation solves the problem by creating some *mutually exclusive* folds (subsets). These folds are created after a test set is taken out from the original set of data. Many classifiers are trained with different splits, resulting in an average that estimates the complete cross-validation (Kohavi et al., 1995). This gives an unbiased view as to how the classifier performs, as it is not limited to one fixed set of testing data.

One popular algorithm is k-fold cross-validation. It does not run validation on all possible splits on the folds but one different split for each run to save on computational complexity, while giving a good estimate on how the classifier is performing.

2.4.3 Evaluating Unsupervised Classifier

Although we are not attempting clustering as a classification solution, we should understand the methods that can be used to evaluate them.

Clustering methods are more difficult to evaluate, but not dissimilar to how supervised classifiers are evaluated (Murphy, 2012). There are two types of evaluations. *Internal evaluation* looks at the data from within the cluster, much like the training accuracy in a supervised classifier. We try to predict the same set of data as used for training it. *External evaluation* uses data that has not been seen before, which is similar to evaluations of supervised classifier that obtain a test set accuracy score.

Type	**Method**
Internal Evaluation	Davies-Bouldin Index Dunn Index Silhouette Coefficient
External Evaluation	Rand Measure F-measure (F-score) Jaccard Index Fowlkes-Mallows Index Mutual Information Confusion Matrix

Table 2.6: *Evaluation methods for unsupervised classifiers (clustering).*

Internal methods aim to evaluate the separation and density between clusters, and identifying outlying points. Most external methods require a ground-truth dataset (an unseen labelled dataset that can be used to compare with the trained model) (Färber et al., 2010). For instance, F-measure provides a single value by taking the mean of precision and recall harmoniously.

2.5 Tools for Machine-learning Application

There are many tools out there that provide machine-learning capabilities. For the purpose of this project, we are going to use Python due to the availability of some useful packages and our familiarity with the language.

Nevertheless, we shall look into the three most popular languages and their associated libraries for machine-learning.

2.5.1 Python

Python is a general purpose programming languages widely used for scripting tasks. It is one of the top programming languages used for many purposes, with a wide range of libraries for different tasks. Python is open-sourced, meaning that it is freely available for use. However, Python suffers from the Global Interpreter Lock, disallowing multiple threads to run at one time (Python, 2015). This could hamper performance when a large amount of memory or parallelism is required.

- `scikit-learn` is an actively developed machine-learning library for Python. It has quality implementations of popular machine-learning algorithms for different machine-learning problems. It is built on `numpy` and `scipy`, making it easy for data manipulation (Pedregosa et al., 2011).

  A wide range of companies use scikit learn for machine-learning purposes, including Spotify and Evernote.[3] Pedregosa et al. (2011) compared it with other Python machine-learning packages which sees scikit-learn beating the others in baseline tests in terms of speed and performance. It has implementations of the supervised and unsupervised classification algorithms we briefly mentioned in section 2.3.2.

  It also supports several classification tasks to run at the same time when using a multi-core computer with its integration with `joblib`. However, it is not possible to run them in parallel across multiple machines.

- `numpy` is a powerful tool for manipulating and storing arrays. It allows an array to be reshaped into any dimension without altering the underlying data, enabled by its powerful memory management. It is able to be represented much more efficiently than the built-in list structures in Python (van der Walt et al., 2011).

2.5.2 R

`R` is a popular language used by the data analytics and data science community. It provides an extensive selection of libraries for data manipulation and machine-learning purposes. It provides great visiualisation libraries such as `ggplot2` for graph plotting and `data.tables` for efficient data manipulation.

[3] According to scikit-learn at `http://scikit-learn.org/stable/testimonials/testimonials.html`.

R is an open-sourced statistical programming language. It has an engaged community and can be used in conjunction with other popular general-purpose programming languages. Also, R has a large presence in big technology companies, such as Google, Facebook and Twitter, as well as in academia and sciences[4]. However, R has a steep learning curve because of its syntax. And, it is known to not scale very well as data has to be stored on RAM.

2.5.3 MatLab

MatLab is widely used mainly in academia, and research and development purposes. It provides quick prototyping due to its high-level language. It also contains many libraries and toolbox to perform a wide range of operations. For instance, the Image Processing Toolbox provides for computer vision application, such as image filtering which is used for edge detection. For classification, MatLab has implementations for popular algorithms that we mentioned previously.

2.5.4 Choosing a Tool

It is difficult to decide which language is the best. Like any programming situation, there is no one language that is superior than another. There is only one that is more suitable than another based on its capability to perform a given task.

In this case, they all provide rich libraries and community support with many guides available. R and Python are the languages used by the popular data science competition website, Kaggle, while MatLab is well-established in academia with an abundance of libraries, making it a difficult decision.

Python is chosen because of its simple syntax and our personal experience with it. It is shown to have solved real-life machine-learning problems for reputable companies and widely used. Also, being a language that is widely used for scripting purposes, command line arguments can be used easily to supply variables to the underlying code without having to change the code itself. This is particularly useful for this project, as we will have to run our scripts for some specified functions with different arguments many times.

2.6 Hardware Considerations

We are potentially dealing with many datapoints of large feature vectors. Although a moderately modern computer with relatively large amount of memory and multi-core processors will suffice to moderately challenging machine-learning tasks, it would be ideal to be able to off-lift these heavy operations to a more powerful system that can be used any time anywhere with an Internet connection.

[4]According to Revolution Analytics, maker of R, at `http://www.revolutionanalytics.com/companies-using-r`.

There are two notable options - the High Performance Computer cluster at the University, Balena[5], and Amazon EC2[6]. They are both ideal services as they provide flexibility and scalability that cannot be matched by a personal computer. The following discusses some of their features, and pros and cons.

- **Balena**
  It is made up of many computers to form one big connected computer service. It contains 78 nodes with 64GB of memory and 80 nodes with 128GB of memory. A user can submit a job with a simple script and it will enter a queue. The job will run when appropriate resource is available and that the job has reached the top of the queue. Also, it supports parallel computing, where multiple node can be joined together to perform some tasks. This service can be accessed by any University of Bath personnel that has a need for it.

  There are different groups of accounts available. For a free account, a user can only run jobs up to 6 hours and has limits on how many computing-hours that one can be used at one time. Also, the queue time could be long at busy periods when the cluster has a high volume of jobs.

- **Amazon EC2**
  Amazon EC2 (Elastic Compute Cloud) has been the computing platform used by many renowned companies such as Airbnb and Netflix. It is a very flexible system in that a user can start a new instance of their preferred distribution of Linux or Microsoft Windows Server quickly, with various processor and memory configurations. The user assumes complete control of the system. It acts as a real machine so that they can install and configure the system as it suits them.

  However, there is much overhead to get the instance ready to perform the required tasks for this project. For instance, it might be required to install the appropriate packages and libraries every time we start up a new instance, before any task can take place. Also, it gets quite expensive when more memory and run time is required, as it is charged per hour for when the instance is running. And, there is not the luxury of running many jobs, or jobs over multiple machines to perform operations in parallel.

Given access to a free service providing great computing resource, Balena is the obvious choice. Although there is a time limit and queue times may vary during the course of this project, it provides a great platform to run large or memory-intensive code without much overhead in setting up the servers before they can be used. On the other hand, while AWS provides complete control on the system and provides great flexibility, the cost of running a decent server could go up quickly as more memory and computing time is required.

[5] https://wiki.bath.ac.uk/display/BalenaHPC/Balena+High+Performance+Computing+Service
[6] https://aws.amazon.com/ec2/

2.7 The Project

In this chapter, we learnt that depth information can be useful for different applications. We also looked at the NYU Depth Dataset to find that it is a useful dataset which could be used as our base dataset.

We examined different classification algorithms and found out that BST-DT, RF and SVM performed the best in benchmark comparisons. There are a few algorithms for performing boosting with decision trees. We are going to use AdaBoost as it is one of the most popular implementations of boosting (discussed in detail in section 3.3). We are going to see how they perform with our dataset in chapter 5. After all, no single classification method fits all problem. It is not known how well they will perform on our dataset or whether they will fit for the purpose at all.

After training some classifiers, it is important to conduct many tests to ensure it is performing properly and well. As discussed, precision-recall rates and cross validation are helpful tools to evaluate the performance of the classifiers. They ensure that the classifiers perform properly by examining them empirically.

Also, recall that a *validation dataset* for testing purposes to find out the predictive power of a classifier. If there is a huge difference between training and test dataset error, it is an evidence of overfitting. A *test dataset* is required to evaluate the final classifier. It should not be seen beforehand by the classifier until the final classifier is created, so that the more realistic performance can be measured.

As for tools, we decided to use Python and its associated libraries due to our familiarity with the language and its popularity for machine learning.

Also, we decided to use Balena to perform machine-learning tasks, as it has much more processing power and memory than an ordinary laptop. However, we have to take into account the limitations as a free user on Balena.

Now that we have established a background knowledge about the problem space, we will next look into detail of how our chosen classificaiton models work formally.

Chapter 3

Technical Background

Before we attempt to train models using depth information from images, we need to understand the characteristics of them. In the scope of this project, we are going to focus on three popular models for supervised classification - support vector machine (SVM), random forest (RF) and AdaBoost (ADA), as mentioned in section 2.3.2.

Looking ahead, we should also look into K-means clustering, an unsupervised classification model. Rather than create a classifier using K-means clustering, we intend to use it to engineer our dataset. We will discuss this in chapter 4.

3.1 Support Vector Machine (SVM)

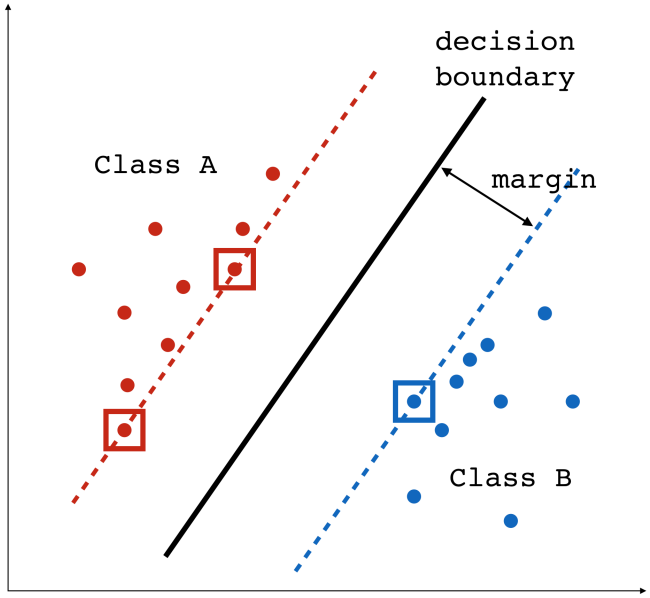

Figure 3.1: *This illustration shows how SVM works with linearly separable datasets and a linear kernel. The squared datapoints represent support vectors for their corresponding class.*

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b \tag{3.1}$$

The linear model above describes the objective of a two-class classification problem. Given a fixed feature-space, $\phi(\mathbf{x})$, there exists at least one pair of weight vector $\mathbf{w}$ and bias b values that gives a solution that separates the input data exactly in the linear feature space. This input data is known as the training dataset, with input vectors $\mathbf{x}_1, ..., \mathbf{x}_N$ each corresponding to a target t with value -1 or 1, i.e. the two different classes. 'Separating exactly' means that

$$\begin{cases} y(\mathbf{x}_n) > 0 \text{ for any point with } t_n = +1 \\ y(\mathbf{x}_n) < 0 \text{ for any point with } t_n = -1 \\ \text{hence, } t_n y(\mathbf{x}_n) > 0 \text{ for any point} \end{cases}$$

If there is more than one pair of parameters that achieve the above, the pair with the smallest generalisation error should be chosen. In SVM, this is done by finding the decision boundary when the margins are maximised. A margin is the smallest perpendicular distance between the decision boundary and any datapoint. The datapoints that enable the margins to be maximised are known as the support vectors. This is

illustrated in Figure 3.1.

$$\frac{t_n y(\mathbf{x}_n)}{||\mathbf{w}||} = \frac{t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b)}{||\mathbf{w}||} \tag{3.2}$$

Assuming all data is classified correctly, the distance between any point $\mathbf{x}_n$ to the decision surface is given by Equation 3.2. Hence, the maximum margin is obtained by considering the pair of weight vector and bias that maximises the distance between the boundary and the closest datapoint $\mathbf{x}_n$, as illustrated in Equation 3.3. As $||\mathbf{w}||$ does not depend on any datapoint, we take out the factor outside of the min function.

$$\underset{w,b}{\text{argmin}} \left\{ \frac{1}{||\mathbf{w}||} \min_n \left[t_n \left(\mathbf{w}^T \phi \left(\mathbf{x}_n \right) + b \right) \right] \right\} \tag{3.3}$$

3.1.1 Overlapping Classes

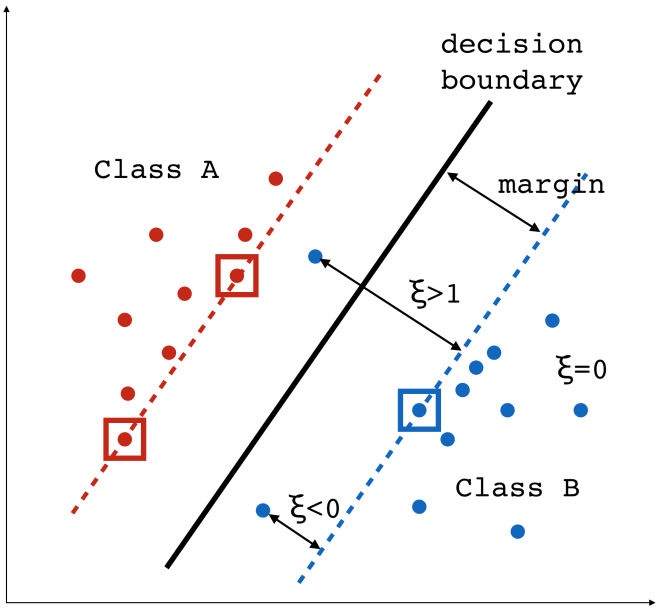

Figure 3.2: *An elaborated Figure 3.1 showing slack values of points based on their location in relation to the margin and decision boundary.*

In a realistic situation, classes may overlap. If we impose a hard separation between these classes, the classifier may become too specific and not be able to predict new data. Therefore, there needs to be a trade-off between splitting the classes perfectly and misclassifying some of the datapoints.

The slack value describes the status of a datapoint. There are three types of values as illustrated in Figure 3.2 and Table 3.1. Correctly classified points lie on or within

the margin, whereas misclassified points are on the other side of the decision boundary. Some datapoints lying on the wrong side of the margin but within its side of the boundary causes margin violation.

Slack Value	**Description**
$\xi = 0$	Points that are correctly classified.
$0 < \xi \leq 1$	Points that lie between the margin and the side of the hyperplane of its class.
$\xi > 1$	Points that are misclassified.

Table 3.1: *Slack values and their meanings.*

3.1.1.1 The Cost Parameter (C)

By allowing some datapoints to be misclassified, we can create a more generalised SVM classifier. In other words, we impose a **soft margin** through penalising these datapoints and relaxing the constraints. In a non-linear dataset, we can think of this as the smoothness of the fit. The smoother the margins, the less we take into account the exact position of all datapoints.

$$\min \left(C \sum_{n=1}^{N} \xi_n + \frac{1}{2} ||\mathbf{w}||^2 \right) \tag{3.4}$$

This trade off between the margin and misclassification is controlled by the regularisation parameter, cost (C). The bigger C is, the more correctly classified points there are, the harder the boundary becomes. As C tends to infinity, we have a hard margin. Hence, we need to minimise the C value in order to find our optimal margin with Equation 3.4.

Cost ($C > 0$)	**Description**
Small C	A little constrained, large margin.
Large C	A more constrained, narrower margin.
$C \to \infty$	A hard margin where all constraint is in place.

Table 3.2: *The effects of different C values on the margin.*

3.1.2 Kernels

By mapping datapoints to higher dimensional feature spaces, originally non-linearly separable datasets become separable linearly. Recall Equation 3.3, this feature space raising is represented as $\phi(\mathbf{x}_n)$ which could be of infinite dimensions and unable to be stored. Instead, in SVM, we can represent this complex decision boundary in efficient kernel representation without computing $\phi(\mathbf{x}_n)$.

Kernel	**Mathematical Expression**
Linear	$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$
D-degree polynomial	$k(\mathbf{x}, \mathbf{x}') = \left(1 + \mathbf{x}^T\mathbf{x}'\right)^d$
Radial Basis Function (RBF)	$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma\|\|\mathbf{x} - \mathbf{x}'\|\|^2\right)$
Sigmoid	$k(\mathbf{x}, \mathbf{x}') = \tanh\left(a\mathbf{x}^T\mathbf{x}' + r\right)$

Table 3.3: *Available kernels for SVM Classifiers in the* `scikit-learn` *library.*

3.1.2.1 The Gamma Parameter (γ)

By taking C into account, we are able to create a more generalised classifier, but it would still be affected by outliers. The RBF kernel gives another parameter, γ, to control for the problem and shape of the decision boundary, as shown in Table 3.3.

γ defines how far the influence of each datapoint reaches. The smaller γ is, the further it reaches. The higher the value, the more the model tries to avoid misclassification. However, if the value is too high, the decision boundary become too specific, causing overfitting (illustrated in Figure 3.3). Hence, the optimisation problem now includes the need to minimise γ so to find the optimal decision boundary.

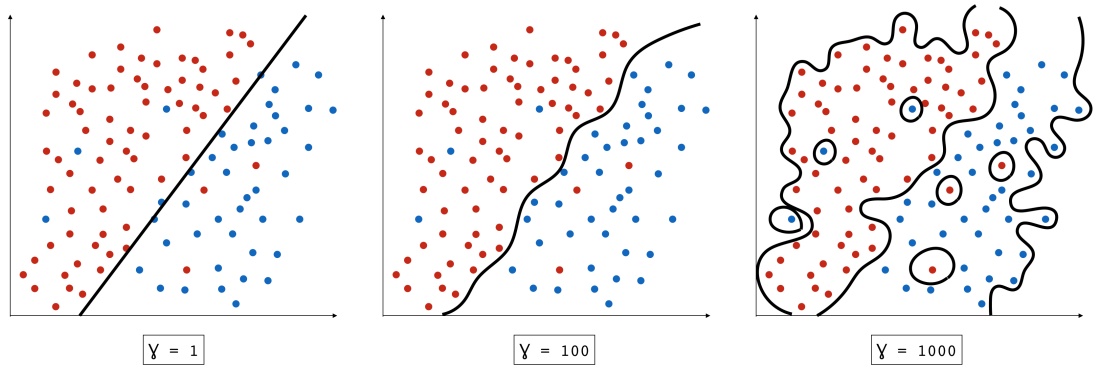

Figure 3.3: *Effects of different γ values in ascending order.*

3.1.3 Multiclass Extension

Fundamentally, SVM is a binary classifier designed to split between two classes. Two approaches, namely *one-versus-the-rest* and *one-versus-one*, have been developed to allow SVM to work with more than two classes by combining multiple binary SVMs.

- **One-versus-the-Rest**

  The approach trains K SVM classifiers for C_K classes. Each SVM classifier $y_k(\mathbf{x})$ uses the k^{th} class C_k as the positive class and the rest as the negative class.

However, this approach has a few issues.

A major issue is that this causes an imbalance of classes as the positive class will be much smaller than the rest of the classes. This problem is worsened as more classes are added to the classifier, as it skews the results to the negative class, giving a false impression that the classifier has a good performance.

- **One-versus-One**

Another approach is to train many SVMs with all possible pairs of classes (i.e. $K(K-1)/2$ SVM classifiers). The relevant classifiers then 'vote' to return the highest voted class to give a prediction.

However, it takes significantly longer to converge than one-versus-the-rest due to the number of classifiers that need to be trained. It trains in $O(K^2N^2)$ time compared to $O(KN)$ for one-versus-the-rest. It also takes a long time to predict – $O(K^2N_s)$, where N_s is the number of support vectors (Murphy, 2012).

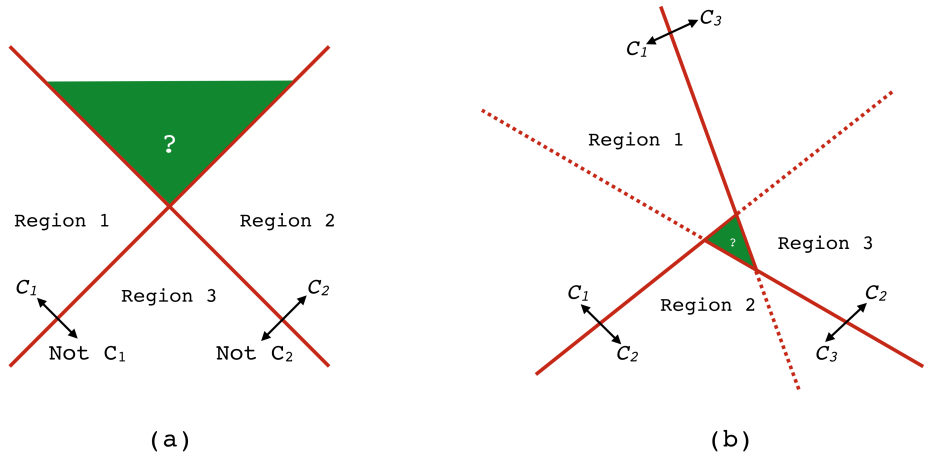

Figure 3.4: *Diagram showing ambiguity regions (green) for (a) the one-versus-the-rest approach, and (b) the one-versus-one approach (based on figure 4.2 of Bishop (2006)).*

Either way, it causes some ambiguity due to the separate evaluation of the classifiers. Some regions will be assigned to multiple classes due to the multiple classifier setup. Figure 3.4 illustrates this issue, and we can see that one-versus-one has a much smaller ambiguity area than one-versus-the-rest, in the expense of a much longer training time.

3.2 Random Forest (RF)

We now look at a rather different classification algorithm, random forests. It is an ensemble method that combines results from different decision trees to give the final classifier. We start by understanding what decision trees are.

3.2.1 Decision Tree

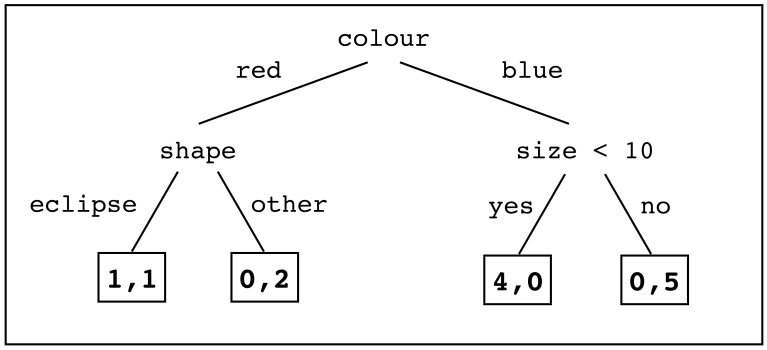

Figure 3.5: *A (classification) decision tree with two classes, A and B, and 'colour', 'shape' and 'size' as features (adapted from Murphy (2012)).*

Decision trees, formally classification and regression trees (**CART**), obtain predictions by recursively splitting the input space and defining each region with a local model. Visually, this can be displayed as a tree, with each node representing a split by some threshold or criterion.

Illustrated in Figure 3.5, the distribution of class labels is stored at each leaf. This gives a probability for each class satisfying a set of criteria by using the number of positive and negative samples. For instance, the criteria 'if the *colour* is red and the *size* is less than 10' gives a prediction for class A $p(y = 1|\mathbf{x}) = 4/4 = 1$ and $p(y = 2|\mathbf{x}) = 0/4 = 0$ for class B. Practically, decision trees are binary such that each node splits into 'yes' and 'no' according to some comparison criteria (Murphy, 2012).

For each point $\mathbf{x}$ to be predicted, it follows down the tree from the top, known as the root, to the resulting leaf node, according to the decision criteria learned from the training data.

$$f(\mathbf{x}) = E[y|\mathbf{x}] = \sum_{m-1}^{M} w_m I(\mathbf{x} \in R_m) = \sum_{m=1}^{M} w_m \phi(\mathbf{x}; \mathbf{v}_m) \tag{3.5}$$

Formally, this model is defined as Equation 3.5. R_m is the m^{th} region and $\mathbf{v}_m$ represents the feature used to split the node. Also, $\mathbf{w}_m$ represents the mean response in the m^{th} region. To generalise the model for classification purposes, this is replaced by the

distribution of class labels as mentioned.

$$f(\mathbf{x}) = w_0 + \sum_{m=1}^{M} w_m \phi_m(\mathbf{x}) \tag{3.6}$$

This is essentially an adaptive basis-function model (ABM) (compare equations 3.5 and 3.6) which tries to learn about the kernel rather than having to be specified manually like that in SVM. Without the need to estimate the kernel parameter which is computationally expensive, random forest learns much quicker than kernel methods such as SVM (Murphy, 2012).

3.2.1.1 Growing a Tree

$$(j*, t*) = \underset{j \in \{1,\ldots,D\}}{\operatorname{argmin}} \min_{t \in \tau} \left(cost\left(\{\mathbf{x}_i, y_i : x_{ij} \leq t\}\right) + cost\left(\{\mathbf{x}_i, y_i : x_{ij} > t\}\right)\right) \tag{3.7}$$

We require some methods to decide which feature and what value to use for splitting. The split function in Equation 3.7 returns the best feature and best value by comparing a feature x_{ij} to a numeric value t. This t value is bounded by the feature values from the rest of the datapoints, stored as τ_j for each feature j.

As seen in the equation, we want to minimise the cost of picking this feature. Cost is measured in several ways, namely, *misclassification rate*, *cross-entropy* and *Gini index*. We can minimise the *misclassification rate*; we can maximise the information gained through the split by minimising the *cross-entropy*; or, we can minimise the *Gini index*, which is the expected error rate.

Cross-entropy and Gini index are preferred because it would choose the feature that gives a pure split, i.e. contains only one class, in case of a draw in the error rate (Murphy, 2012). Such a split is more desirable as the class choice is more definite. We also need to think about how deep the tree should go. If the tree becomes too deep, it causes overfitting, which we want to avoid as discussed (Murphy, 2012).

3.2.2 Linking back to Random Forests

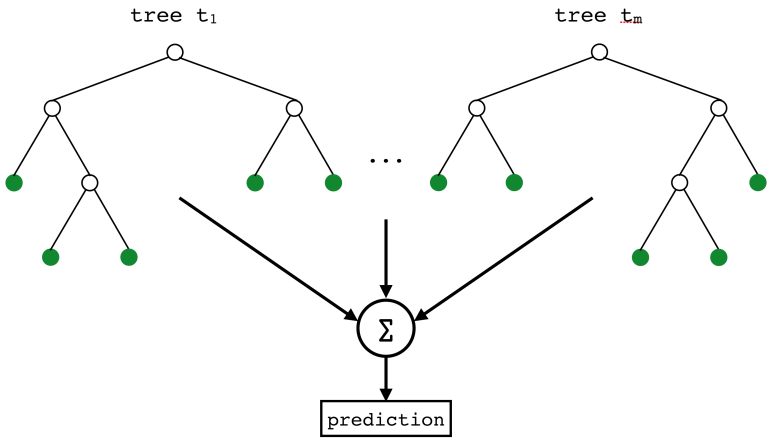

Figure 3.6: *The random forest model uses bagging to obtain a prediction by averaging the results from many decision trees.*

Decision tree is known to perform worse than other classifiers due to the instability of the trees. These trees are called high variance estimators, as their structures are affected greatly by small changes. A solution is to use Random Forests.

3.2.2.1 Bagging

$$f(x) = \sum_{m=1}^{M} \frac{1}{M} f_m(x) \tag{3.8}$$

Bagging attempts to reduce the variance by averaging the results from many noisy but approximately unbiased trees, created by a randomly selected subset of datapoints. Each tree in the forest casts a vote to decide on the predicted class through averaging the results of the trees in the forest, as illustrated in Equation 3.8.

However, simply growing multiple trees with different sets of data would create highly correlated trees, which curbs the amount that the variance can go down. Random forests attempt to avoid this by not only randomly selecting subsets to be used, but also the features for building each tree.

The randomness of the dataset for each tree may introduce extra bias in the forest. But, the averaging of trees usually decrease the variance, which results in a better model.

3.3 Boosted Trees

Boosted decision trees (BST-DT) is the best performing classifier according to Caruana and Niculescu-Mizil's (2006) comparisons, surpassing both decision trees and random forests in terms of misclassification error. In this section, we will look into the technical details of how boosted trees work.

A notable example is AdaBoost, which we will discuss in this section. But first, we shall look at the concept of boosting.

3.3.1 Boosting

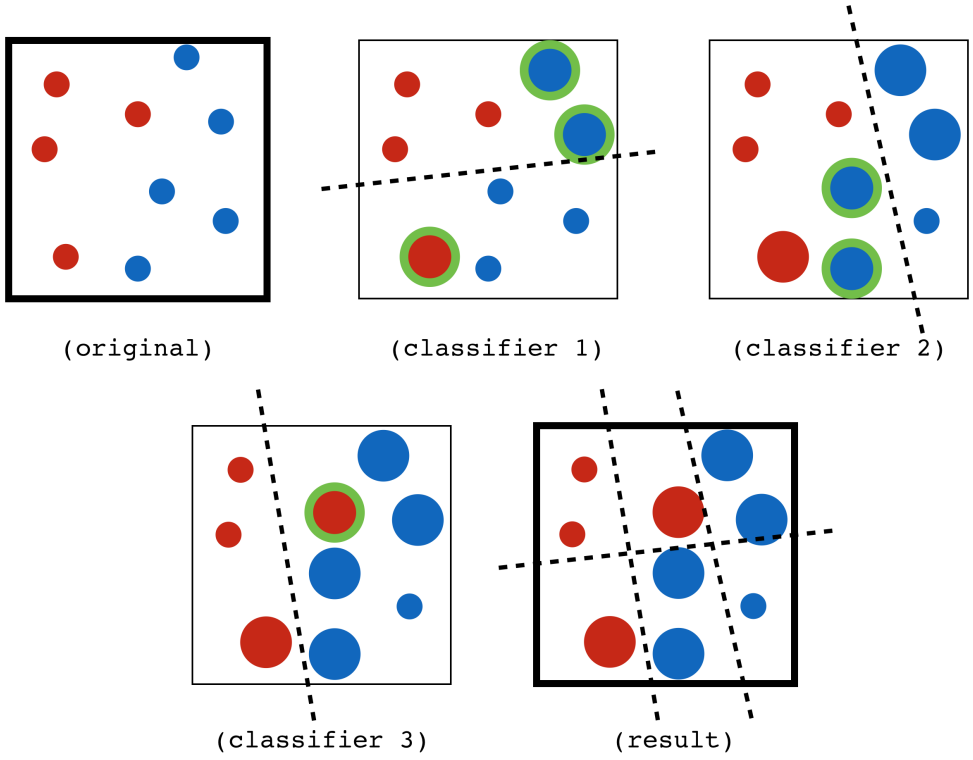

Figure 3.7: *Demonstration of how boosted trees function. For each iteration, misclassified points are given a higher weight (points with a green border). This new dataset is then used to train the next weak classifier, eventually resulting in a perfectly segmented classification space. (Adapted from Lazebnik (2016))*

Commonly used in conjunction with decision trees, boosting attempts to boost the performance of weak learners (ϕ_m in Equation 3.6) by increasing the weight of misclassified datapoints after training each weak learner (Schapire and Freund, 2012). The next weak learner will use the newly weighted dataset from the previous weak learner.

One criterion for this to function properly is that these trees must perform better than random (e.g. if we have 500 classes, the classifier needs to have an accuracy score greater

than $1/500 = 0.002$). Also, boosting is very resistant to overfitting (Murphy, 2012).

$$\min_{f} \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i)) \tag{3.9}$$

Boosting aims to minimise the loss function across all N samples as illustrated in Equation 3.9. Here, we consider a binary classification with $y \in \{0, 1\}$, $L(y, \tilde{y})$ being the loss function and f being an ABM model.

$$f^*(\mathbf{x}) = \underset{f(\mathbf{x})}{\operatorname{argmin}} = \mathbb{E}[Z] \tag{3.10}$$

In order to optimise for Equation 3.9, we need to find the optimal estimate for f, i.e. f^*. Formally, we express this as Equation 3.10, where Z is the derivative of the loss function. Note that $\mathbb{E}$ means an estimation.

$$f_0(\mathbf{x}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i; \gamma)) \tag{3.11}$$

$$(\alpha_m, \gamma_m) = \underset{\alpha,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(\mathbf{x}_i) + \alpha_m \phi(\mathbf{x}_i)) \tag{3.12}$$

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \alpha_m \phi(\mathbf{x}_i; \gamma) \tag{3.13}$$

This is a hard optimisation problem and should be dealt with sequentially with the **method forward stagewise additive modelling** (Murphy, 2012). We start by assigning the loss function of the chosen algorithm as the initial estimate by Equation 3.11. Then, we iterate for M times. At each m, we compute α_m and γ_m by Equation 3.12 and use them to evaluate f at m by Equation 3.13, without changing any parameter at previous m's. We can *early stop* once performance drops, even if we haven't reached M iterations. By combining f_m's, we will find the resultant strong classifier.

Note that each loss function computes Equation 3.12 differently (Murphy, 2012). In large, there are four types of loss functions used in various boosting algorithms, notably, squared error for L2Boosting, absolute error for gradient boosting, exponential loss for AdaBoost and logloss for LogitBoost.

Now that we have seen the general details of optimising the boosting problem, we shall see how this applies to AdaBoost.

3.3.2 AdaBoost

$$f(x) = \sum_{m=1}^{M} \alpha_m \phi_m(\mathbf{x}) \tag{3.14}$$

AdaBoost adapts the boosted algorithm by increasing the weight of each misclassified point at each iteration. Usually, deicsion trees are used as weak learners, which segment into areas and use weighted datapoints to improve accuracy, as illustrated in Figure 3.7. Equation 3.14 shows how weak learners are combined to form a strong learner $f(x)$, which follows with the foward stagewise additive modelling method.

There are two AdaBoost algorithms - **discrete AdaBoost** (`SAMME` in `scikit-learn`) and **continuous AdaBoost** (`SAMMME.R` in `scikit-learn`). Discrete AdaBoost, illustrated below, uses binary class labels from weak learners, whereas continuous AdaBoost uses probabilities instead (Murphy, 2012).

$$L_m(\phi) = \sum_{i=1}^{N} w_{i,m} \exp(-\alpha y_i \phi(\mathbf{x}_i)) \tag{3.15}$$

Again, we consider a binary classification problem here for simplicity. It can be generalised for multiclass problems easily (Murphy, 2012). At each iteration m, the goal is to minimise the loss function for the corresponding weak learner, ϕ_m, which requires the learning rate parameter, α_m.

Here, $w_{i,m}$ is the weight applied to the dataset i and $y_i \in \{-1, 1\}$ as the two binary classes. We use {-1, 1} because it makes it possible to use the sign function to obtain the class labels for discrete AdaBoost.

3.3.2.1 The Learning Rate Parameter (α)

$$\alpha_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m} \tag{3.16}$$

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{I}(y_i \neq \phi_m((x)_i)}{\sum_{i=1}^{N} w_{i,m}} \tag{3.17}$$

α is the main parameter of AdaBoost. It manages how influence each weak learner is to the resultant classifier, based on the error rate, as illustrated in Equation 3.16 (Schapire and Singer, 1999). We want to minimise this while to avoid overfitting. By using weight and the weighted samples, we obtain the error rate, as illustrated in Equation 3.17.

3.4 K-means Clustering

Recall that we briefly looked at unsupervised classifiers in section 2.3. Rather than learning from a labelled dataset, the algorithm compares values between points and creates K groups, or clusters, in a multi-dimensional spacei, as specified by the user.

Later on, in Section 4.1.2, we will look at how K-means clustering is utilised to downsize the training dataset so that we can perform tests with the aforementioned classification models. But first, let us understand the mathematical basis of K-means clustering.

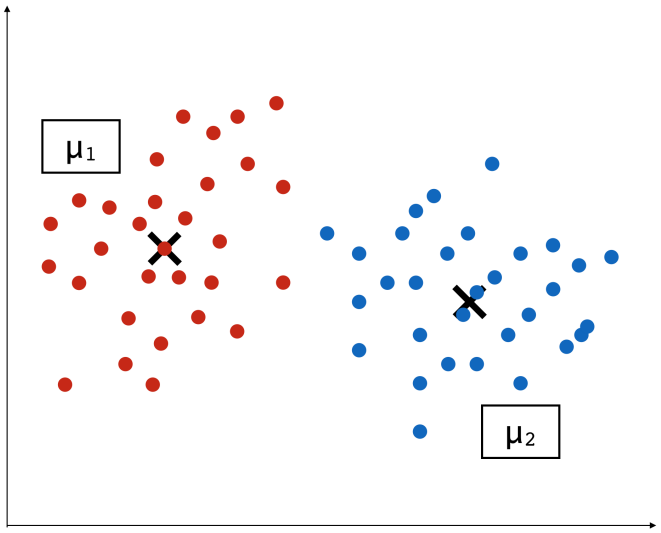

Figure 3.8: *An example of two clusters found with centroids μ_1 and μ_2 using K-means clustering.*

The algorithm aims to split the data into K clusters where the differences of distances between the points in the group is smaller than that to the points outside of the group.

Formally, we have to minimise the distortion measure, shown as Equation 3.18, by finding the appropriate $\{r_{nk}\}$ and $\{\mu_k\}$, and through the sum squared differences between each datapoint and each μ_k. For each point $\mathbf{x}_n$, r_{nk} is a set of binary values indicating which cluster the point belongs to. μ_k is a set of vectors associated with the k^{th} cluster, representing the centres of each cluster.

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||\mathbf{x}_n - \mu_k||^2 \tag{3.18}$$

Equation 3.18 can be posed as a two-stage optimisation problem. We first find r_{nk} by fixing μ_k (the expectation stage, **E**) and vice versa (the maximisation stage, **M**):

- **Stage 1: Finding Optimal r_{nk}**

$$r_{nk} = \begin{cases} 1 & \text{if } k = \text{argmin}_j ||\mathbf{x}_n - \mu_j||^2 \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

Staring with some randomly selected datapoints from the dataset as μ_k, we compute the sum squared differences between each point and the proposed cluster centres. We assign each datapoint to its corresponding closest cluster centres. In the end, we will have a set of datapoints that belong to a cluster centre, hence finding groups in the data.

- **Stage 2: Finding Optimal μ_k**

$$\begin{aligned} 2\sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \mu_k) &= 0 \\ \mu_k &= \frac{\sum_n r_{nk}\mathbf{x}_n}{\sum_n r_{nk}} \end{aligned} \tag{3.20}$$

These potential clusters might not be optimised to minimise Equation 3.18. So, We have to do so by computing the derivative with respective to μ_k as illustrated in Equation 3.20. Also, notice that by finding μ_k, we find out that they are the means of all the datapoints assigned to them, i.e. the centroids of the groups.

The two stages are repeated until the model converges, meaning that no further reassignment happens. The end result will be K groups governed by the mean values of their corresponding cluster. Note that these means may not necessarily be concrete datapoints from the dataset, but the actual mean of the corresponding clusters.

Recall in section 2.3.2, we mentioned about the Expectation Maximisation algorithm. Usually used in conjunction with DBSCAN, this can be used by most clustering problems like K-means as we discussed above.

3.5 Summary

So far, we have learnt that SVM contains a main parameter, C, which controls the trade-off between the margin and misclassification. With an RBF kernel, we can control the shape of the decision boundary through the γ parameter. By finding the appropriate parameters, we can obtain an optimal SVM classifier that gives the best possible accuracy with the given dataset.

Also, we have learnt how the combination of randomised data and features in random forests give a much better accuracy and prediction power than a single decision tree. Comparing with SVM, we have learnt that random forests can be trained faster. Rather than having to estimate the correct parameters for the base kernel, RF learns it themselves. However, we do have to choose an appropriate height of the tree and the number of the trees to train to avoid overfitting.

Boosted algorithms such as AdaBoost use a collection of weak classifiers and weight up ill-classified datapoints to reduce the bias of the resultant classifier, whereas RF averages the results of more complicated trees over a random set of datapoints and features. This means that boosted algorithms are theoretically more efficient, as they require fewer and simplier trees.

We shall now discuss the overall process used for training these algorithms.

Chapter 4

Methodology

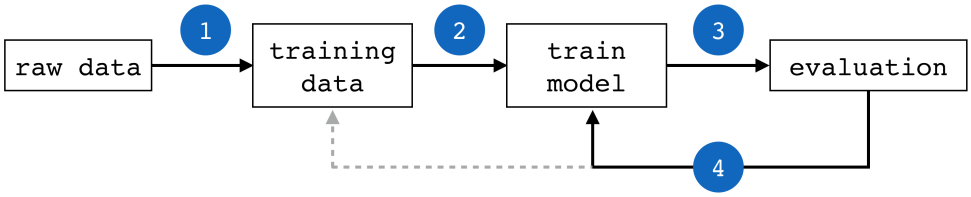

Figure 4.1: *The iterative process of building a resultant model for accurate prediction.*

Building a successful classifier requires careful considerations from the beginning. The quality of the dataset used is crucial to producing quality results. We would want to avoid the computer science analogy of 'garbage in, garbage out', in that given some poor input data, the results will be poor no matter what is done to it.

The NYU Depth Dataset, as discussed in section 2.2, provides a good base dataset to work with. Using this base dataset, we can transform it into our desired dataset format, which can then be used to train different classifiers.

In this section, we are going to discuss steps 1 and 2 in the iterative process, as shown in Figure 4.1. We will train and evaluate some classifiers in the next chapter. We will not explicitly describe how this is translated into code. Refer to the appendix for more information about the structure of the code (section A.1) and its documentation (section A.2).

4.1 Feature Engineering (Step 1)

Before we can run any supervised classification algorithm, we have to create a labelled dataset for it to learn from. Labels represent different classes of objects. Some examples of labels are 'chair', 'desk' and 'lamp'. For each label, we form data points, known as feature vectors, where each point contains information about an object belonging to that label. Much consideration is required to create a quality dataset that can produce a generalised classifier that can predict unseen data properly.

The NYU Depth Dataset provides a great starting point for creating the training dataset required for this project. It contains useful data of many scans at many different scenes. For instacne, these images are already labelled per pixel, which makes it easier to form our dataset. Here, we are assuming that this densely labelled dataset is correct, as we are going to measure the performance of our classifiers against this dataset.

4.1.1 Depth Patches as Features

Depth information of each image scan is used to form the feature vectors. In the NYU Depth Dataset, each pixel is given a depth value, which represents the distance between the camera and the real-world position in the scene (as discussed in section 2.1.3). A simple approach would be to use the depth value of each pixel as features. However, this would not form an effective classifier, as the distance of one point could not separate a label from another. For example, a red dot could represent many different classes of objects in the real world.

The solution is to obtain a patch around each pixel. This provides some context as to what that pixel might represent by taking its neighbours into account. However, the size of the patch should not be too small or too large. In both cases, it would be hard for the classifier to distinguish between labels due to too little information or the presence of too much noise. After all, machine learning algorithms attempt to create correct boundaries between data points so that they can be labelled as a class. Considering the size of the input images (640-by-480 pixels) and the sizes of objects in most scenes, a 15-by-15 pixels patch (7 neighbouring pixels at each direction) is chosen.

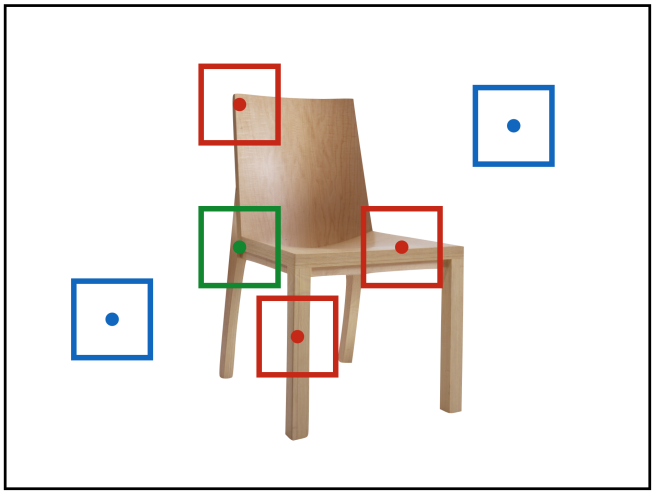

Figure 4.2: *Showing how each patch represents a window on the original image in this simplified view. Two classes, red and blue, are represented in this illustration. We shall illustrate the green patch in Figure 4.3.*

Each patch is then normalised by deducting the mean of the patch from each pixel. The aim is to remove the actual distance between the camera and the object, so that each patch is approximately independent of factors such as the angle of which the image is taken from and the actual distance between the object and the camera.

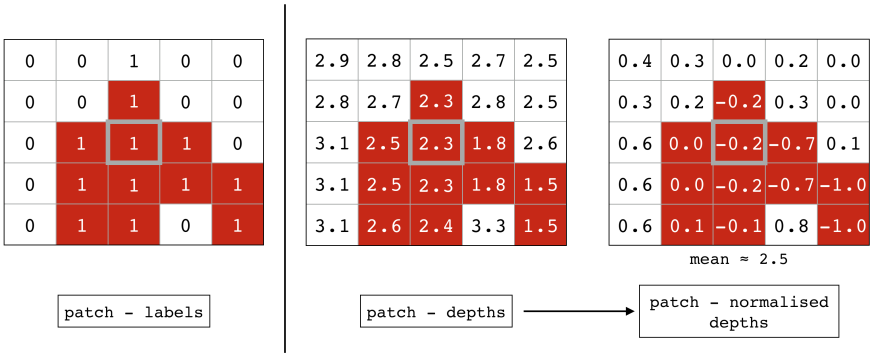

Figure 4.3: *An example of how we obtain a patch around a pixel (coordinate) and normalise it. Note that we are using* $15 * 15$ *patches rather than* $5 * 5$ *patches as shown in this figure.*

4.1.2 Reducing Number of Datapoints

Each image is composed of 640-by-480 pixels, meaning that more than 290 000 patches are generated from each image. If all 1449 images are used, 460 million patches are generated. Even more efficient classifiers such as AdaBoost would take a very long time to train on this unrealistically large dataset. Given the time and technical limitations, a manageable dataset is required.

Let us consider two methods:

- **Randomly Select Datapoints**
  A method for reducing the number of datapoints is to to collect a random number of datapoints from a random selection of images possessing the label. In this way, we aim to collect a vast set of data from a wide range of images. The drawback is that it is uncertain what 'random' entails. Importantly, some valuable and important features may be missed out, making the classifier less powerful.

- **K-means Clustering**
  A better way is to spot the similarities in the data and extract key interest datapoints. By finding groups within the data, known as clusters, it is possible to generalise each of them by its centroid. The obtained points provide a more reliable dataset, as it does not throw away potentially important information about a label.

  By doing this, we can be more certain that our dataset is representative. We want to represent datapoints and features across all images, rather than picking random points from a big pool of datapoints and risking missing out key features as we cannot guarantee these randomly picked points cover all images.

Due to the number of patches present in some of the labels and computational limitations, both methods are used to obtain the datasets. k-means clustering is directly applied to labels with at least 1,000 and at most 120,000 datapoints, so to compute within the six-hour time limit on Balena as a free user. A random 100,000 datapoints are selected for classes with datapoints more than the limit.

By running k-means on these classes, we obtain 1,000 datapoints for each of them. Classes with at most 1,000 datapoints are left as is to form a combined dataset.

A few exceptions require a random number of images to be chosen before performing the random pick and clustering, due to the amount of images containing those labels and the number of patches that are generated for them.

Number of Datapoints	**Operation**
$<$ 1,000	do nothing
$>$ 1,000 *and* $\leq$ 120,000	run k-means to extract 1,000 datapoints
$>$ 120,000	select random 100,000 points, then k-means to extract 1,000 datapoints
many images *or* $\gg$ 120,000	select random 250 images, select random 100,000 points, then k-means to extract 1,000 datapoints

Table 4.1: *Summarising methods used to extract a representative subset for a class.*

Figure 4.4: *Distribution of classes. (Blue) shows the distribution of our dataset before K-means clustering was used; (Red) shows the distribution after running K-means clustering on classes with more than 1,000 datapoints.*

In Figure 4.4, we see a wide variation in the number of datapoints between classes before we used K-means to reduce our dataset. As we mentioned, this causes some practical issues given the resource we have. Also, the imbalance of classes would introduce bias into the classifier, making it more favourable towards the larger classes. By performing K-means on the very large classes, we aim to create a representative dataset that limits the number of datapoints, hence reducing the bias.

4.1.3 Creating Datasets

Some Considerations

To train and measure the performance of the classifier, two datasets are created. The first dataset is the 'training dataset'. This dataset should be as large as possible to enable the classifier to learn the characteristics of the labels effectively. After all, we aim to obtain high recognition rate with the classifier. This dataset is further split into two parts - training and validation. The testing part enables the classifier to evaluate its performance.

A 'test dataset' is created to measure a more realistic performance of the classifier. This dataset is not seen by the classifier during training and testing. It is important to avoid measuring performance of a machine-learning model with datapoints seen by the classifier. Otherwise, unrealistic prediction rates would be reported due to feature leakage (learning from *future* datapoints). To avoid feature leakage during any stage of the process, all of these datasets should be mutually exclusive.

The Datasets

After reducing datapoints to an amount manageable by k-means for some classes, we can create the three datasets by combining the datapoints of all the classes. This combined dataset is split into 'training and testing' and 'validation' in a 70%-30% split. Then, the 'training and testing' is split into 60% training and 40% testing. Here, we split the dataset using a *stratified* approach, in which we try to retain the original ratio between classes in each subset. The `StratifiedShuffleSplit` function in `scikit-learn` enables us to do this easily.

Some scenes are not used for any of the datasets above, so that they may be used as another way to assess the performance of the classifier in the end. We will attempt to predict some of these images later in chapter 5.

We lost some classes as a consequence of not using some scenes. The total number of classes in the final dataset is 875, compared to 895 classes in the raw NYU Depth Dataset. With closer inspection, we see that these classes only exist in images in the same scene but of different angles. They would be of limited usefulness as there will not be enough data to test if the classifier can recognise that class of objects generally across different scenes.

4.2 Training a Classifier (Step 2)

A classifier is then trained using our dataset. Although the datasets have been carefully crafted, there is no guarantee that the dataset would provide any useful result. Perhaps more features are required to provide enough information for the classifier to effectively split the labelled data correctly, or that our assumption that depth is a useful metric is ill-founded. We can only find this out by performing tests, which we will be doing in chapter 5.

One of the important tasks before training our final classifiers is to find the correct parameters to enable optimal prediction power.

4.2.1 Optimising Parameters

Each classifier has its own tunable parameters to achieve optimal results.

We will attempt two methods for finding an optimal set of parameters. The first way is to use exhaustive search with a subset of the data with less classes. If these parameters become stable for a few relatively large subsets, we would assume this works best in general. The next best option is to perform a randomised search by training classifiers with a randomly selected subset of parameter combinations. Note that this is bounded by the training time required.

Recall the parameters required to create useful classifiers of the corresponding algorithms:

- **Support Vector Machine**
  As discussed in section 3.1, the SVM algorithm requires a few optional but influential parameters depending on the chosen kernel. Cost (C) is required to be tuned across all kernel. If the selected kernel is RBF, the coefficient of the kernel (γ) has to be tuned as well to achieve optimal results.

  SVM is known to be very slow at training, so considerations have to be taken when finding these optimal parameters and training the resultant classifier.

- **Random Forest**
  We need to obtain the optimal set of parameters to ensure that the forest does not overfit. There are numerous parameters required to be specified as discussed in section 3.2. At the individual tree level, this includes maximum depth, maximum number of features and the minimum number of samples to allow for a split. At the forest level, we need to optimise the maximum number of estimators (the number of trees to be built).

  As Random Forest is a much more efficient algorithm, we expect that the classifier will converge much quicker than SVM and be able to handle all the classes and data. Similarly, we could at least manually train a few classifiers with different parameters to find out the optimal settings.

- **AdaBoost**
  The main parameter for AdaBoost, as described in section 3.3, is the learning rate parameter (α). It controls how much each weak learner contributes to the strong final classifier. We also need to choose whether to use the discrete or real alogirhtm.

  A less important but useful parameter is the maximum number of weak learners to train. It is less important as the algorithms can early stop when a perfect fit is obtained before the end.

  AdaBoost is known to be even more efficient than Random Forest, as it uses weaker learners and requires fewer estimators of less depth (Schapire and Freund, 2012).

4.2.1.1 Methods for Finding Optimal Parameters

An **exhaustive search**, also known as grid search (`GridSearchCV` in `scikit-learn`), can be used to find the best parameters for classifiers that supports them. It trains classifiers on all combinations of a given list of testing parameters. It returns the parameters that produce the best score. Cross validation (defaults to 3 folds) is used to produce this score for comparison.

Sometimes, a classification problem is too big that it is not possible to perform an exhaustive search. One solution is to run many classifiers with each combination of testing parameters manually. However, this is a time-consuming task which uses a lot of computational power.

In such cases, a **randomised search** can be performed (`RandomizedSearchCV` in `scikit-learn`). It fits classfiers with a randomly chosen set of parameters from the list of testing parameters. The number of randomly chosen sets is user-specified. This is to enable the user to control the scope. Similarly, cross validation is applied to obtain the score for each chosen set of parameters.

It is obvious that there is a trade-off of quality of parameters and speed between grid and randomised search. To ensure the results are representative when performing randomised searches, one can run the randomised search for a few times and then obtain the best results.

Later in chapter 5, we are going to use randomised search for some classification problems. We are going to obtain 5 randomised sets per search, and perform this search 3 times, and take the parameters from the test that gives the highest accuracy score.

The next step is to train some SVM and Random Forest classifiers with our dataset.

Chapter 5

Results

Recall that our goal is to find out if depth data is a useful dataset for classifying object classes. In this chapter, we attempt to obtain the best possible results through optimising their corresponding parameters and discuss how different classifiers might fit for our problem. We will look at these classifiers:

- Support vector machine with a linear kernel (SVM (linear))
- Support vector machine with a RBF kernel (SVM (RBF))
- Random Forest (RF)
- discrete AdaBoost (ADA)
- discrete AdaBoost (ADA.R)

For clarity, we will be referring to these classifiers by their acronyms most of the time.

Broadly speaking, we have to perform the following tasks:

- finding the range of potentially appropriate parameters using smaller subsets of our dataset;
- training the 'out-of-the-box' classifiers with the whole training dataset without parameter optimisation; and,
- optimising parameters for the highest achieving classifier and evaluate it through precision-recall and visualising the predicted images.

Optimising parameters could make a big difference, especially for SVM. For instance, in one of the SVM parameter searching operations, one set of parameters obtained a cross-validated accuracy of more than twice of another set. Let us now look at how unoptimised classifiers perform with our dataset.

5.1 Finding Appropriate Parameters

It is a challenging task to find the optimal parameters for a classification model. Ideally, we want to run an exhaustive search over all possible parameter values for each classifier. Realistically and taking into account the constraints of Balena, this task is more difficult than it appears.

To simplify the problem, we will look at how these parameters change as the number of classes increases. For each algorithm, we perform grid or randomised search on the first 2, 5, 10, 50 and 100 classes to find the parameters that produce the best cross validation score. Note that these are purely patches from some specified number of classes. We want to find out how the algorithms perform in a binary situation (2 classes), small multiclass situations (5 and 10 classes) and large multiclass situations (50 and 100 classes).

As more classes are used, we expect the accuracy score to fall, due to the added complexity of the increased number of classes and the distributions of them. Their results can help us decide if we should continue to pursue these algorithms, and the range of parameters we should focus on when evaluating the algorithms using the whole dataset.

Recall that the accuracy score is the ratio between correctly classified datapoints and the whole dataset. Generally speaking, the higher the score, the better the classifier performs. We will use 3-fold cross validation to find an average accuracy score, which we will refer to as cross validation score.

Note that we are purely taking datapoints from some number of classes rather than considering how it fits with classifying datapoints in the context of an image. We will do so when we have our final classifier trained in section 5.3. Thus, our aim here is to narrow down the range of values we have to search for, and to get a sense of how these algorithms scale as the number of classes increases, before digging into the whole dataset.

No. of Classes	**SVM (Linear)**	**SVM (RBF)**	**RF**	**ADA (SAMME)**	**ADA (SAMME.R)**
2	61.0%	64.8%	**70.1**%	67.5%	67.1%
5	35.7%	38.7%	**44.0**%	39.8%	40.6%
10	32.1%	34.4%	**44.1**%	31.6%	31.9%
50	13.1%	18.9%	**33.8**%	9.94%	10.9%
100	10.9%	15.6%	**33.6**%	5.17%	7.47%

Table 5.1: *Best accuracy scores found using grid search with 3-fold cross validation or the best of 3 randomised searches with 3-fold cross validation for various classifiers – (in order) SVM with linear kernel, SVM with RBF kernel, RF, discrete AdaBoost and continuous AdaBoost.*

Random Forest

Table 5.1 shows the cross-validated accuracy scores of our chosen algorithms. We can see that RF outperformed all other classifiers. Not only did it give the best scores across different number of classes, it also scaled the best with our dataset where its accuracy dropped only by about 10% when going from 10 classes to 50 classes and stabilised thereafter, compared to around 20% with other algorithms and seeing a continual drop with more classes.

RF worked well with our dataset with its ability to locate outliers and find subtle relationships within the input features.

AdaBoost

On the other hand, even though boosted decision trees such as ADA and ADA.R were found to be the top performing classifier by Caruana and Niculescu-Mizil (2006), it performed the worst with our dataset. This could reflect the composition of our dataset. We would expect our dataset to be closely packed together due to the nature of the data and the normalisation method we used. This also means that the data could be 'noisy' and with outliers, which is a weak point of AdaBoost algorithms.

ADA.R is known to require fewer trees to achieve a similar or even better accuracy score than ADA. In our parameter searches, we see that ADA used a maximum of 1000 decision trees to compose its final classifier, compared to a maximum of just about 100 trees for ADA-R throughout. This means that if we were going to use an AdaBoost classifier, we should pick the ADA.R algorithm.

Support Vector Machine

Like many complex problems, our dataset performed better with SVM (RBF) than SVM (linear), especially with many classes. However, the scores decreased more quickly when more classes were used to train an SVM (RBF) than SVM (linear).

It appears that SVM could not find a good balance between misclassification and being specific, due to the complexity of our dataset. Also, it took much longer than RF and ADA's to search on the same number of parameters.

5.1.1 Overview

These tests show some interesting characteristics about classification problems in general. We expect binary classification to perform well, as it is a rather simple problem bounded by few factors. As we moved on to multiclass classification (from 2 classes to 5 classes in our case), the scores dropped rapidly. This shows that multiclass classification gets complicated quickly. When many more classes were used (from 10 to 50 classes for example), the scores dropped rapidly again. The added variance and noise made it even harder for the classifier to distinguish between their subtle characteristics.

Another interesting characteristic is that classifiers have similar or even stronger predictive powers as the number of classes increases, even though the accuracy rates decreases. This is particularly clear with the SVMs and RF results as shown in Table 5.1.

Take the accuracy scores of RF as an example: it is 20.1% better than random when it is trained on 2 classes. More impressively, it is 32.6% better than random for 100 classes, which is a 12.5% increase despite a 36.5% decrease in the accuracy score. Hence, in-depth evaluation is important in understanding the characteristics and performance of classifiers.

The complexity of our dataset means that RF seems to best suited for our problem amongst the other two algorithms. It produced the most consistent results and scaled very well with classification results due to its robustness to outliers and noise. Also, we saw that SVMs are more resilient to noise than AdaBoost algorithms.

For now, we should continue to pursue all classification models mentioned here, as we need more concrete information before we can rule out their usefulness to our problem.

5.2 Unoptimised Classifiers

From now on, we are going to examine how effective these algorithms are by using the whole training dataset. Recall that our training dataset contains 875 classes, which was created using K-means clustering. With our analysis in the previous section, we expect to see RF to remain rather stable and be the most effective classifier with our dataset.

Classifier	**Cross Validation Score**
SVM (Linear)	–
SVM (RBF)	–
RF	28.7%
ADA	2.33%
ADA.R	3.97%

Table 5.2: *3-fold cross validation scores with unoptimised classifiers.*

In fact, RF remained rather stable, achieving an accuracy score of 28.7%, with only a small drop in the score when compared to the score of 100 classes. Although ADA and ADA.R performed much worse than RF, they still performed better than random ($1/875 = 0.0114\%$). Again, the complicated and closely-packed dataset makes it difficult for AdaBoost algorithms to be of any use.

We attempted to run the SVM classifiers on the whole dataset but it did not converge within the time constraints of Balena. This is unfortunate, as it would be a good

comparison to see how SVM classifiers fair against RF.

Thus, we will focus on optimising and evaluating an RF classifier as our final classifier.

5.3 Evaluating the Best Classifier

The fact that our training dataset is rather big – containing about 390 000 datapoints across 875 classes in our training set – makes it difficult to work with. Large exhaustive searches with cross validation and many parameter candidates would not converge under the time and memory constraints on Balena.

Our workaround is to run a few models with different parameters informed by the optimal parameters obtained in our previous tests. We then compute a 3-fold cross validation score for each of these models and predict the test set with the corresponding trained model. The score will provide us with some confidence about the performance of the classifier.

Following on, we should look into the precision-recall rates to understand how these classifiers actually perform. Remember that precision-recall aims to find out the predictive power and sensitivity of a classifier. We will also attempt to predict some images with our final classifier to see how it works.

Optimising parameters

Recall that we want to minimise the number of samples for a split to occur (`min_samples_split`), maximise the depth of each tree (`max_depth`) and maximise the number of trees used (`n_estimators`). We also want to decide on how we want to weight our classes (`n_weight`).

Name	**min_ samples_ split**	**max_depth**	**n_estimator**	**class_weight**	**cross val score**
Classifier 0	2	–	–	balanced	25.8%
Classifier 1	2	40	40	None	39.3%
Classifier 2	2	40	40	balanced	42.2%
Classifier 3	2	45	45	balanced	41.6%
Classifier 4	2	60	60	balanced	42.5%
Classifier 5	2	100	100	balanced	43.8%

Table 5.3: *Random forest models with various parameter settings and their 3-fold cross validation scores.*

Given our knowledge from previous tests, we expect similar or even better performance from RF than its unoptimised version, if we tune the parameters properly. In fact, as seen in Table 5.3, we found that depth and the number of estimators in the forest are the key in obtaining a good RF classifier. Classifier 2 was 13.5% better than classifier 0 which had none of those parameters set.

Also, if we supply an insufficient set of parameters, we risk making the classifier worst than its 'out-of-the-box' state, as we demonstrated with classifier 0 (see Table 5.3) and the unoptimised RF classifier (see Table 5.2).

We are only testing some parameters here as the constraints on Balena mean that we could not test a higher maximum depth or number of estimators. Classifier 5, with the highest values for `max_depth` and `n_estimators`, performed the best with a cross validation score of 43.8%, which is 15.1% better than an unoptimised RF classifier. We are going to use this as our 'resultant classifier' from now on.

We also found that a 'balanced' weighting (i.e. a weight that is inversely proportional to the class frequency in the input data) gave better results for our unbalanced dataset (shown in Figure 4.4) than no weighting at all. The small increase suggests that the RF classifier is rather resilient to an unbalanced situation. Nonetheless, a balanced weighting produced better results.

Name	**cross val score**	**test score**	**time taken to predict (min:sec)**
Classifier 0	25.8%	34.9%	0:57
Classifier 1	39.3%	44.9%	2:29
Classifier 2	42.2%	46.7%	2:48
Classifier 3	41.6%	45.6%	3:55
Classifier 4	42.5%	48.0%	3:38
Classifier 5	43.8%	49.0%	5:43

Table 5.4: *3-fold cross validation and test scores for some random forest classifiers we ran.*

The influence of the depth of each tree and the number of trees in the forest is shown more clearly through predicting the test set in Table 5.4.

It is worth noting that while classifier 5 is the highest achieving classifier, the performance improvement is marginal. However, the increased number of trees and maximum depth for each tree in the forest when compared with that of classifier 4 (100 versus 60 for both parameters) mean that the classifier became more complex, causing it to use up more memory (more than 128 GB of RAM is required) and longer to perform prediction on the test set. Here, we focus on the performance of the classifier, so we will continue to use classifier 5 as our preferred classifier.

Precision and Recall

We obtained a precision of 51% and recall of 49% on our test dataset with classifier 5. Here, precision is about 'how many of the selected items are relevant' and recall is 'how many relevant items are selected'. In other words, our classifier is able to pick up 51% of the whole set of the test datapoints, of which it correctly classified 49% of them.

This is in-line with our expectation of the classifier through our tests. We expect our classifier to be accurate no less than 40% of the time. Give the complexity of our training dataset, RF is holding up to the challenging of noise and outliers, and appear to continue to perform well with a large multiclass classification problem.

Predicting Images

So far, the evidence we gathered seem to suggest that our approach is performing well. We should attempt to predict some images with our best classifier, classifier 5, to see if it would work well with out dataset.

Not dissimilar to how we approached our training and testing stages, we are going to predict each pixel patch and assign a colour for its corresponding pixel based on its prediction. By predicting all pixel patch of an image, we can combine them into a resultant image, and examine if it is able to resemble the scene.

We performed three image tests – an image from a bathroom (Figure 5.1), another from an office (Figure 5.2) and one from a bedroom (Figure 5.3).

We observe that the precision rates varied. While the bathroom scene performed the best with a precision rate of 47%, the bedroom scene obtained a precision of 17%. Flat surfaces with little variation, such as the walls in the bedroom and the tables in the office scene, did not perform very well. The little variance between surrounding areas make it difficult for the classifier to decide what they actually are. A flat surface could potentially be classed as anything.

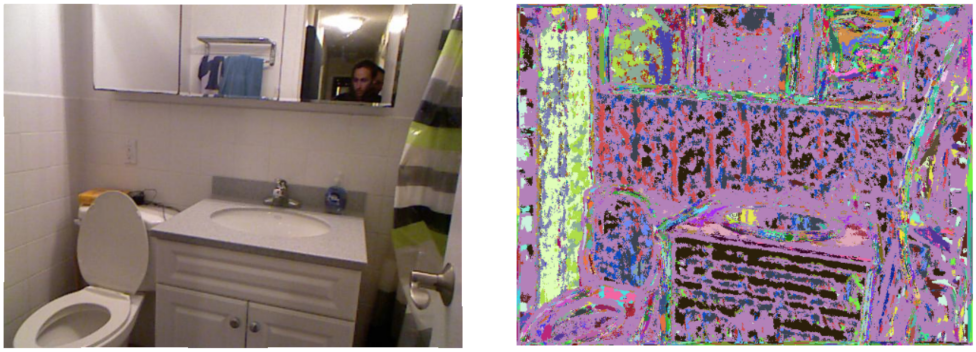

Figure 5.1: *(Left) a bathroom scene from the NYU Depth Dataset; (right) our prediction.*

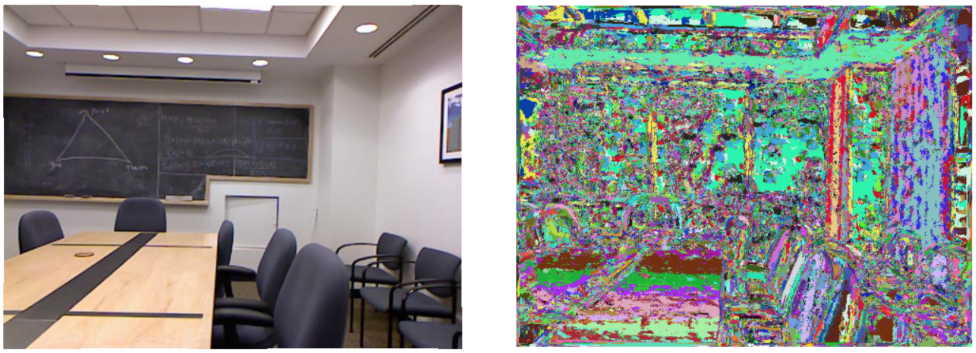

Figure 5.2: *(Left) an office scene from the NYU Depth Dataset; (right) our prediction.*

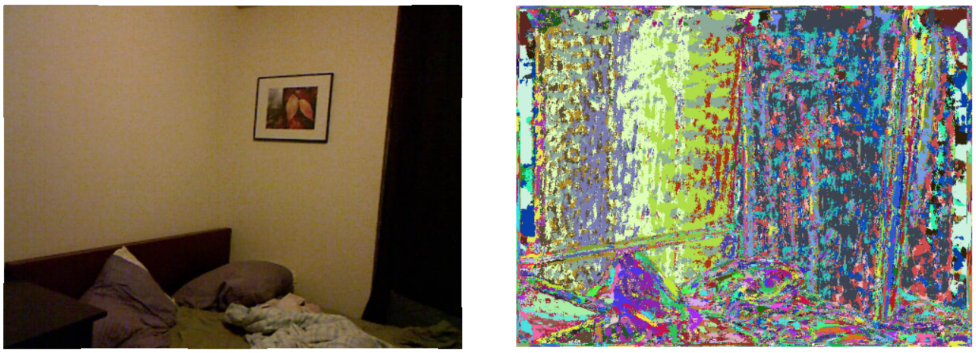

Figure 5.3: *(Left) a bedroom scene from the NYU Depth Dataset; (right) our prediction*

Again, this illustrates the complexity of such a classification problem. On one hand, we appeared to have trained a good enough classifier through the majority of our testing. On the other hand, not until the last hurdle when we put the classifier into application could we evaluate if it is performing well.

Chapter 6

Conclusion

Even with the help of a sophisticated and easy-to-use machine-learning library, the process of producing an accurate and precise classifier requires much consideration.

The first hurdle comes from feature engineering. It is crucial to obtain a well-defined dataset so that classification methods can generalise the dataset and learn about the features about different classes. Thorough experimentation is then required to understand which algorithm performs the best with our dataset, and to make sure it is not just a coincidence that it appears to work.

Recall our goals of this project: we aim to explore if we can use depth to accurately predict classes of objects in an image using depth information with supervised classification models, and how to proceed with a large, real-world classification problem.

We successfully trained a random forest classifier, giving a cross validation score of 43.8%, precision rate of 51% and recall rate of 49%. However, it is shown to be an over-estimate when it is used to predict test images. In this chapter, we will collate what we have learnt about classifiers and depth data for classification, and discuss in more detail what we have achieved with our classification problem within the scope of this project.

6.1 Achievements

Overview

In chapter 3, we learnt about the characteristics of three supervised classification methods, namely, **support vector machine** (a kernel method), **random forest** (an ensemble method) and **AdaBoost** (a boosting method), for which Caruana and Niculescu-Mizil (2006) and Amancio et al. (2014) found to perform the best in baseline tests. We also looked into the unsupervised technique, K-means clustering to facilitate our feature engineering. Knowledge about these classifiers have also informed our decisions on training and testing.

Effectiveness of Different Classification Algorithms

Through our tests, we learnt that AdaBoost algorithms did not perform well especially when there are many classes in the data with our complex and closely-packed dataset. Our tests clearly demonstrated the weaknesses of AdaBoost algorithms, that they are prone to noise and outliers. With the preliminary test results shown in tables 5.1 and 5.2, we were able to rule out AdaBoost algorithms for being useful with our dataset or other datasets with similar set-up.

In our 'different number of classes' test (Table 5.1), we found out that support vector machine (SVM) classifiers did not scale well as we increased the number of classes to be used in training. Neither a linear nor RBF kernel could define effective decision boundaries to separate the data.

They also took significantly longer to run due to the number of binary classifiers it needs to build behind-the-scene. It was soon realised that this weakness became problematic when we attempted to train on our complete training dataset. No SVM classifier could converge under the time constraints of Balena. Even though we were not able to compare the performance on SVM on the whole dataset empirically, the evidence showed in Table 5.1 suggests that it would not result in a classifier with good enough quality.

Random forest was found to be the most resilient to our dataset, which displayed its ability to handle complex datasets. We saw small decreases in cross validation scores as the number of classes increases in our 'different number of classes' test. In fact, we were able to achieve higher cross validation scores on the whole dataset with some parameter tuning than this test.

Performance of the Best Classifier

Our best model was produced using the random forest algorithm. It achieved a cross validation score of 43.8%, test score (score obtained by predicting the test set) of 49%, precision rate of 51% and recall rate of 49%.

In hindsight, one might suggest that this is not of high accuracy. However, we have to bear in mind that a classification problem in the computer vision domain is inherently difficult due to the complex distribution of the underlying data. The ability to predict better than random would have been a good achievement. Here, our classifier showed to be performing much better than random – in fact, 43.7% better.

Parameter tuning is key to achieve the potential ability of a classification algorithm. As seen in Table 5.3, a well-tuned classifier (classifier 5) performed 18% better than a poorly-tuned classifier (classifier 0) in terms of cross-validated accuracy.

Our classifier is heading towards the right direction, obtaining good accuracy values. As seen in our tests, figures 5.1 to 5.3, our classifier performed much better than random. Even if it did not perform as well as the precision-recall values suggested (which is expected), we can still observe the outline of the original scene, and in some cases, recognising a continuous area in the scene, such as the that of the bathroom scene (Figure 5.1).

Again, this shows the complexity of creating an effective classifier. Empirical evaluations can only provide an estimate of how the classifier would perform. Much application evaluations is required to provide a full picture of how the classifier perform in reality.

6.2 Other Lessons Learnt

Although random forest classifiers are fast to train (100 estimators with a maximum depth of 100 on the training dataset took about 20 minutes to train, compared to unconverged SVM classifiers), they use up a large amount of memory. Memory requirement increased significantly as we attempted to build a forest with more estimators or deeper trees. In fact, we found out that it required more than 512 GB of RAM with `scikit-learn` and Python to build a random forest with more than 150 trees.

Our feature engineering approach with K-means clustering took a long time to be performed. K-means, whilst appeared to have successfully given a representative subset of the originally very large dataset, took a long time to converge. In fact, we were only able to obtain centroids with a dataset of at most 120,000 datapoints to compute within the constraints of Balena.

While we emphasise on accuracy, time and computer resources evidently became the bottleneck for some of our experiments. For instance, we were unable to obtain an SVM classifier on the whole dataset, and were unable to run random forest classifiers with more than 150 estimators as it consumed too much memory.

6.3 Future Work

Building on top of this encouraging result, there are more that we can do to find a classifier or classification algorithm that works well with our dataset.

In this project, it is fortunate that our approach appeared to have worked. Random forest is found to be the best classifier for this category of problem through our tests. Much work can be

More tests should be performed. We could compare the effects different sizes of patches have on the resultant classifier. In our feature engineering stage, we hypothetically decided that 15x15 patches would give us a good balance between complexity and noise for our dataset. We could only justify that we were heading to the right decision through such comparisons. Also, we could perform more parameter tuning. As we have seen in Table 5.3, using appropriate parameters would enable the true potential of the given model.

Moreover, we could introduce new features such as orientation and image (RGB) information to complement the patches. Before doing so, much consideration is required as more features may not mean a better outcome but noise.

Few scenes in the NYU Depth Dataset have scans from different angles. Inspired by SemanticPaint (Valentin et al., 2015) which continuously learn from user defined objects and 'paint' recognised objects with the same colour as the user explores the area, we can create scans that covers different angles of a scene to achieve a good classifier. With

different angles, we will be able to learn from more distinctive features of different objects in the same class, and combine this knowledge about the scene using ensemble methods not dissimilar to random forest.

Furthermore, we could obtain better predictions by taking into account the information of the surroundings. As mentioned, Silberman et al. (2012) used support inferences to improve their segmentation classification problem. Together with the classifier, we could vote on the most probable class by incorporating the predictions of surrounding patches, so that outlying prediction can be 'reverted' to the correct class.

We should also attempt other sampling techniques to see if they could obtain a better representative dataset more efficiently. For example, Markov Chain Monte Carlo (MCMC) is a more sophisticated and popular algorithm for machine learning-related data sampling tasks.

Lastly, we could experiment with clustering techniques to see how they compare with supervised classification techniques. Clustering techniques K-means are used in computer vision for class or instance recognition.

6.4 Final Thoughts

A classification problem is more difficult than it may seem. There are many factors contributing to the success of a classifier, and it is a challenge to have consider all possibilities.

In this project, we looked at three different classification models in detail and attempted to create classifiers with depth information. We then discussed our approach and performed tests to attempt to find the best classifier within the scope of this project. We shared some interesting findings where much more can be done in search for the best classifier utilising depth information to predict classes of objects.

Appendix A

Python Script Documentation

A.1 General Descriptions

There are four main scripts to perform the tasks required for this project:

- `extract.py`
  The script extracts the raw data from the HDF5 binary data format into more Python friendly objects, such as `numpy` arrays (as discussed in section 2.5) and dictionaries (maps between unique names and some values). This facilitates feature engineering and classification tasks using Python and `scikit-learn`. Number datasets can be extracted easily, while datasets containing texts and MatLab maps require more conversion due to the different representation used.

- `transform.py`
  The script contains functions that transform raw data into the final traininng, testing and validation datasets. By using the depth maps, the script creates normalised patches around each pixel of sizes specified by the user. Together with the class label information, these patches can be used by the script to generate a list of patches for a given class. Then, then random selection (self-implemented function) and K-means clustering (enabled by `scikit-learn`) can be performed, which can then be combined into one list, and then stratifiedly randomly split into the three datasets.

- `model.py`
  The script performs exhaustive and randomised parameter search, and model training. It uses the `scikit-learn` library to perform these tasks. With the script, classifiers of support vector machine, random forest and AdaBoost can be optimised and trained with input data created by `transform.py`. Also, it produces confusion matrices and precision-recall rates to measure the performance of the classifier.

- `prediction.py`
  The script is used to predict an entire image and store the predicted results as an

image. A generated colour dicitonary (generated once before this script is being used) assigns a unique colour to each label. This can be used to visualise the performance of the resultant classifier by comparing the results with the original image and segmentation image provided by the NYU Depth Dataset.

A.2 Technical Documentation

A.2.1 General Usage Guide

These scripts can be run on any system that supports Python. To simplify usage, part of the process uses command line arguments to provide arguments for functions within the code:

`python` *[script.py] [function] [-option1 value1 ... -optionN valueN]*

For each script except `extract.py`, there are different functions to run different parts of the code for each script. Each command line function argument contains various options to act as arguments for those functions in the code. The option `-h` or `-help` shows all available functions and options when the script is being called.

On Balena, a Bash-styled SLURM script is used to specify the properties of a job. This Python command forms the 'execution' part of the code. By using command line arguments, only one script has to be changed to perform different functions, making the process much simplier.

The Balena Wiki produced by the Universtiy provides useful guides and documentation about SLURM scripts.

A.2.2 Commands for Each Python Script

Script	Function	Option	Description
model.py	svc	-file	supply the filename of the input data
		-id	supply the filename for the output
		-k	kernel used by the classifier
	rf	-file	supply the filename of the input data
		-id	supply the filename for the output
	ada	-file	supply the filename of the input data
		-id	supply the filename for the output
		-alg	algorithm used - discrete (SAMME) or real (SAMME.R)
	gridsearch	-f	supply the filename of the input data
		-mdl	model to be trained (ada, ada-r, rf, svc-linear, svc-rbf)
		-m	specify to use do grid (gs) or randomised (rs) search

Table A.1: *Function and options for* model.py.

Script	Function	Option	Description
transform.py	patches	-fn	a patch-related functionality (per_pixel - obtain a patch for each pixel)
		-dim	specify the dimension of each patch
		-img_s	first image to be processed
		-img_e	last image to be processed
	rand	-ls	first label to be explored
		-le	last label to be explored
		-n	the number of random samples required
	lbl	-ls	first label to be explored
		-le	last label to be explored
		-d	dimension of patches
	pts	-n_init	the number of runs with different starting centroids
		-n_cluster	the number of clusters/ points to generate
		-ls	first label to be explored
		-le	last label to be explored
	combine	–	combine multiple files into one feature-target dictionary
	data	-f	filename of the input data
	merge	–	merge a given set of labels into one new label

Table A.2: *Function and options for* transform.py.

Script	Function	Option	Description
prediction.py	predict	-df	filename of patches to be predicted
		-mf	filename of the model to be used for predictions
		-save	save filename
		-s_flag	whether to save the prediction
		-t	whether to predict the test set or an image
	gen	-img	the image used for the predictions
		-p_file	filename of the prediction file
	precall-data	-p	filename of the predicted values
		-o	filename of original dataset
	precall-img	-p	filename of the predicted values
		-img	image we are dealing with (to obatin original labels)
	save-fig	-img_file	filename of the data to be plotted as an image
		-out_file	filename of the output image

Table A.3: *Function and options for* `prediction.py`.

Bibliography

Alpert, S. et al. (2012), 'Image segmentation by probabilistic bottom-up aggregation and cue integration', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **34**(2), 315–327.

Amancio, D. R. et al. (2014), 'A systematic comparison of supervised classifiers', *PLoS ONE* **9**(4), 1–14.

Bishop, C. (2006), *Pattern Recognition and Machine Learning*, Information Science and Statistics, Springer.
URL: *https://books.google.co.uk/books?id=kTNoQgAACAAJ*

Caruana, R. and Niculescu-Mizil, A. (2006), An Empirical Comparison of Supervised Learning Algorithms, *in* 'Proceedings of the 23rd International Conference on Machine Learning', ICML '06, ACM, New York, NY, USA, pp. 161–168.
URL: *http://doi.acm.org/10.1145/1143844.1143865*

Chinery, A. (2016), Lecture 15 - Retargetting, Matting.

Davis, J. and Goadrich, M. (2006), The Relationship Between Precision-Recall and ROC Curves, *in* 'Proceedings of the 23rd International Conference on Machine Learning', ICML '06, ACM, New York, NY, USA, pp. 233–240.
URL: *http://doi.acm.org/10.1145/1143844.1143874*

Färber, I. et al. (2010), On using class-labels in evaluation of clusterings.

Firman, M. (2016), 'List of RGBD datasets', `http://www0.cs.ucl.ac.uk/staff/M.Firman/RGBDdatasets/`.

Hall, P. (2015), 'Fundamentals of Pattern Analysis Notes', `http://www.cs.bath.ac.uk/~pmh/Teaching/Pattern_Analysis_files/notes.pdf`.

Izadi, S. et al. (2011), Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera, ACM Symposium on User Interface Software and Technology.
URL: *http://research.microsoft.com/apps/pubs/default.aspx?id=155416*

Kohavi, R. et al. (1995), A study of cross-validation and bootstrap for accuracy estimation and model selection, *in* 'Ijcai', Vol. 14, pp. 1137–1145.

Lazebnik, S. (2016).

Li, L. (2014), 'Time-of-flight camera–an introduction (texas instruments)'.

Microsoft (2013), Kinect Fusion MSDN Documentation.
URL: *https://msdn.microsoft.com/en-us/library/dn188670.aspx*

Murphy, K. P. (2012), *Machine learning: a probabilistic perspective*, MIT press.

Pedregosa, F. et al. (2011), 'Scikit-learn: Machine Learning in Python', *J. Mach. Learn. Res.* **12**, 2825–2830.
URL: *http://dl.acm.org/citation.cfm?id=1953048.2078195*

Powers, D. M. (2011), 'Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation'.

Python (2015), Python wiki: Global interpreter lock.

Ren, X., Bo, L. and Fox, D. (2012), Rgb-(d) scene labeling: Features and algorithms, *in* 'Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on', IEEE, pp. 2759–2766.

Rother, C. et al. (2004), Grabcut: Interactive foreground extraction using iterated graph cuts, *in* 'ACM transactions on graphics (TOG)', Vol. 23, ACM, pp. 309–314.

Sahoo, G. et al. (2012), 'Analysis of parametric & non parametric classifiers for classification technique using WEKA', *International Journal of Information Technology and Computer Science (IJITCS)* **4**(7), 43–49.

Sarbolandi, H. et al. (2015), 'Kinect range sensing: Structured-light versus time-of-flight kinect', *CoRR* **abs/1505.05459**.
URL: *http://arxiv.org/abs/1505.05459*

Schapire, R. E. and Freund, Y. (2012), *Boosting: Foundations and algorithms*, MIT press.

Schapire, R. E. and Singer, Y. (1999), 'Improved boosting algorithms using confidence-rated predictions', *Machine learning* **37**(3), 297–336.

scikit-learn (2016), scikit-learn Documentation.

Shao, L., Han, J., Kohli, P. and Zhang, Z. (2014), *Computer Vision and Machine Learning with RGB-D Sensors*, Springer.

Silberman, N. et al. (2012), Indoor Segmentation and Support Inference from RGBD Images, *in* 'ECCV'.

Szeliski, R. (2010), *Computer vision: algorithms and applications*, Springer Science & Business Media.

Valentin, J. et al. (2015), 'Semanticpaint: Interactive 3d labeling and learning at your fingertips', *ACM Trans. on Graphics (TOG)* .
URL: *http://research.microsoft.com/apps/pubs/default.aspx?id=244725*

van der Walt, S. et al. (2011), 'The numpy array: A structure for efficient numerical computation', *Computing in Science Engineering* **13**(2), 22–30.