Extracting Structured Information From Legal Documents

Alexey Riepenhausen

23rd August 2019

The University of Edinburgh

MSc in High Performance Computing With Data Science

Abstract

Taking unstructured text and converting it into structured information that can be used for further processing is of enormous value to organisations. In the case of the Registers of Scotland (RoS), an organisation responsible for the compilation and maintenance of public records, this approach is not even optional but essential. Over the past few years, RoS has successfully tried many approaches to achieve this goal, yet challenges remain. One of these challenges is the mapping of textual information into spatial representation in form of maps. While RoS has already implemented a number of solutions to tackle this problem, what they had not yet attempted was the use of machine learning. Thus, RoS came forward with the idea of collaborating with the University of Edinburgh by providing a master's student with a scholarship in exchange for the student's dissertation focussing on this particular problem.

Generally speaking, a model consisting of several steps was proposed. The first step involved the conversion of words into vectors by means of Mikolov's (2013) Word2vec model. Then, these vectors were fed into different version of neural classifiers based on Hochreiter and Schmidhuber's (1997) Long short-term memory architecture. Each classifier looked for a certain aspect within the text and returned a positive or negative answer concerning the presence of said aspect. For example, the positive or negative answer could signifiy the presence of a certain type of ownership within the text or the type of property the text was describing, e.g. a flat or a house. However, the most important aspect was the existence of colours within the text used to describe entities on a given map. What RoS desired was a way of automatically detecting what types of ownership each of these colours were describing. If successful, this information could be transferred onto the corresponding maps, combining textual and spatial information as a result.

To achieve this goal, manual annotation of the given text had to be done with assistance from a domain expert, turning this a supervised learning problem. Ultimately, it was shown that the proposed model was indeed suitable for solving this type of problem and could be even further improved upon by using more annotated data and a number of optimisations, all of which will be discussed in detail in this dissertation.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

This work would not have been possible without support from RoS and the University of Edinburgh. In particular, it should be noted that this project is funded by the Registers of Scotland 400 year anniversary scholarship, meaning that the tution fees for this MSc are taken over by RoS. For this reason, I would like to express my sincere gratitude towards the Registers of Scotland.

Further, I would also like to express my gratitude to my supervisor Mr Ally Hume for his clear guidance and honest feedback throughout the master's project. I would also like to thank RoS staff, most notably Mr Alan Howie and Dr Anthony Beck, for their organisational and technical support on getting the resources needed to successfully complete this project.

1 Introduction

As organisations are increasingly aiming to digitise their services, the significance of machine learning as a means to aid or supplement this development is also growing. While the latter is often plagued by the black box problem, a phenomenon where the underlying process by which decisions are made is inherently opaque, using it in conjunction with more traditional approaches can still be very useful.

In recent years, the Registers of Scotland (RoS) has been finalising its digital transformation process with one of the objectives being the migration of services into the digital realm. In a nutshell, the Registers of Scotland's main responsibility is the maintenance and compilation of legal documents concerning land and property across the entirety of Scotland. Given the large amount of textual information at their disposal dating back to the 17th century, the benefits of migrating from a paper-based to a fully digital system are numerous.

One of the digital services provided by RoS as of 2019 is the online land information system ScotLIS and the Digital Discharge Service (DDS). The former provides users with information concerning land and property, whereas the latter is primarily used by solicitors for the purpose of discharging mortgage securities. However, the final goal is the integration of different types of property data into an all-encompassing Land Registry System (LRS), which requires linking both textual and map data in a consistent manner. Most of the time, the absence of direct references between these two types of data poses a considerable obstacle to said integration. As a result, RoS has been looking into using machine learning as additional means of achieving this goal along with the current systems that they have in place.

In order to address this challenge from multiple angles, RoS decided to form a collaboration with the University of Edinburgh by providing a scholarship to a selected master's student. In exchange for the scholarship. the student's dissertation had to make a contribution towards one of the projects at RoS. Ultimately, the chosen project was concerned with the automatic extraction of ownership patterns from legal documents describing various types of properties. The documents themselves contained descriptions of different types of ownership and verbally assigned different colours to these ownerships. Further, each document came with a map where different areas were coloured according to the types of ownership they were corresponding to. To put it simply, the objective was to turn the textual, implicit colour links within the documents into explicit colour-to-ownership mappings by means of Natural Language Processing (NLP) to address the lack of direct links between textual and spatial data.

2 Problem Definition

Taking an implicit cross-referencing system based on text and turning it into an explicit link requires a way of isolating the implicit colour references from the text and classifying them according to the types of ownership they represent. To paint a clearer picture of this problem, the structure of the legal documents has to be examined in great detail before a potential approach for tackling this problem can be proposed.

2.1 Title Sheets

The title sheet is a legal document representing the right of ownership, which in legal terms is referred to as the *subject*, as well as the person or entity who owns it. Additionally, the title sheet states the benefits and encumbrances associated with the *subject*. Broadly speaking, benefits grant the permission to use the land in question in some beneficial way, whereas encumbrances represent a restriction or limitation, for example in form of a liens (Beck, 2019).

Overall, a title sheet encompasses several sections detailing the legal circumstances described above. In the context of this dissertation, the only section of interest for extracting the relationships between colour and ownership are detailed in the A-section (ibid.).

2.1.1 General Structure of A-Sections

The A-section details the ownership right described in land and any other benefits by dint of owning land. A generic A-section follows a textual structure (ibid.):

- **The subject** itself, i.e. what land is owned and where that land is located geographically which is described by means of address. Overall, the subject constitutes the *cadastral unit*
- **The type of ownership** concerning the property or part of the property. If the ownership includes the land area of said property, i.e. the *solum*, it has to be clear what separate tenements (rights of ownership) have to be explicitly excluded from this ownership type. For instance, these separate tenements can be elements within the land owned by a third party. Conversely, certain tenements that are normally excluded have to be explicitly included. An example are the separate tenements held by the British Crown, e.g. coal, gold, petroleum, salmon fishing and mussels.
- **The beneficial ownership rights** extending beyond the said *cadastral unit*. These are also described as *pertinents* and have to state whether the ownership is exclusive or common. In the latter case, the proportion of the ownership has to be given as well.

- **The cartographic description of said property**, which ideally allows the identification of the property's spatial extent. If no such description is provided, the spatial extent may be verbalised or cross referenced.

2.1.2 Flatted Properties

Owning a flat essentially means that exclusive access and ownership to a volume of space above the solum, also called *strata*, is granted along with the common right of access to other elements of the property, e.g. stairs. Normally, this also means that by owning said *strata* a partial ownership is granted concerning the solum upon which the tenement is built, which is referred to as common solum in short (Beck, 2019). The concept of solum and strata given the specific context of flatted properties is illustrated in figure 1.

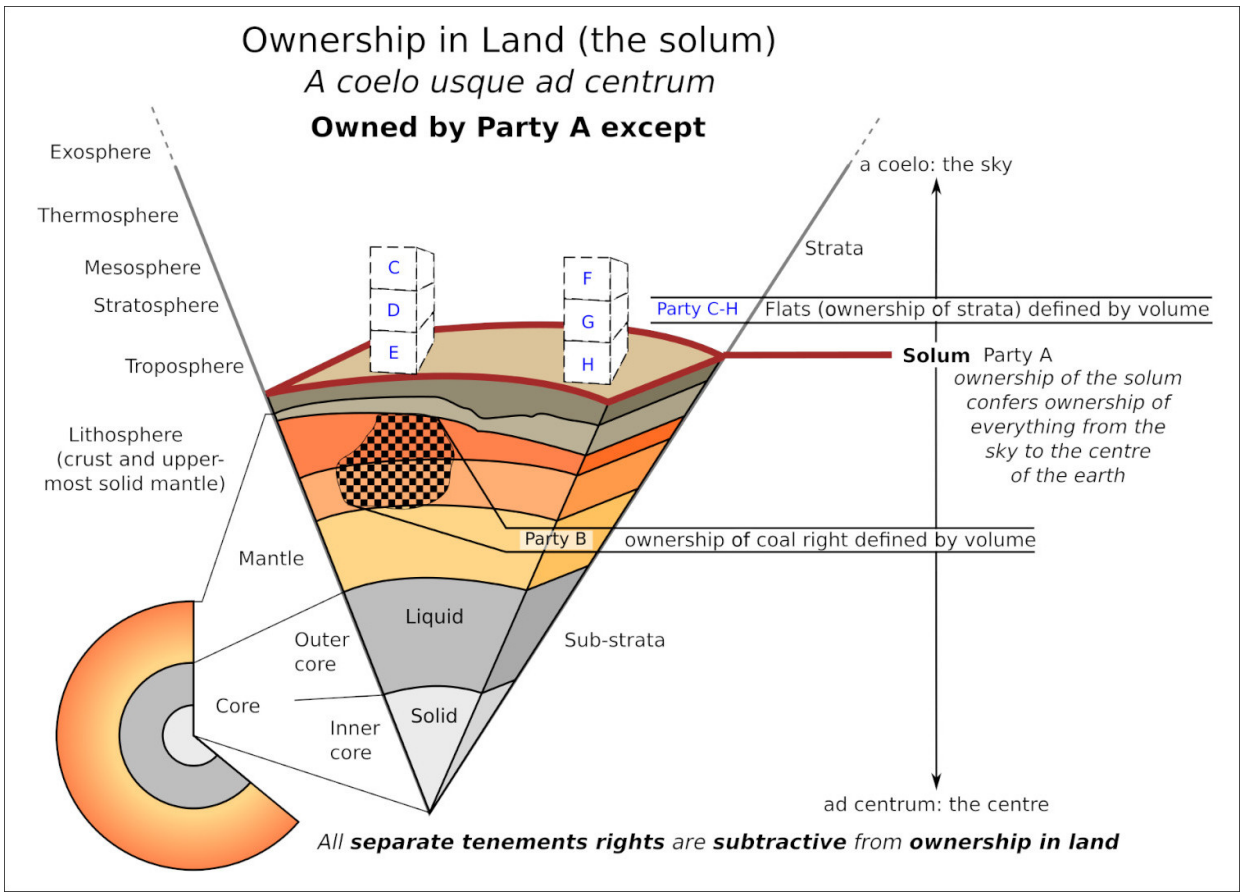

Figure 1: Conceptual representation of the solum and strata (Beck, 2019)

As outlined previously, a flat is represented as a volume of space above the solum, whereas the solum itself can be viewed as the surface upon which the property in question is built. When taking *separate tenements* into consideration such as coal, it should be clear that

ownership is not only confined to a two-dimensional plane; in fact, it extends from the centre of the earth all the way up until the exosphere.

However, since the goal is to isolate colours denoting different types of ownership and map them onto a two-dimensional spatial representation of said property, the aforementioned tenements such as coal are not really relevant within the context of this task. Rather, the types of ownership concerning flats that are actually relevant for the classification problem are:

- exclusive strata
- exclusive solum
- common strata
- common solum

The concept of solum and strata have already been discussed above; what common and exclusive refer to in this context is whether the solum or strata is owned by one or several entities. Hence, exclusive solum and strata are denoting those items on the map that are exclusively owned by the entity or entities described as owners in the title sheet. Conversely, common solum and common strata refer to those items on the map that are shared with entities not explicitly denoted as owners within the title sheet.

2.2 A-Section Example

With the generic structure of A-sections in mind, taking a look at a specific sample to visualise the process of mapping colours to ownership is an important step towards understanding the task at hand. Here, figure 2 shows the A-section of a flat from Glasgow with manually highlighted colours, while figure 3 depicts the spatial representation of said flat along with a manual mapping of the colours to their respective types of ownership.

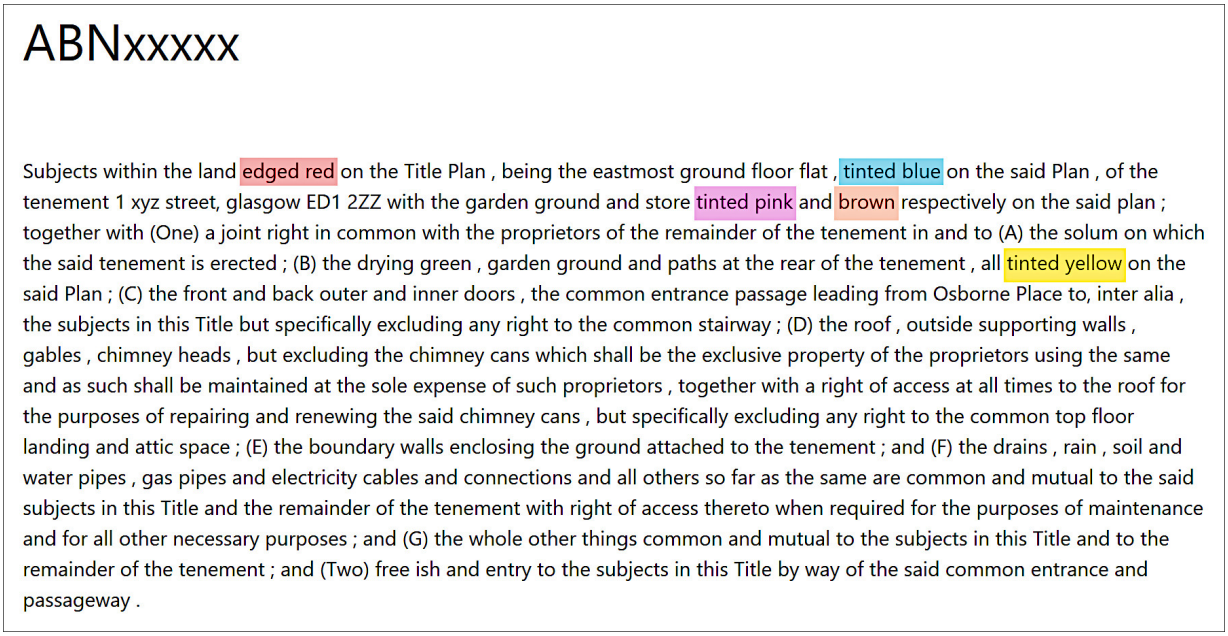

Figure 2: Example A-section with manually highlighted colours

Most importantly, every colour is accompanied by a particular style or configuration, in this case *edged* and *tinted*. Other configurations and colours such as "hatched" and "green" also exist, but they do not appear in this particular example. To avoid confusion, pairs of configurations and colours will be referred to as colour-configuration pairs, colour-style pairs or for simplicity's sake colour-pairs throughout the dissertation.

The first colour-pair in this document is *edged red* and refers to the tenement steading of the property in question. In essence, the tenement steading denotes the boundaries of the property and does not represent ownership itself. Given that every flatted property has a tenement steading, however, it is still necessary to annotate this particular colour-pair to ensure that the classifier will make a distinction between colour-pairs representing ownership and colour-pairs representing different concepts. In this case the latter happens to be tenement steading; other colour-pairs can also represent right of access, even though in this example the right of access is described verbally, e.g. by stating " ... together with a right of access at all times to the roof for the purposes and renewing the said chimney cans ... " (fig. 2).

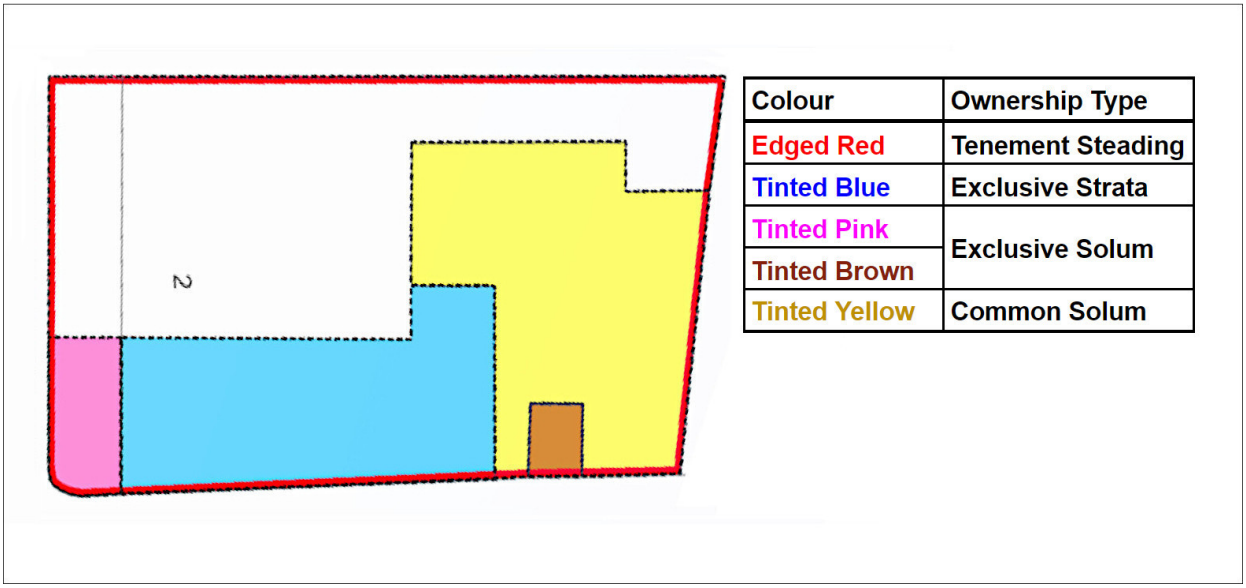

Figure 3: Example A-section with manually highlighted colours

Apart from *edged red*, there are four more colour-pairs within this piece of text. Two of these colour-pairs represent exclusive solum, while the remaining two represent exclusive strata and common solum (fig. 3). On the map, the colour-pair "tinted blue" points to the location of the actual flat on the ground floor, which can be easily inferred from the first line of text: " ... being the eastmost ground floor flat, *tinted blue* on the said Plan ... " (fig. 2). Here, it should be noted that even though the flat itself is located on the ground floor, it is still classified as exclusive strata. The flat's strata is exclusively owned by the proprietor of the flat, while the solum upon which the flat is built is implicitly shared between the owners of the flats within the tenement. In this document, however, both the solum and the strata happen to overlap in two-dimensional space, even though they are still two different concepts.

Next, *tinted pink* and *tinted brown* denote items on the map that are both located on the solum and that are exclusively owned by the proprietor of the ground floor flat. The reason why these colour-pairs are exclusive is due to them being mentioned before the A-section explicitly talks about common ownership, which it describes as " ... a joint right in common with the proprietors of the remainder of the tenement ... " (fig. 2). Hence, *tinted pink* and *tinted brown* represent exclusive solum.

Lastly, *tinted yellow* points to the " ... drying green, garden ground and paths at the rear of the tenement ... " (fig. 2). Since these items are part of the solum and occur in the text after a joint right in common had been mentioned, their respective colour-pair is therefore labelled as common solum.

2.3 Colour-Pair Classification Approach

The main idea behind the classifier is based on the conversion of individual words into their respective vector representations and the subsequent supervised training of a Long Short-Term Memory (LSTM) classifier using said vectors.

In short, word vectors are a means of representing words in a distributed fashion whereby related words are put into semantically similar groups (Mikolov, Sutskever, et al., 2013). This is a necessary pre-processing step to allow a given algorithm to pick up the context surrounding the colour-pairs. Once the conversion into vectors is achieved, a model capable of deducing the right combination of colour-pair and ownership needs to be chosen. In the past, word-vector-based neural models were successfully used in the realm of sentiment analysis as stated in the respective chapter in the book on Deep Learning in Natural Language Processing (Tang and M. Zhang, 2018). Basically, there are two approaches of potential interest to RoS's use case layed out in the said book. The first approach obtains sentence representations from the given word vectors and feeds them into a convolutional neural network. While this is very useful for identifying sentiment polarity at the sentence level, the downside of this approach are its limitations when it comes to learning long-term dependencies between different words (ibid.). Having looked at the overall structure of A-sections, it is relatively clear that descriptions of ownership patterns and colour-pairs do in fact form long-term dependencies across the text. This is exactly what the second approach using an LSTM, a variation of a recurrent neural network (RNN), is trying to identify during learning (ibid.). The main reason for using LSTM's as opposed to standard RNN's are the latter's overly simple, additive operations to its given word embeddings, leading to several problems ranging from exploding or vanishing gradients to the lack of flexibility (Wang et al., 2015). LSTM cells are not affected by these issues nearly as much due to their gated structure; they have also shown great performance in twitter sentiment prediction (ibid.). Since the previously discussed A-sections are much more structured compared to anything that can be found on Twitter, it is expected that this method is going to yield results of similar or potentially even higher quality.

To make supervised learning possible, manual annotation of a certain number of documents is required. From a practical point of view, an A-section has to be read, understood and its colour-pairs have to be correctly labelled with the corresponding type of ownership. When it comes to converting the words within the documents into vectors, the colour-pairs themselves need to be replaced with specific tokens to make sure that the classifier can correctly identify the existence of one or more colour-pairs. Wang (ibid.) also briefly mentions a similar issue in his paper on LSTM cells, where the positive or negative sentiment of specific words cannot be distinguished by means of distributed word representation, i.e. word vectors. While the latter does not matter too much in the use case of sentiment analysis due to the LSTM classifier eventually learning to pick up sentiment from phrases rather than single words (ibid.), there is a concern that the colour-pairs will be assigned a vector representation obscuring their intended meaning. Fortunately, the colour-pairs are limited in numbers and are easy to identify within the text, thus making a one-by-one replacement of with bespoke vectors

feasible.

On a final note, the manual annotation of A-sections requires additional induction from a domain expert. When it comes to managing time, this is something to take into consideration. The original plan intended the allocation of two full weeks to the annotation process.

3 Background

Having outlined the intended classification approach in the previous section, the next topic to be discussed is the theoretical background forming the basis of the envisioned classifier. More specifically, this section will present the vectorisation of words based on the Word2Vec model and the workings of the LSTM classifier.

3.1 Word2vec

The Word2vec model encompasses a set of methods for generating vector representations of words, most notably by means of continuous bag-of-words (CBOW) and skip-grams (SG). Furthermore, there exists a range of optimisations such as negative sampling and hierarchical softmax for improving performance and accuracy of the word representations (Rong, 2014).

In general, vector representation of words in an n-dimensional space results in the clustering of words with similar meanings and semantic interrelationships. This can be easily observed by applying simple vector arithmetic to an example given in Mikolov's paper (2013): When subtracting the vector for Spain from Madrid and subsequently adding the vector for France, as a result, one would get a vector that is very close to the vector for Paris in terms of spatial distance.

In the context of document classification, converting a set of words into their respective vector representations is a necessary pre-processing step before feeding them into the given classification model. This is a fairly obvious approach, since almost any type of practical document classifier such as Support Vector Machines (SVM) or neural network models (Tang, Qin, et al., 2015) requires a numerical representation of features to enable learning.

3.1.1 The Skip-gram model

In simple terms, the skip-gram model can been seen as a simple, one-layer neural network learning to predict the probability of observing different words next to a given input word. Ultimately, the purpose of this exercise is the extraction of word vectors from the hidden layer at the end of the training session (Rong, 2014).

The number of weights within the hidden layer is determined by two dimensions, namely, the total number of unique words or word embeddings times the desired size of the final vectors (McCormick, 2016). Assuming the number of unique words to be 10,000 and the desired vector size to be 300, the weight matrix would look as follows (fig. 4):

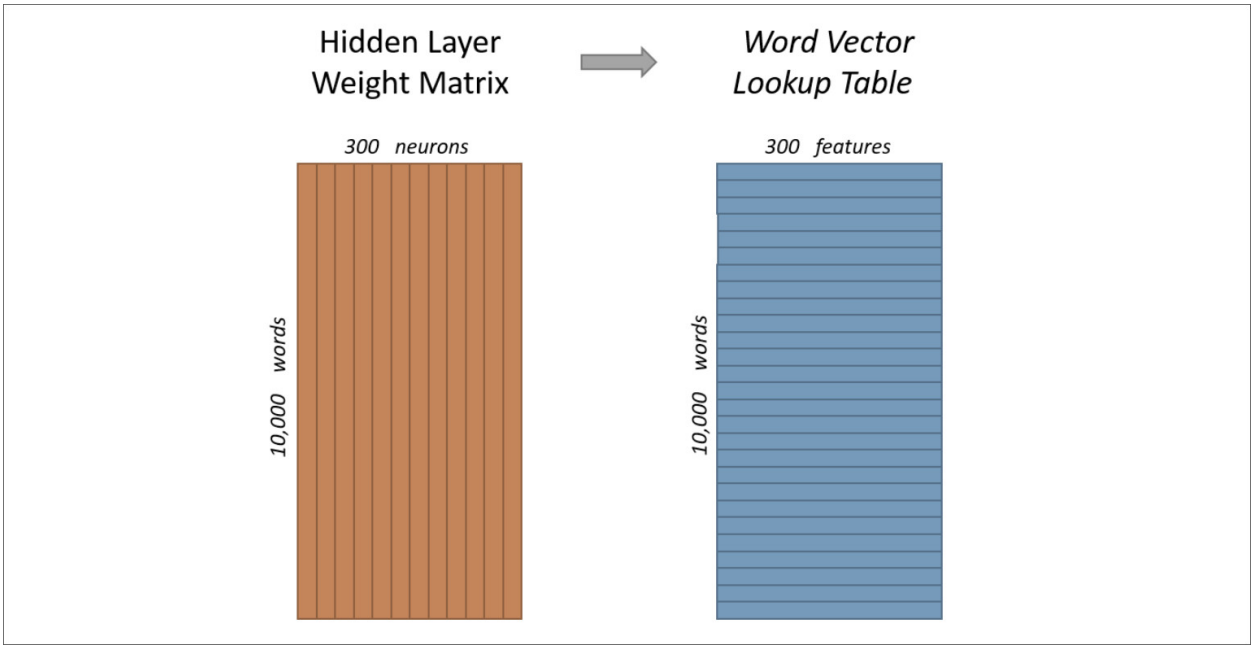

Figure 4: High level overview of the skip-gram model (McCormick, 2016)

Once again, the hidden layer is trained to predict the neighbouring words for every unique word encountered within the entire text corpus. In this particular example (McCormick, 2016), the weight matrix of the hidden layer is used as a lookup table at end of the training session to extract the final vectors. Every single row of the blue matrix representation in figure 4 stores the information concerning the final vector embeddings for every unique word in the text corpus. As a result, the dictionary of unique vectors is essentially stored within said weight matrix once the training session has finished.

A more mathematical perspective is given in figure 5, which depicts the general structure of this variation of the word2vec model as described in Rong's paper (2014). In this figure, V represents the size of the vocabulary, i.e. the number of unique words within the text corpus, which is effectively the size of the input and output layer; N represents the size of the hidden layer and W is the matrix of weights between input and output layer, with its dimensions being $V \times N$.

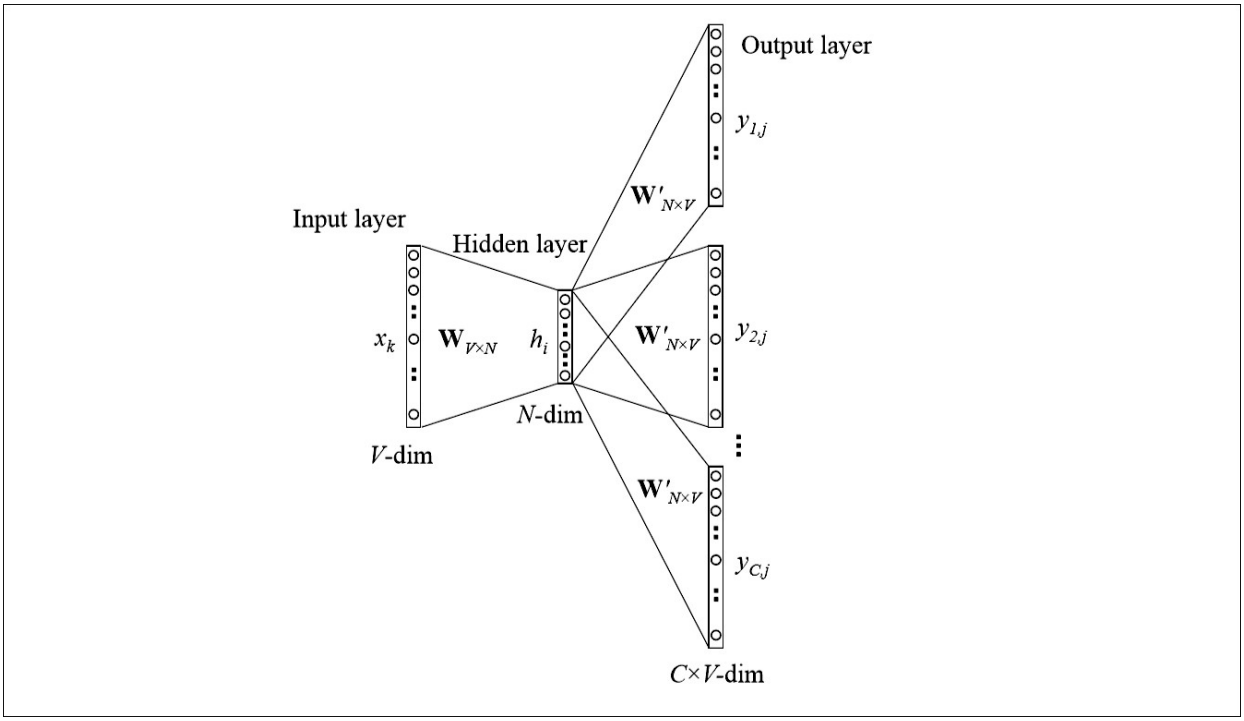

Figure 5: High level overview of the skip-gram model (Rong, 2014)

Before training of the skip-gram model via gradient descent can commence, the words in the vocabulary have to be converted into a one-hot vector of size V. Essentially, this means that each word is represented by a vector in which all of its elements are set to zero apart from the one element located at the vector's index representing the word's position in the vocabulary, which will be set to one (Rong, 2014).

Given the time constraints of this project, Skip-grams were chosen as the sole model for the implementation. From a practical perspective, both Skip-grams and CBOW arrive at similar vector representations, albeit Skip-grams show a tendency to perform slightly better with smaller data sets (Mikolov, V. Le, and Sutskever, 2013). In other words, the difference between the two was not proven to be significant enough to give preference to one or the other; yet, a choice had to be made. Future attempts at optimising the classifier may consider using CBOW instead of Skip-grams if deemed necessary.

3.1.2 Objective Function

Looking at it from a more mathematical perspective, the goal of training the skip-gram model is to maximise the average log probability given a sequence $w_1, w_2, ..., w_n$ of training words (Mikolov, Sutskever, et al., 2013):

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{-d\leq i\leq d, i\neq 0}\log p(w_{n+i}|w_n) \tag{1}$$

Here, d represents the training window, a variable usually defining the number of words to be considered during training to the centre word's left and right. More complex implementations such as setting d to be a function of the centre word are also possible.

The probability $p(w_{n+i}|w_n)$ is defined by the softmax function, with V being the unique number of words within the vocabulary (Mikolov, Sutskever, et al., 2013):

$$p(w_I|w_J) = \frac{\exp({v'_{w_I}}^{\top} v_{w_J})}{\sum_{w=1}^{V}\exp({v'_w}^{\top} v_{w_J})} \tag{2}$$

While both vectors v_w and v'_w represent the same word w, their origins are in fact different. Vector v_w originates from the rows of matrix W and vector v'_w is drawn from the columns of W. Both Rong (2014) and Mikolov (2013) refer to v_w as the input vector because the rows of W represent the mapping from the original input vector to the hidden layer. Conversely, v'_w is referred to as the output vector due to W's rows mapping from the hidden to the output layer.

Despite its simplicity, using this particular implementation is quite problematic for large vocabularies, since the time required to calculate the gradients for every $\log p(w_{n+i}|w_n)$ grows in proportion to the vocabulary size V (ibid.). Fortunately, there are more effective ways of accomplishing this task such as negative sampling and hierarchical softmax (ibid.).

3.1.3 Negative Sampling

Rather than updating all probabilities via gradient descent, negative sampling only updates the positive output words, i.e. the words that are supposed to appear around the centre word, and a sample of negative words not associated with the centre word. For this reason, this method is called negative sampling (Rong, 2014). With that being said, the training objective is redefined as (Mikolov, Sutskever, et al., 2013):

$$\log\sigma({v'_{w_I}}^{\top} v_{w_J}) + \sum_{i=1}^{k}\mathbb{E}_{w_i\sim P_n(w)}\left[\log\sigma(-{v'_{w_i}}^{\top} v_{w_I})\right] \tag{3}$$

replacing $\log p(w_{n+i}|w_n)$ in equation 1 as the Skip-gram target. Here, k represents the number of negative samples and $P_n(w)$ represents the noise distribution from which these samples are drawn. Mikolov (ibid.) mentions that empirical evidence suggests k to be set between 5-20 for smaller data sets and 2-5 for larger data sets.

3.1.4 Hierarchical Softmax

Another approach to reduce the number of updates is hierarchical softmax. Unlike negative sampling, this method constructs a binary tree from the output layer, resulting in a random walk assigning probabilities to words (Mikolov, Sutskever, et al., 2013):

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma\Big(||n(w,j+1) = ch(n(w,j)|| \cdot v'^{\top}_{n(w,j)} v_{w_I}\Big) \tag{4}$$

Here, $n(w,j)$ is the *j-th* node in the representation of the binary tree, with $L(w)$ being the length of the path from the root to said node. The child nodes of any given node n are denoted by $ch(n)$. In the end, the equation $||x||$ is set to 1 if x is true and -1 if x is false. Overall, the amount of update operations gets reduced to $Log(N)$.

As with CBOW and Skip-grams, one method had to be chosen over the other when it came to implementating the word2vec model. In this case, negative sampling was chosen because it was easier to implement than the tree structure required by hierarchical softmax.

3.1.5 Word Subsampling

One common issue that comes with a very large corpus is the dominance of frequent words such as *a* or *the*. This is a considerable obstacle to training the vector representations efficiently, since combinations of *a* or *the* with more meaningful words do not provide much useful information for the given model. In addition, vector representations ceases to noticeably change after a certain amount of training samples (ibid.).

To counteract this problem, Mikolov (ibid.) introduces the idea of subsampling to rectify the skewed balance between more frequent and less frequent words:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{5}$$

The variable t denotes a pre-selected custom threshold and $f(w_i)$ denotes the frequency of word w_i. The subsampling threshold chosen for this particular application was set to be 10^{-5}, which was shown to work well in practice (ibid.).

3.1.6 Cosine Similarity

Cosine similarity is a widely used method for measuring the similarity between two vectors and is calculated as the normalised dot-product (Sidorov et al., 2014):

$$a \cdot b = \sum_{i=1}^{N} a_i b_i \tag{6}$$

$$||x|| = \sqrt{x \cdot x} \tag{7}$$

$$cosine(a, b) = \frac{a \cdot b}{||a|| \times ||b||} = \frac{\sum_{i=1}^{N} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{N} a_i^2}\sqrt{\sum_{i=1}^{N} b_i^2}} \tag{8}$$

Equation 10 represents the dot-product between two vectors, while equation 11 denotes normalisation. Thus, the cosine similarity is defined as the dot-product of two vectors over the product of their normalisations (fig. 12). The range of possible values for cosine similarity is set between -1 and 1, where a value of 1 means that the vectors are identical and -1 meaning that the vectors are completely opposite in regards to their location in vector space.

Measuring similarity between vectors is not the main goal of the project, yet it is very useful when trying to get a better understanding of the workings of the LSTM classifier when this particular property is known. For instance, the performance in distinguishing colour-pairs properly may very well be dependent on the similarity or dissimilarity of their vectors.

3.2 LSTM Cells

LSTM stands for long short-term memory and was developed by Hochreiter and Schmidhuber (1997) to address the problem of exploding and vanishing gradients observed with recurrent neural networks (RNN). In the past, LSTM's have been used in the realm of sentiment analysis with great levels of success, in particular when sentiment is target-dependent (Tang, Qin, et al., 2015). Target-dependent sentiment analysis resembles the problem this dissertation is trying to solve in many ways. To illustrate this point, the following sentence was drawn from Tang's paper (ibid., p.1): *I bought a new camera. The picture quality is amazing but the battery life is too short.* In this example, there are two possible targets of sentiment. The first target is picture quality, which would be labelled as clearly positive. On the other hand, if the main concern is battery life, the label is obviously negative. While sentiment itself is not of great interest to RoS, assigning ownership labels to different colour-pairs in the same document does in fact bear a striking similarity to assigning sentiment labels to specific entities within a product review.

3.2.1 RNN vs LSTM

Recurrent neural networks have a hidden layer passing its own outputs back to itself when processing the next element in a sequence, i.e. a word vector. The network retains a form of memory of previous states, thus enabling it to pick up the context within a sequence (Graves et al., 2008). This mechanism and its limitations are illustrated in figure 6, where the workings of a simple RNN are shown over a period of time:

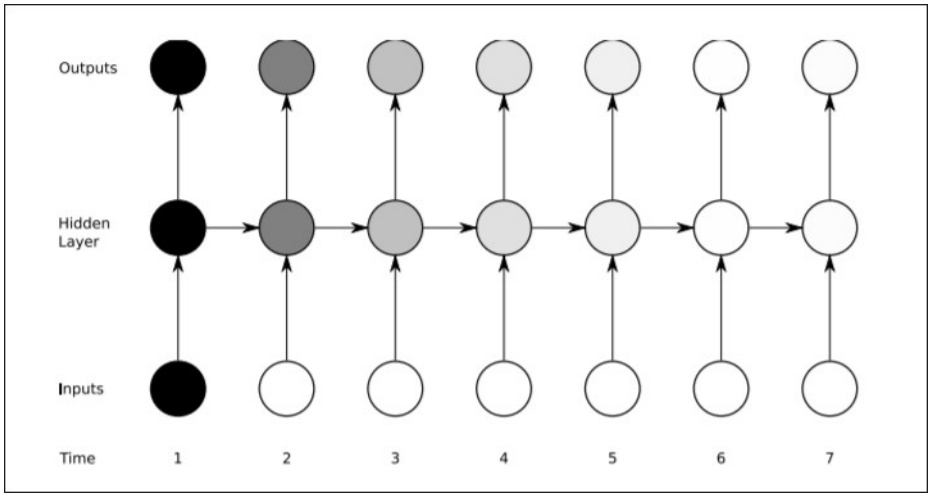

Figure 6: Vanishing gradient problem illustration (Graves et al., 2008)

In this diagram, the nodes are coloured according to their sensitivity towards the very first input. Clearly, the influence or memory of previous inputs fades away very rapidly, making RNN's less suitable for the long stretches of text within RoS's A-sections. In fact, Graves (ibid.) suggests that it is inherently difficult for an RNN to remember relevant context when the distance becomes greater than ten time steps, far too short for the sizes of a normal A-section.

LSTM cells, albeit more complex in structure compared to standard RNN's, do not suffer this type of problem. Here, the hidden layer is made of several smaller, recurrently connected networks. These networks are referred to as memory blocks containing several cells, which in turn are activated by the input, output and forget gate (fig. 7). Having this particular setup enables the individual LSTM cells to retrieve and remember information over potentially very long sequences (ibid.).

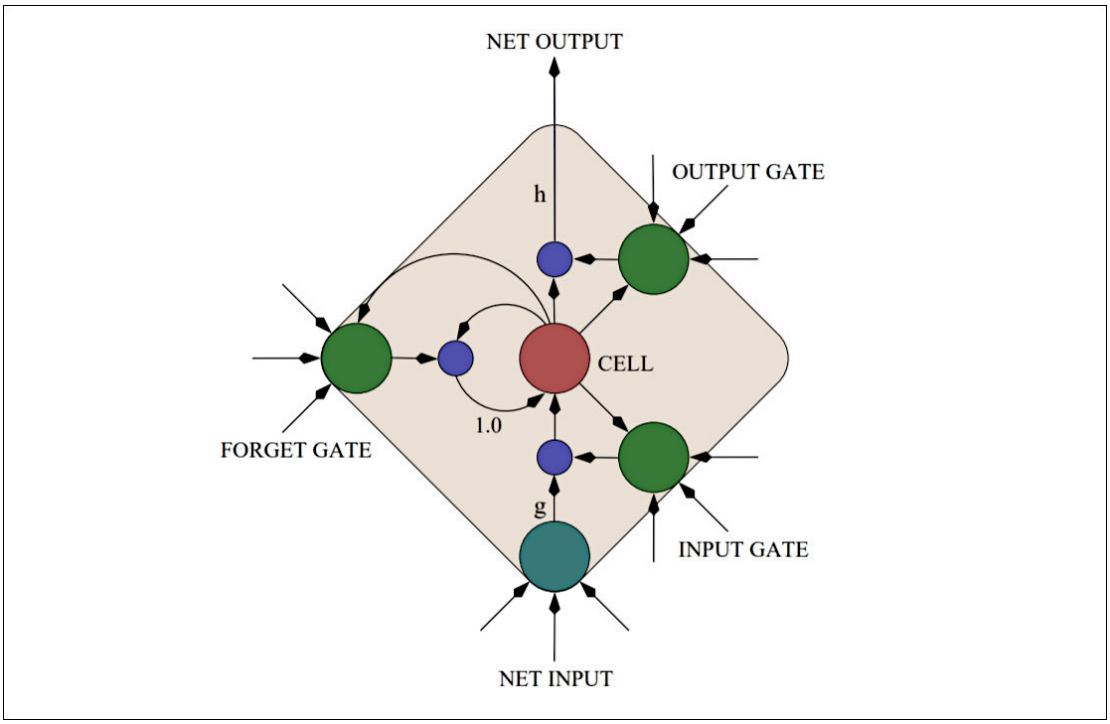

Figure 7: High level overview of an LSTM cell (Graves et al., 2008)

In this example (fig. 7), the LSTM cell has its weight set to 1.0 concerning its connection with the forget gate. The blue dots connecting it to the three gates represent units of multiplication, while the gates themselves gather their input from various regions of the neural network. In essence, the three gates function as modulation units concerning the cell's recurrent connection, its input and output (Graves et al., 2008).

3.2.2 LSTM Formal Definition

A formal definition of the standard LSTM cell is given by Johnson (2016) in his paper on supervised and semi-supervised text categorization:

$$
\begin{aligned}
i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}) \\
o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}) \\
f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}) \\
u_t &= \tanh\left(W^{(u)}x_t + U^{(u)}h_{t-1} + b^{(u)}\right) \\
c_t &= i_t \oplus u_t + f_t \oplus c_{t-1} \\
h_t &= o_t \oplus \tanh\left(c_t\right)
\end{aligned}
\tag{9}
$$

In this case, σ represents the sigmoid function, which limits the output values of the respective gate between 0 and 1, while the $\oplus$ symbol represents element-wise multiplication. The variable $x_t \in \mathbb{R}^d$ represents the input vector t with its size being denoted by d. Likewise, the dimension h represents the size of the hidden layers. Each gated unit has its own output, namely $i_t \in \mathbb{R}^h$, $o_t \in \mathbb{R}^h$ and $f_t \in \mathbb{R}^h$, as well as its own set of weights in form of the two matrices $W \in \mathbb{R}^{h \times d}$ and $U \in \mathbb{R}^{h \times d}$. Ultimately, the state of cell c_t is determined by the forget gate and its previous state, the input gate as well as the activation function denoted by u_t in equation 9.

3.2.3 LSTM Loss Function

The loss function used for backward propagation at the end of the classifier's feedforward process was cross entropy loss (Grosse, 2017). Its definition is given in equation 10 and 11, whereas a more general definition using several classes is given in equation 12 (ibid.):

$$CrossEntropy(y,t) = \begin{Bmatrix} -\log(y) & if \quad t=1 \\ -\log(1-y) & if \quad t=0 \end{Bmatrix} \tag{10}$$

$$CrossEntropy(y,t) = -t \log y - (1-t) \log 1 - y \tag{11}$$

$$Loss_{CE} = -\sum_{k=1}^{K} t_k \log y_k \tag{12}$$

4 Implementation Design

This section is concerned with the actual implementation of both the Word2vec model and the LSTM classifier. Details concerning the preprocessing of text as well as the use of external frameworks and parts of the code are going to be discussed as well. It should also be noted that the basic implementations of the Word2vec and LSTM model were borrowed from the two Github users *emadRad* (2018) and *Andras7* (2018). The specifics of the borrowed code and the two authors of the Word2vec and LSTM implementation are described in more detail in the appendix (7.2.2).

4.1 Text Preprocessing

The process of converting words into vectors requires a set of preprocessing steps. Firstly, the words themselves have to be converted to lower case to avoid duplication of upper and lower case words. This also reduces the amount of vectors required. Secondly, special characters such as commas and potentially other types of characters, e.g. brackets and quotes, have to be removed. In some configurations those characters can actually be kept, for instance when performing semantic analysis or when sentences need to be isolated. However, in the basic implementation of the Word2vec model this is arguably unnecessary. Lastly, all words within all training samples have to be aggregated within a dictionary of unique words (ibid.).

4.1.1 Word Dictionary

There are two main reasons for creating a dictionary of unique words: Recording every word's frequency within the entire corpus and saving memory space by assigning every word with a unique integer identifier. Word frequencies are crucial when it comes to sampling infrequent words to avoid training imbalance. In addition, the dictionary size also determines the embedding size of the one-hot vectors used as the input layer for the word2vec model (ibid.).

4.1.2 Removing Addresses

One problem with descriptions of properties is that they contain one or more different addresses, which in turn consist of postcodes, numbers and street names. If approaching this naively, the Word2Vec model would treat all of these entities as separate vectors. While this may not be the worst possible problem, it introduces a lot of clutter and unnecessary training cycles. At the same time, this problem can be mitigated very easily without too much of an additional implementation effort by simply replacing the different components of an address with one single vector. This measure was possible because the addresses of the A-sections

were provided separately from the actual text. The results of this measure can be seen in table 1:

	Houses		**Flats**	
Address removed?	Embeddings	Words	Embeddings	Words
No	7,702	394,586	9,373	1,179,267
Yes	6,259	380,813	8,222	1,162,625

Table 1: Vector embedding sizes and number of words for houses and flats

RoS provided two separate data sets, one exclusively for houses and the other one exclusively for flats. In this particular table, the columns for *Words* refer to the total amount of words encountered in 5,000 A-sections for either exclusively flatted properties or houses. Replacing the address components with one bespoke vector resulted in a considerable reduction of embedding size by 18.74 % for houses and 12.28 % for flats. At the same time, the total amount of words was only reduced by 3.49 % for houses and 1.41 % for flats. As suspected, the words removed were mostly postcodes, street names and numbers, all of which tend to have a very low frequency throughout the entire set of words. This is not very surprising, since addresses are intended to be unique identifiers of property.

4.1.3 Unknown Words

Testing the classification model requires the use of test samples that the classifier has not been trained on. Inevitably, there will be a certain percentage of words for which there exists no vector, unless the Word2vec training was performed on both the training and test set. The latter approach is problematic in the sense that in a realistic deployment scenario, the model will eventually encounter previously unknown pieces of text. Thus, this particular Word2vec implementation simply skips those words if the vector is unknown. A different approach is to use a specific vector as a substitute, e.g. a dictionary-sized vector consisting of zeros. More sophisticated approaches may exist, but were not considered due to time restrictions, especially since the above methods are very straightforward.

4.1.4 Dummy Vectors for Colour-pairs

One of the issues with vector representation is that the colour-pairs themselves will get a vector based on their surrounding words. The latter is not necessarily wrong, but given the importance of the colour-pairs, a more recognisable set of vectors is required.

To address this issue, it was decided that the colour-pairs will be assigned a unique one-hot vector as their main representation. The main idea behind the one-hot vectors is that every one hot vector is going to be clearly distinct from every other possible vector in terms of cosine similarity (eq, 12).

One consequence of introducing the one-hot vectors is that the trained non-colour-pair vectors themselves had to be padded with zeros at their end (fig. 8).

0	1	2	...	50	51	52	53	54	Words
1.432	0.286	-0.74	...	0.343	0	0	0	0	Land
0	0	0	...	0	**1**	0	0	0	Edged
0	0	0	...	0	0	**1**	0	0	Red
0	0	0	...	0	0	0	**1**	0	Tinted
0	0	0	...	0	0	0	0	**1**	Blue

Figure 8: Vector representations with 50 dimensions and the additional padding

With this particular setup, it is guaranteed that the vectors representing colour-pairs will have a similarity of zero with every other possible vector. To ensure that this actually worked in the implementation, a similarity matrix for every unique word in the vocabulary was produced after introducing the one-hot vectors (fig. 9):

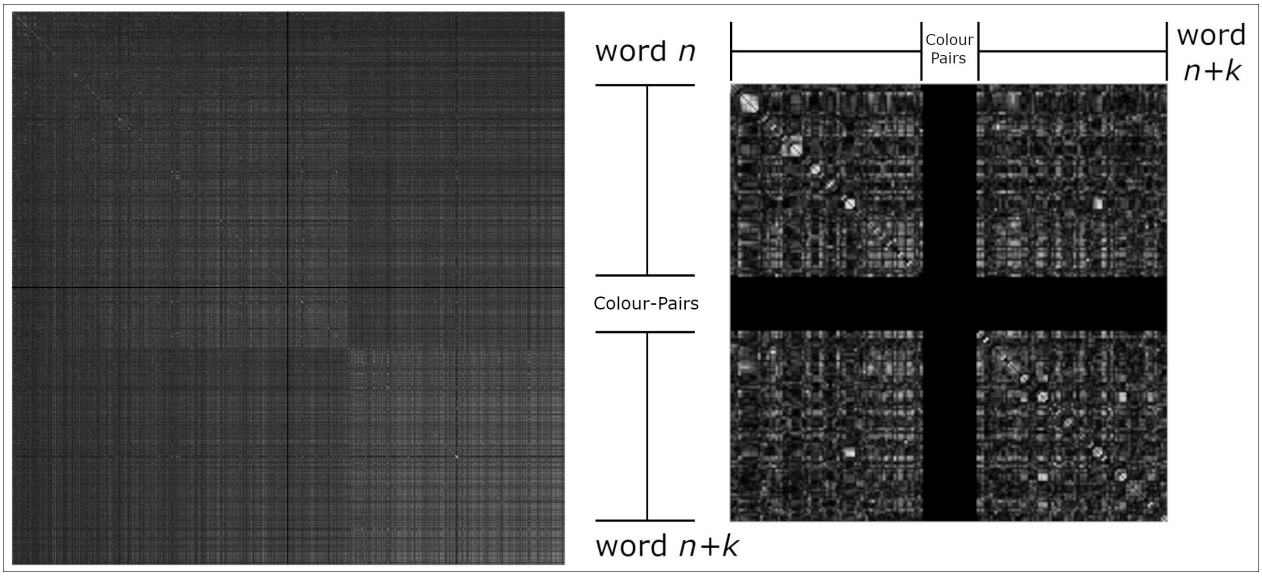

Figure 9: Cosine similarity matrix in full scale (left) and zoomed in (right)

In this similarity matrix, every pixel represents a similarity value between 0 and 1, with 0 being coloured completely black and 1 being coloured completely white. The pixel rows represent every unique word in the order it was encountered when parsing the A-sections. Likewise, the columns feature the same representation of words, just in a transposed manner. Furthermore, the pixel values representing the colour-pairs were deliberately placed in the

middle to highlight their similarity compared to other vectors. Since the equation for cosine similarity (eq. 12) produces values between -1 and 1, the implementation itself converted all negative values into positive values. The main purpose of this modification was to highlight the difference between those vectors that have nothing in common versus the vectors that are related or at least semantically opposite to each other.

As can be seen in figure 9, the individual colour-pairs form a thin, black cross in the middle of the similarity matrix, which is precisely what was intended. Not only are the colour-pairs distinct from all other words, they are also completely distinct from each other.

4.2 Classification Models

In this project, two classification models were considered for assigning ownership to colour-pairs. The first model envisioned the use of a colour-pair matrix highlighting different colour-pairs it considered to be a specific type of ownership. With this setup, one would need four different classifiers in total, each of them trying to detect its own colour-pairs. It became clear during the implementation, however, that this approach only works when the number of unique colour-pairs is limited. As the project progressed and more A-sections were annotated, the first model became less and less feasible due to the increasing number of colour-pairs. Thus, a second model consisting of a series of binary classifiers was proposed to address the shortcomings of the first model. Both models will be presented in this section to show the progression from the initial starting point to the final implementation.

4.2.1 Colour-Pair Matrix Classifier

It was originally assumed that there are only three possible configurations, namely edged, tinted and hatched as well as eight different colours. With that, it initially made sense to organise the classifier's output in the form of a three times eight matrix (table 2). When running the classifier on any given document, the output matrix would display a probability distribution for each of the possible colour-configuration pairs. Ideally, the colour-pair denoting the right type of ownership would come out of the classifier with the highest probability among all of the possible colour-pairs.

Config | Colour	**Red**	**Green**	**Blue**	**Pink**	**Brown**	**Yellow**	**Grey**	**Orange**
edged	**0.1**	0.01	-	-	-	-	-	-
tinted	-	-	**0.7**	0.01	0.01	**0.17**	-	-
hatched	-	-	-	-	-	-	-	-

Table 2: Desired output for A-section sample (fig. 2) from an exclusive strata classifier

Since there are four different types of ownership that RoS is interested in, a total of four clas-

sifiers specialised in identifying the colour-pair(s) corresponding to their specific ownership are needed. For instance, the output matrix above (table 2) is tied to a classifier specifically searching for the colour-pair denoting exclusive strata. If provided with the example document from the section on problem definition (fig. 2), the output matrix should produce a relatively high probability for the colour-pair *tinted blue*.

With that being said, a more high level overview of the classification process showing all steps from beginning to end is given in figure 10:

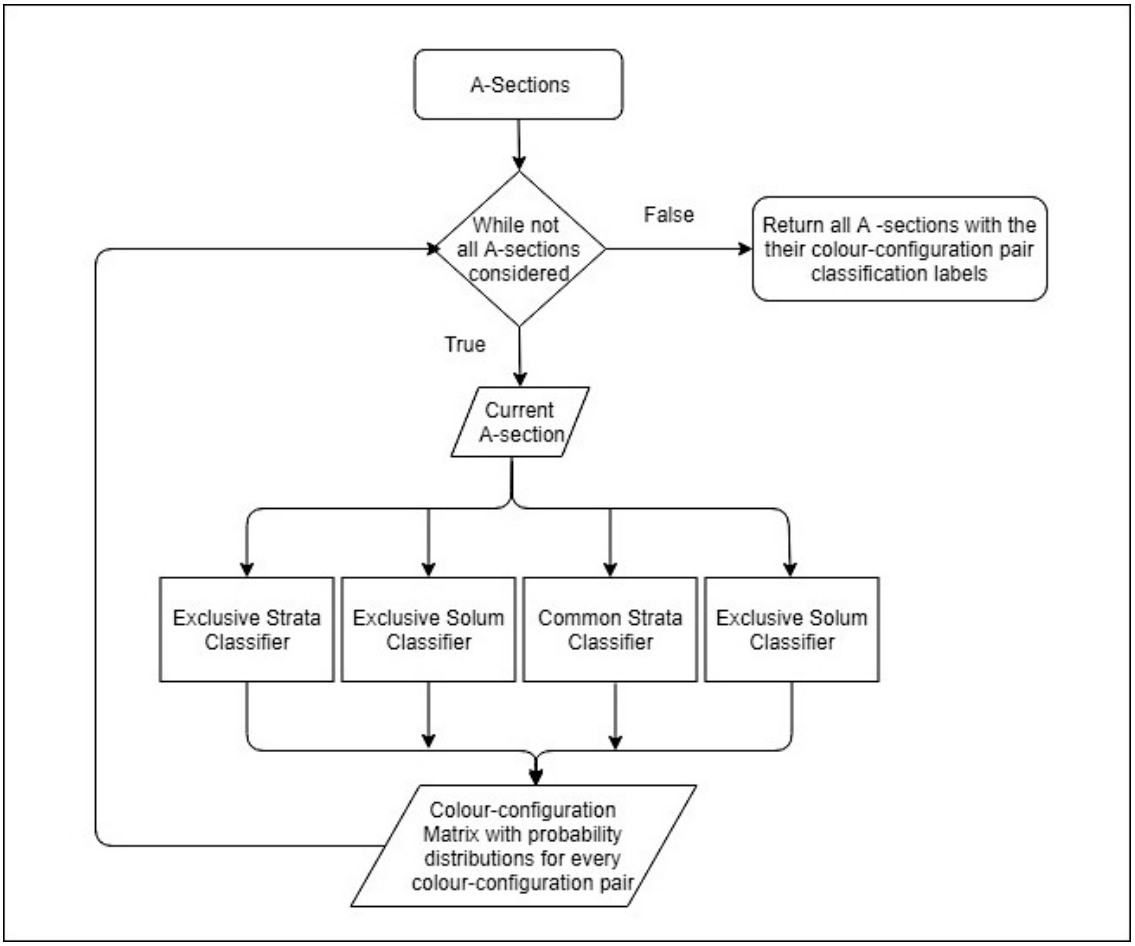

Figure 10: Colour-pair matrix model

In this model, every A-section will be sent through the aforementioned four classifiers. All of the four classifiers will return a matrix containing the probabilities for every colour-pair being their type of ownership. Out of those matrices, the colour-pairs with the highest probabilities will be selected to represent one of the ownerships. The A-sections with ambiguous ownership patterns will receive a flag for future manual review by RoS.

The training process for this classification model follows a similar pattern (fig. 11). However,

it is important to train the classifiers on a balanced set of colour-pairs to avoid potential problems, e.g. biases or no reduction of error in the classifier.

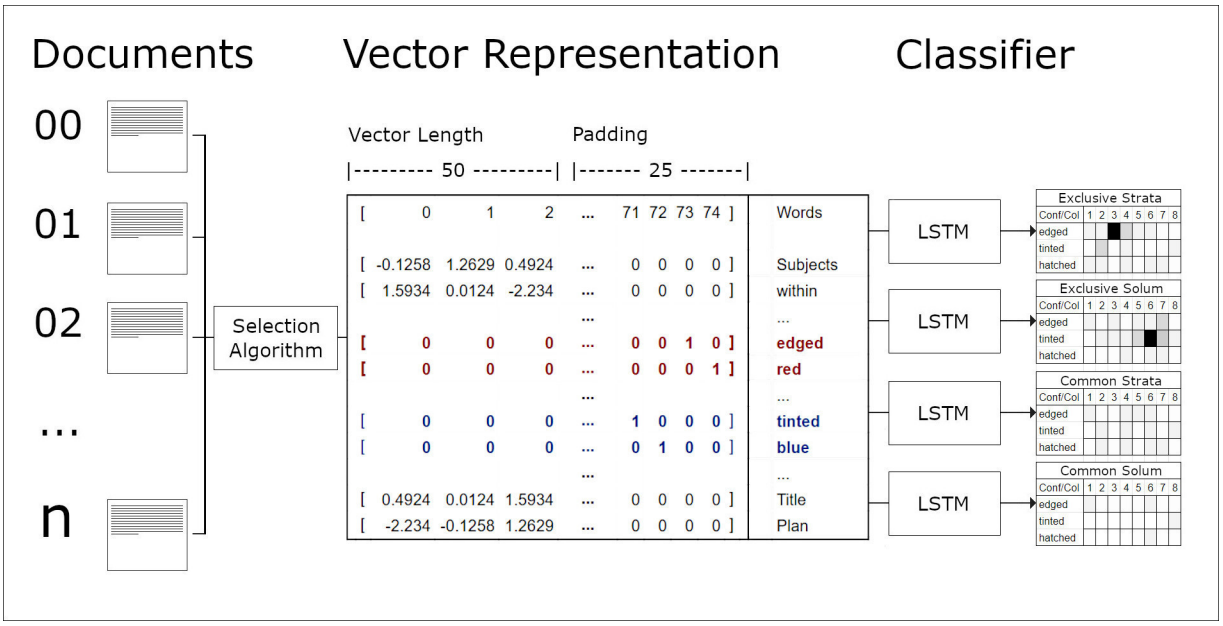

Figure 11: Original structure of the classifier

Since this is the real world environment, it is very unlikely that the A-sections themselves are going to contain a perfectly balanced set of colour-pairs. Hence, a selection algorithm has to be applied to the documents in the training set (fig. 11) to ensure that the classifier does not skew towards a particular colour-pair or sets of colour-pairs.

4.2.2 Series of Binary Classifiers

It is important to note that this section was written with the benefit of hindsight. A lot of the details as to why this particular design was ultimately chosen will be given in the section on annotating the A-sections. This classification model, albeit just having binary outputs, is considerably more complex compared to the first one and consists of multiple stages.

Separating Houses and Flats Firstly, the A-sections are separated according to their type of property (figure. 12). Most properties fall into the category of either houses or flats, but every so often outliers such as shopping centres or estates appear in the data set. Thus, the first binary classifier sorts the A-sections into the categories *outlier* or *non-outlier*, and the second classifier separates the remaining A-sections into flats and houses. Alternatively, having one classifier with three outputs to cover said categories may also be a reasonable approach. However, this idea was not tested because the total amount of outliers in the

provided training data was insufficient. In a future implementation, a decision has to be made concerning the desired implementation for categorising properties. As of now, the only classifier that was implemented was the one distinguishing between flats and houses.

The main reason for separating houses and flats is due to houses not having any strata to speak of. Hence, scanning the different A-sections for exclusive and common strata when a good portion of them are bound to having only exclusive and common solum is not very efficient.

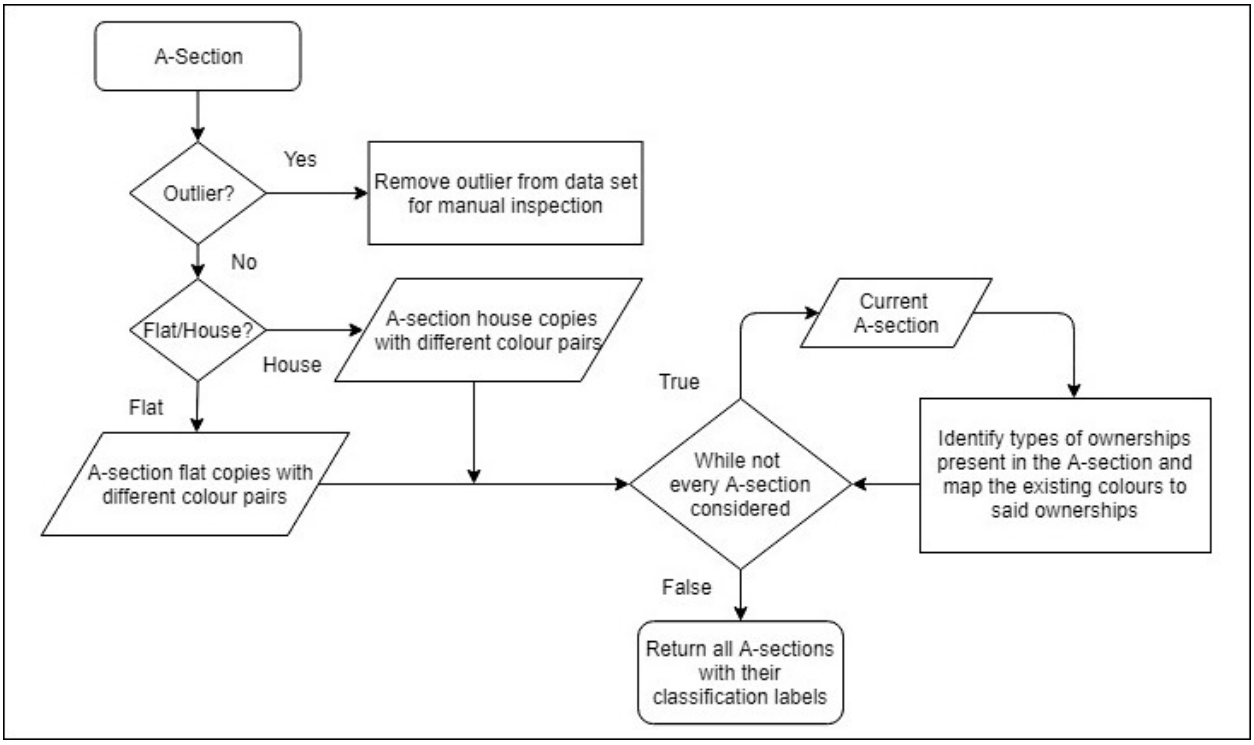

Figure 12: Classification Process Part 1 - Property Type Separation

Confirming Existence of Ownership With the separation of properties now done, the next step in the classification process is the detection of existing ownership types. In other words, each A-section is sent through a classifier searching for the existence of one specific ownership. Once finished, a positive or negative label specifying the existence of a specific ownership is assigned to it at the end (fig. 13). In practice, houses are scanned for exclusive and common solum while flats are scanned for all four types of ownership. To give an example, if there exists a colour-pair within a flatted A-section representing exclusive strata, a positive label for this type of ownership is returned along with the A-section.

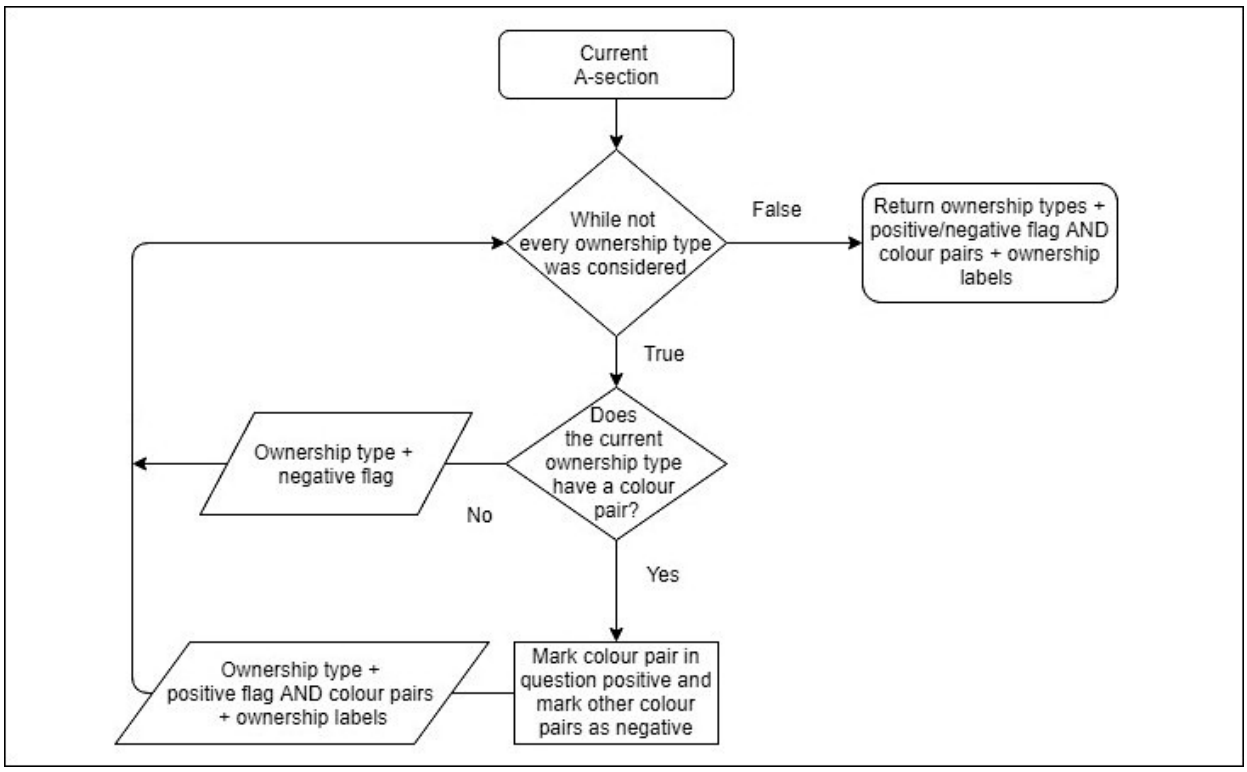

Figure 13: Classification Process Part 2 - detecting existing ownership types

Having this particular setup makes the separation of A-sections into house and flats seemingly unnecessary, since the existence of ownership types within the documents can be detected directly by the above approach (fig. 13) without prior separation. Unfortunately, the reality is somewhat more complex. Quite often, flatted A-sections may have a verbal description of exclusive strata embedded within their text without a corresponding colour-pair. This situation does not occur with A-sections concerning houses. Thus, certain nuances like these may be lost if the type of property is not known to the classifier.

Another reason why separating houses and flats beforehand is likely to yield better results stems from the inherent structural differences between the two property types. Having a clear separation between the two reduces the amount of attributes the classifiers have to learn, allowing them to focus on one specific aspect of the classification process. In fact, this is also the reason why at this point the classifiers are not yet trying to assign specific ownership types to specific colour-pairs. If the existence of a specific type of ownership within an A-section is not known, e.g. because it is verbalised, the classifier has to learn whether said ownership occurs in the A-section or not and if it does, the classifier also has to decide which specific colour-pair represents said ownership. By using the intermediary stage described in figure 13, these two aspects can be tackled separately.

Mapping Ownership to Colour-Pairs Lastly, the final suite of binary classifiers will map the colour-pairs within the given A-sections to their respective ownership. At this point, it is already known which types of ownership actually exist, leading to the following procedure for every A-section (fig. 14):

1. The A-section will be put through every ownership classifier for which it is known that its type of ownership has a colour-pair
2. Every ownership classifier, e.g. the one for exclusive strata, assigns a *yes* or *no* label to every colour-pair it encounters within the text
3. The label *yes* means that the classifier considers the colour-pair to signify exclusive strata, label *no* means that the colour-pair is not considered to be exclusive strata. The same logic applies to all other binary classifiers specialising on different types of ownership.

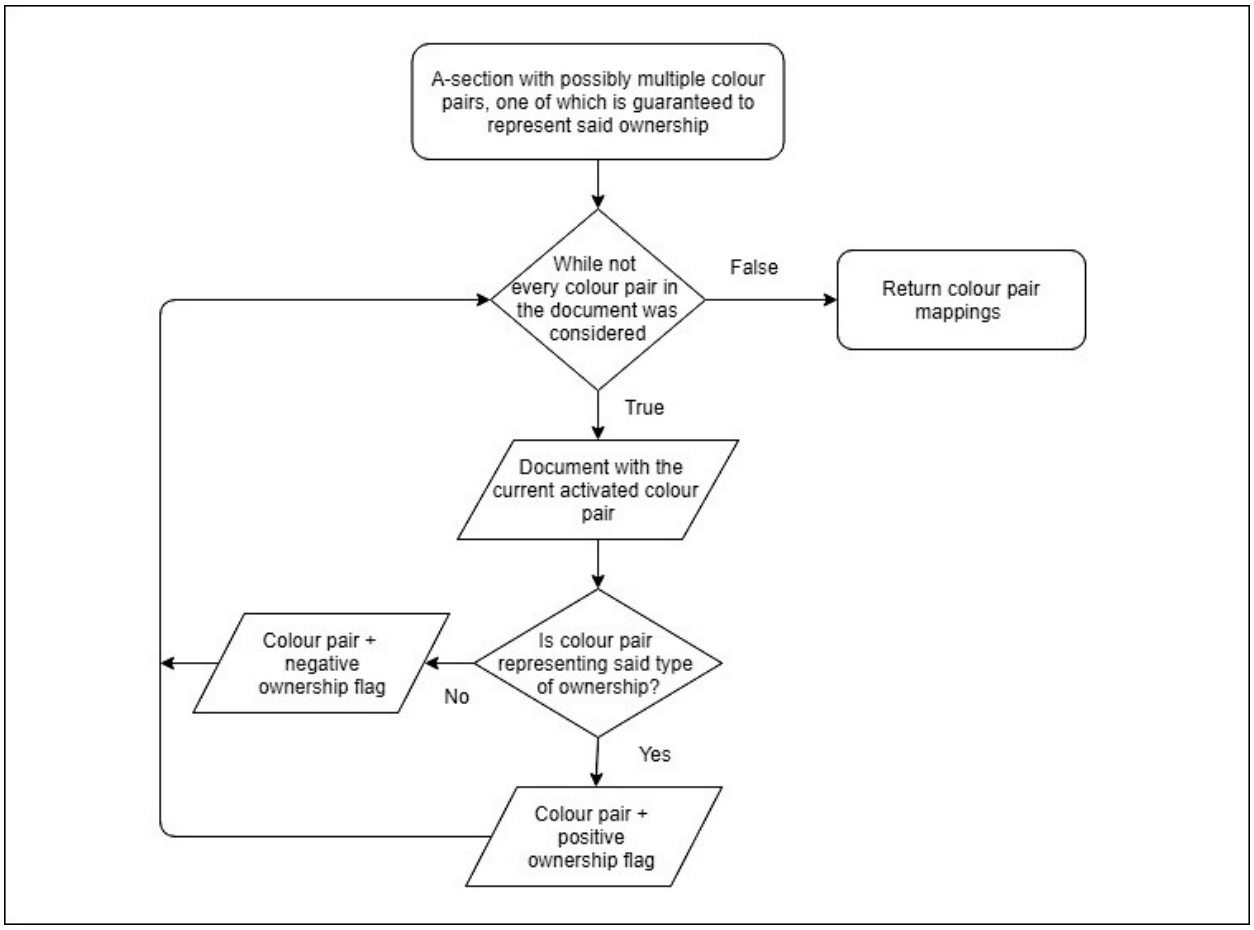

Figure 14: Classification Process Part 3 - Labelling colour-pairs

Parsing A-sections Putting this approach into practice requires a range of adjustments to be made. In the first two rounds, i.e. when separating houses and flats as well as detecting existing types of ownership, the classifiers were working with the full length of the A-sections. More accurately, the entire series of vectors representing the individual words within the A-sections were considered during training and classification. By changing the objective to labelling specific colour-pairs rather than categorising property types or detecting existing ownership, feeding the entire document into the classifier is not exactly useful. For the sake of argument, imagine a document containing the two colour-pairs *edged red* and *tinted blue*. The colour-pair *edged red* represents exclusive strata and *tinted blue* represents exclusive solum. If presented with the colour-pair *edged red*, a binary classifier specialised on exclusive strata should return *yes* as its answer, thus binding *edged red* to exclusive strata. Conversely, this classifier should return *no* if presented with *tinted blue*, while a binary classifier specialised on exclusive solum should return *yes* as its answer.

Thus, the main problem with feeding entire A-sections into each of these classifiers is that they contain multiple colours. When returning *yes* or *no* as the answer, there is no way of knowing which of the colour-pairs actually correspond to the *yes* or *no* label. As a result, the following preprocessing step was proposed (fig. 15):

File number:	9
Active Colour:	Edged Red
Colour position:	2 and 3
Label:	None

Vector/Index	0	1	...	72	73	74	75	Words
0	0.14482473	-0.22482049	...	0	0	0	0	All
1	-0.5941388	0.34713581	...	0	0	0	0	Subjects
2	**0**	**0**	...	**1**	**0**	**0**	**0**	**edged**
3	**0**	**0**	...	**0**	**1**	**0**	**0**	**red**
4	-0.40754583	0.20263372	...	0	0	0	0	and
5	-0.00557029	-0.17700984	...	0	0	0	0	also
8	-0.40754583	0.20263372	...	0	0	0	0	on
9	-0.00557029	-0.17700984	...	0	0	0	0	the

File number:	9
Active Colour:	Tinted Blue
Colour position:	6 and 7
Label:	Exclusive Strata

Vector/Index	0	1	...	72	73	74	75	Words
0	0.14482473	-0.22482049	...	0	0	0	0	All
1	-0.5941388	0.34713581	...	0	0	0	0	Subjects
4	-0.40754583	0.20263372	...	0	0	0	0	and
5	-0.00557029	-0.17700984	...	0	0	0	0	also
6	**0**	**0**	...	**0**	**0**	**1**	**0**	**tinted**
7	**0**	**0**	...	**0**	**0**	**0**	**1**	**blue**
8	-0.40754583	0.20263372	...	0	0	0	0	on
9	-0.00557029	-0.17700984	...	0	0	0	0	the

Figure 15: Duplicating the A-section and switching colour-pairs on and off

The example A-section is duplicated, with every copy containing only one single colour-pair. Both versions are then forwarded to the four different colour-pair-to-ownership classifiers. In an ideal scenario, the result should look as follows (table 3):

Classifier	**Existence**	**Edged Red**	**Tinted Blue**
Exclusive Strata	Yes	Yes	No
Exclusive Solum	Yes	No	Yes
Common Strata	No		
Common Solum	No		

Table 3: Classification labels of flatted A-section after separation of properties

During the implementation it was discovered that mere duplication and omission of colour-pairs as portrayed in figure 15 was not sufficient to train the classifiers effectively. Despite trying different sets of hyperparameters, the classifier's loss did not diminish over time (fig. 16). As expected, the classifier did not perform very well either (fig. 17), with its sensitivity (*yes labels*) and specificity (*no labels*) floating between 40 to 60 percent.

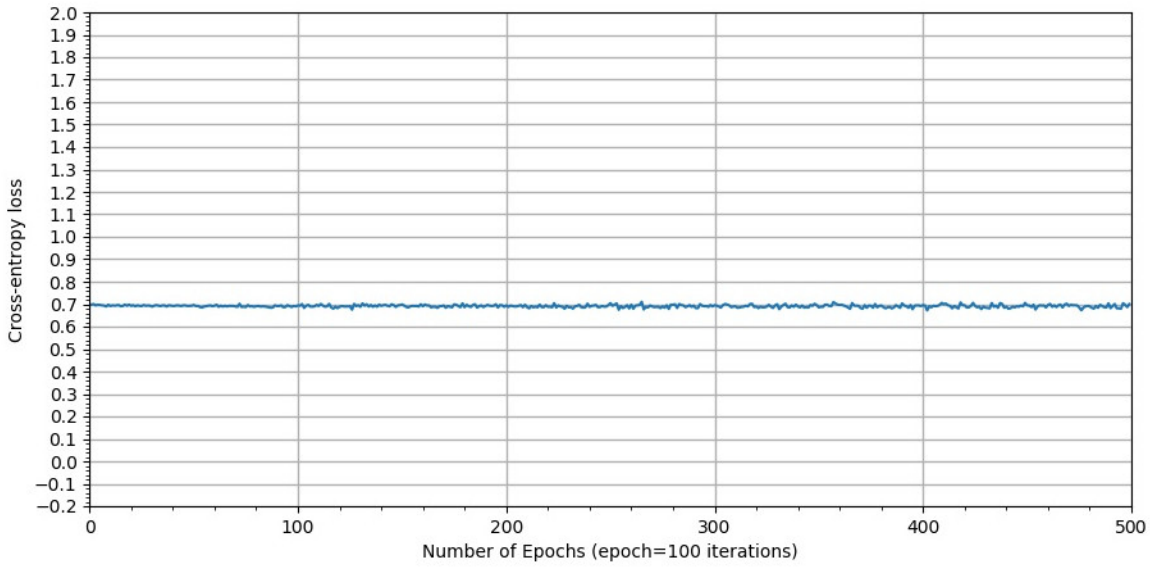

Figure 16: Cross Entropy Loss for labelling colour-pairs as exclusive strata

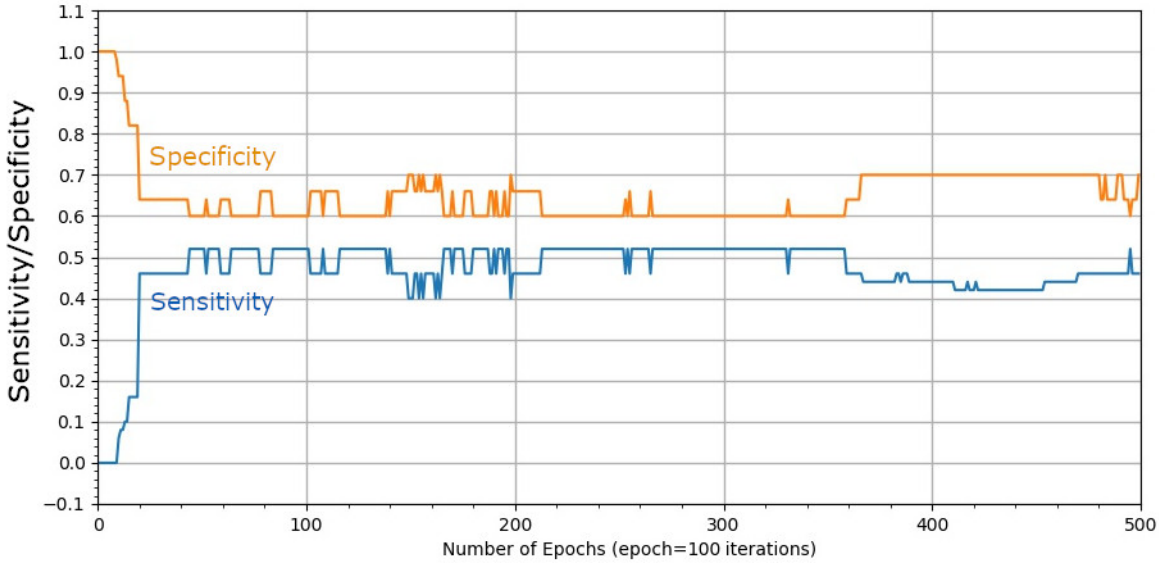

Figure 17: Specificity and Sensitivity for labelling colour-pairs as exclusive strata

This particular example (fig. 16 and 17) was run on exclusive strata with a learning rate of 0.001, vector sizes of 75 and one hidden LSTM layer of size 30. Note that the trained vectors are actually of size 50 with a zero-padding of size 25 attached to them to accommodate the one-hot vectors designated for colour-pairs. Both the test and training set contained 100 A-sections each, with the negative, i.e non-ownership samples, vastly outnumbering the positive samples. To avoid unbalanced training, a sampling algorithm was designed to draw positive samples half of the time. Again, changing these hyperparameters did not result in any significant improvement. Looking at the almost identical copies of A-sections (fig. 15), it became clear that simply activating and deactivating the colour-pairs on its own did not generate enough difference between the copies which the classifiers could pick up upon. To confirm this, a small experiment was run where the classifier was trained on only *yes* samples for a number of epochs before only *no* samples were considered. The results of this alternating training look as follows (fig. 18):

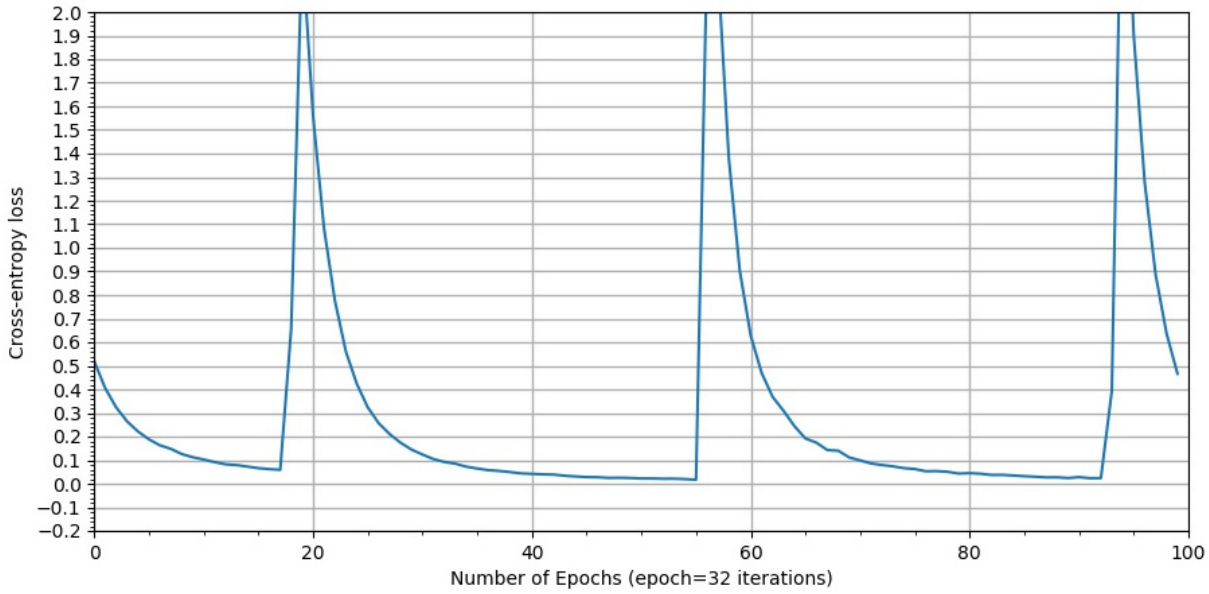

Figure 18: Experiment Loss function

From this experiment, it can be seen that the *yes* samples cancel the learning effect of the *no* samples and vice versa, a strong suggestion that the classifier struggles to learn their differences. To address this problem, a small change had to be made to the duplication process (fig. 19). Instead of simply duplicating the A-sections and activating different colour-pairs, the different copies were truncated after the colour pair that they represent (fig. 19). This resulted in copies of different lengths, making them clearly distinguishable. Also, no colours were omitted; instead, the colour-pair at the very end of the copy was the one which would receive the *yes/no* labels in the end.

File number:	9
Active Colour:	Edged Red
Colour position:	2 and 3
Label:	None

Vector/Index	0	1	...	72	73	74	75	Words
0	0.14482473	-0.22482049	...	0	0	0	0	All
1	-0.5941388	0.34713581	...	0	0	0	0	Subjects
2	**0**	**0**	...	**1**	**0**	**0**	**0**	**edged**
3	**0**	**0**	...	**0**	**1**	**0**	**0**	**red**

File number:	9
Active Colour:	Tinted Blue
Colour position:	6 and 7
Label:	Exclusive Strata

Vector/Index	0	1	...	72	73	74	75	Words
0	0.14482473	-0.22482049	...	0	0	0	0	All
1	-0.5941388	0.34713581	...	0	0	0	0	Subjects
2	**0**	**0**	...	**1**	**0**	**0**	**0**	**edged**
3	**0**	**0**	...	**0**	**1**	**0**	**0**	**red**
4	-0.40754583	0.20263372	...	0	0	0	0	and
5	-0.00557029	-0.17700984	...	0	0	0	0	also
6	**0**	**0**	...	**0**	**0**	**1**	**0**	**tinted**
7	**0**	**0**	...	**0**	**0**	**0**	**1**	**blue**

Figure 19: Duplicating the A-section and removing everything after the last colour-pair

In the case of figure 19, the copy to the left represents colour-pair *edged red* and the copy to the left represents colour-pair *tinted blue*. The desired set of labels for these two colour-pairs is still the same (table 3).

This particular modification proved to be very successful. As can be seen in figure 20, the classifier's loss does in fact diminish over time, albeit never reaching zero completely. The specificity of the classifier is at 98% and its sensitivity at 92% by the end of epoch 500. Overall, this classifier was trained with exactly the same hyperparameters as the one using the original duplication method in figure 15. With this performance, is that most of the relevant context for determining a colour-pairs type of ownership comes before the colour-pair itself.

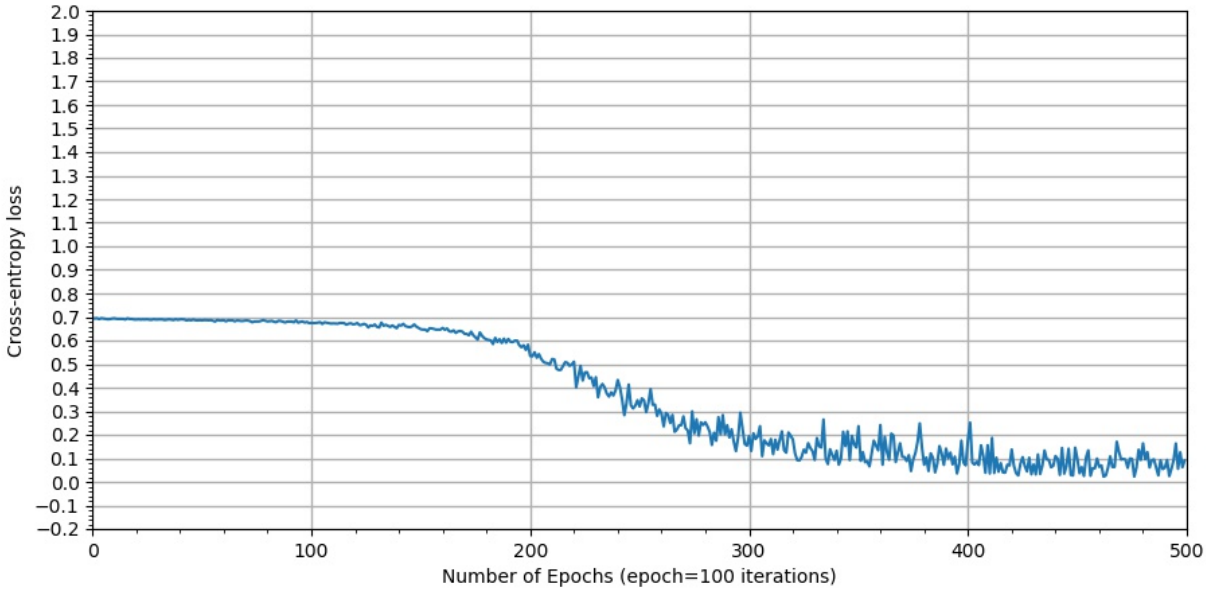

Figure 20: Loss over time for labelling colour-pairs as exclusive strata

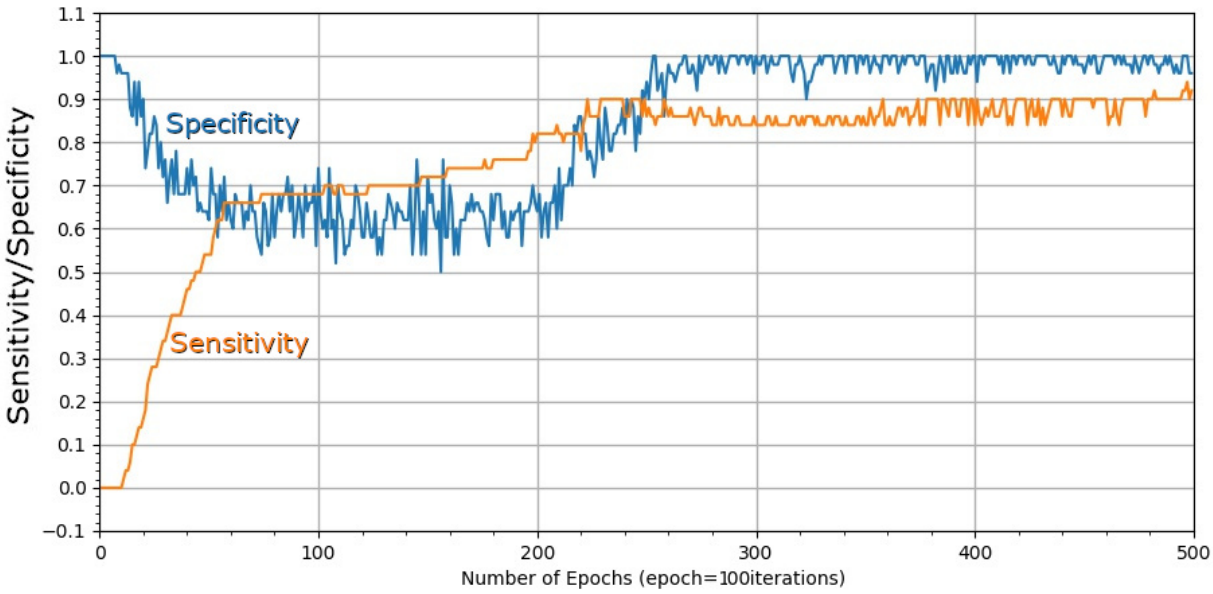

Figure 21: Sensitivity and specificity for labelling colour-pairs as exclusive strata

Advantages and Disadvantages of the Binary Classifier The main drawback of the binary classifier model is its complexity. If one looks at the general process (fig. 22), the initial number of classification steps for a flatted A-section containing colour-pairs for all four ownership types is six. These six steps do not include assigning *yes* or *no* tags to every colour pair. With four types of ownership in place, there are at least four different colour-pairs within the text for every ownership. Thus, every colour-pair classifier would have to iterate through the A-section four times to assign these labels. In total, this equates to 16 classification steps. Putting everything together, this particular example has to run through a wide range of binary classifiers 22 times before its colour-pairs have been mapped to one of the types of ownership.

On the flip side, this model offers a lot of flexibility in terms of optimisation. Firstly, every binary classifier has a range of hyperparameters that can be adjusted empirically to get the best possible performance out of every classification step. Secondly, this model can be expanded relatively easily without having to retrain all of its components. The main goal of this dissertation was extracting ownership patterns from text, but this does not have to stop here. In fact, certain colour-pairs can denote right of access and very often, there are descriptions within the text cross-referencing other documents, e.g. supplementary plans. Lastly, there are also colour-pairs denoting tenement steading. If RoS is interested in extracting more information from its A-sections, this can be easily incorporated into the existing model. In addition, classifiers not based on LSTM's can be tried on the different nodes of the decision tree (fig. 22) if deemed necessary by RoS.

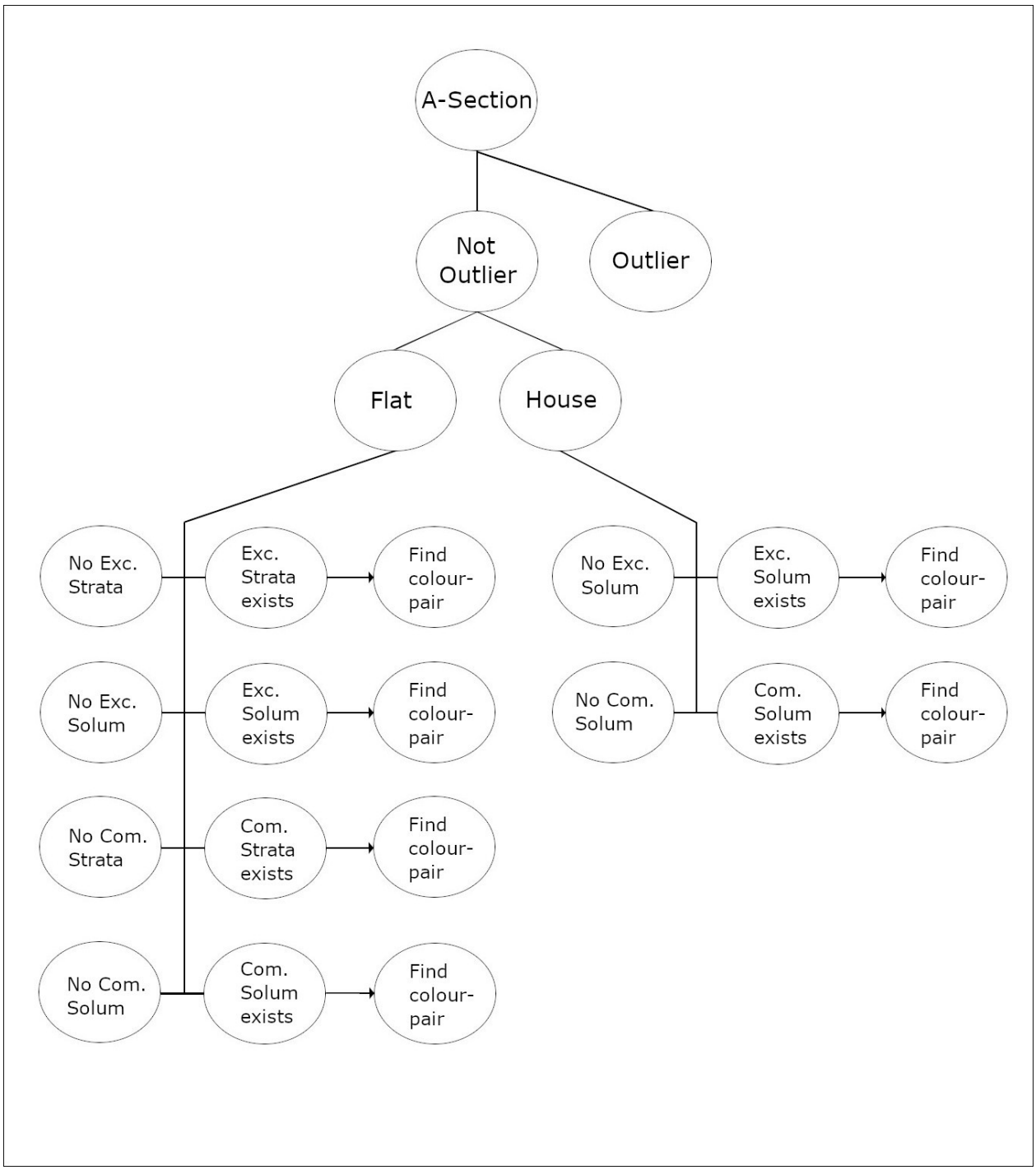

Figure 22: Binary Classifier Decision Tree for one A-Section

5 Testing the model on IMDB movie reviews

Doing sentiment classification on movie reviews was not the main purpose of this project. Yet, there were several strong arguments for testing the model on a publicly available data set. Firstly, getting access to RoS data required the signing of a non-disclosure agreement by RoS, the university and the student himself. As a result, the A-section data was not available until halfway through the dissertation. Secondly, testing the model on a data set where the model was already proven to work could provide certain insights and transferable knowledge on how to proceed with RoS's data. Lastly, there was another unexpected benefit of using two different sets of data. When implementing the model on the IMDB data set, a certain bug went undiscovered for some period of time, rendering most of the test results useless. Once RoS's data became available and the implementation had to be adapted, the bug was discovered and subsequently removed. Without having a direct comparison to work with, the danger of producing fake results with RoS's data could have been significantly higher.

5.1 IMDB Data Set

This data set was first introduced by a paper on *Learning Word Vectors for Sentiment Analysis* (Maas et al., 2011b) and consists of a total of 50,000 movie reviews (Maas et al., 2011a). These reviews are split evenly into a training and test set of size 25,000 respectively. In addition, the movie reviews are also split into positive and negative reviews based on a score between one and ten. A movie review is classified as positive if it falls within the range between seven and ten. Conversely, if the movie review happens to be between one and four it is labelled as negative. Movie reviews with the labels five and six do not appear in the training set, but they are in fact present in another unlabelled set of the same size as the training set.

For the intents and purposes of this training exercise, 1,000 to 5,000 movie reviews will be used as the training set. The main data set containing the legal documents is limited to 5,000 documents as well. Larger numbers of files would also slow down the learning process considerably. Ultimately, both data sets need to have similar proportions so that some form of transferable knowledge concerning the Word2vec and LSTM parameters can be extracted. By the time the actual data becomes available, the results obtained from these experiments will provide some guidance for selecting a set of training hyperparameters.

5.2 Training Hyperparameters

In this section, both the Word2Vec model as well as the LSTM classifier are put into practice for the first time. Naturally, some form of seletion had to be done concerning both models' hyperparameters without indulging in initial trial and error. Hence, the selection of vector sizes and their training times were based on experiments done in Baroni's paper (2014) and

recommendations from Mikolov (2013) concerning the vectors' context window.

5.2.1 Word2vec

The corpus consisted of 2,500 negative and positive movie reviews, resulting in a total of 5,000 samples. In total, this corpus contained 1,116,359 words with 46,591 unique embeddings. Therefore, the size of the input layer to the word2vec model was set to be 46,591. All other hyperparamters for generating the individual vectors are given below in table 4.

Vector Set	Hidden	Window	Epochs	Iterations per Epoch	Learning Rate
1	100	2	100	5,024	0.01
2	100	5	100	5,024	0.01
3	200	2	100	5,024	0.01
4	200	5	100	5,024	0.01
5	300	2	100	5,024	0.01
6	300	5	100	5,024	0.01

Table 4: Different vector configurations generated by Word2vec in chronological order

It was observed that smaller window sizes and larger dimensions resulted in a steeper decline of the loss graphs (fig. 23). This is not too surprising if one considers the additional difficulty of predicting a larger number of neighbouring words and larger dimensions carrying more information. The LSTM classification model was then tested on these different vector sets, albeit without much success due to the aforementioned problem with the bug.

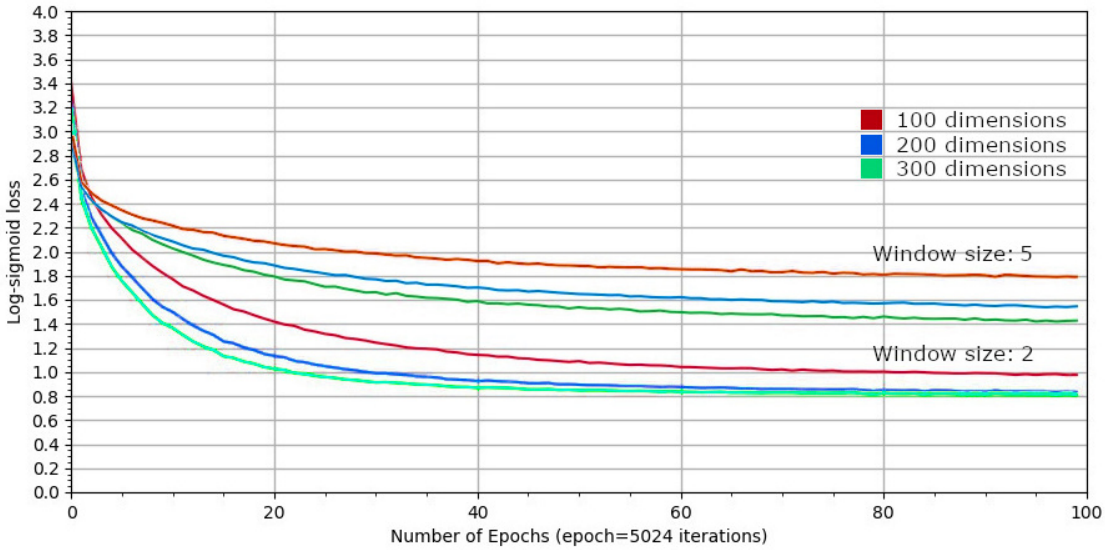

Figure 23: Loss over time for different sizes of hidden layers

5.3 Results

As mentioned previously, the model was not working properly due to a bug that went undiscovered for some period of time. When this problem was finally fixed, the attention was immediately shifted to the RoS data set. For this reason, only two experiments will be shown in this section. The first one will depict the classifier's behaviour during training before the bug was fixed and the second one the situation afterwards.

5.3.1 LSTM Classifier

Hyperparameters	**Run 1**	**Run 2**
Vector Set	1	2
Training Samples	1,000	1,000
Learning rate	0.002	0.075
Input layer size	100	100
Hidden layer size	30	30
Output layer size	8	2
Epochs	200	50
Iterations per epoch	1,000	1,000
Accuracy at the end	14,6 %	74.1 %

Table 5: LSTM Training Hyperparameters

Table 5 depicts the general experiment setup before the bug was discovered in run number one and the experiment setup after the bug was removed in run number two. Note that the output layer size was changed to two from the orginal eight in the end to simplify the process as much as possible. An output label of zero denotes a negative sentiment, whereas an output label of one denotes a positive sentiment.

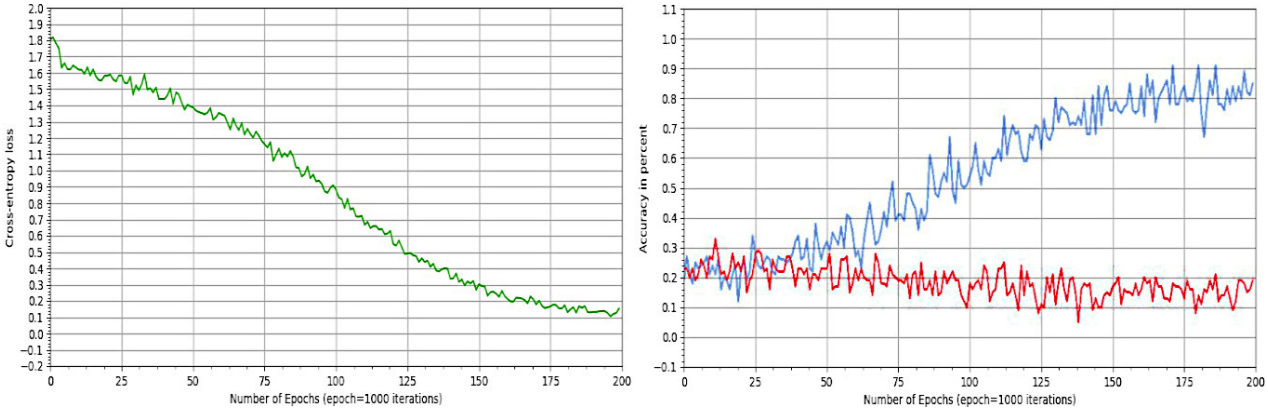

Figure 24: Cross-entropy loss over time

In figure 24, it can be seen how the green loss graph diminishes to almost zero over time, which indicates that the classifier is actually learning from the training data. On the flip side, the performance of the classifier in terms of accuracy did not improve at all. In this figure, the blue graph represents the accuracy on classifying the training set. The accuracy on the test set represented by the red graph, however, did not improve at all.

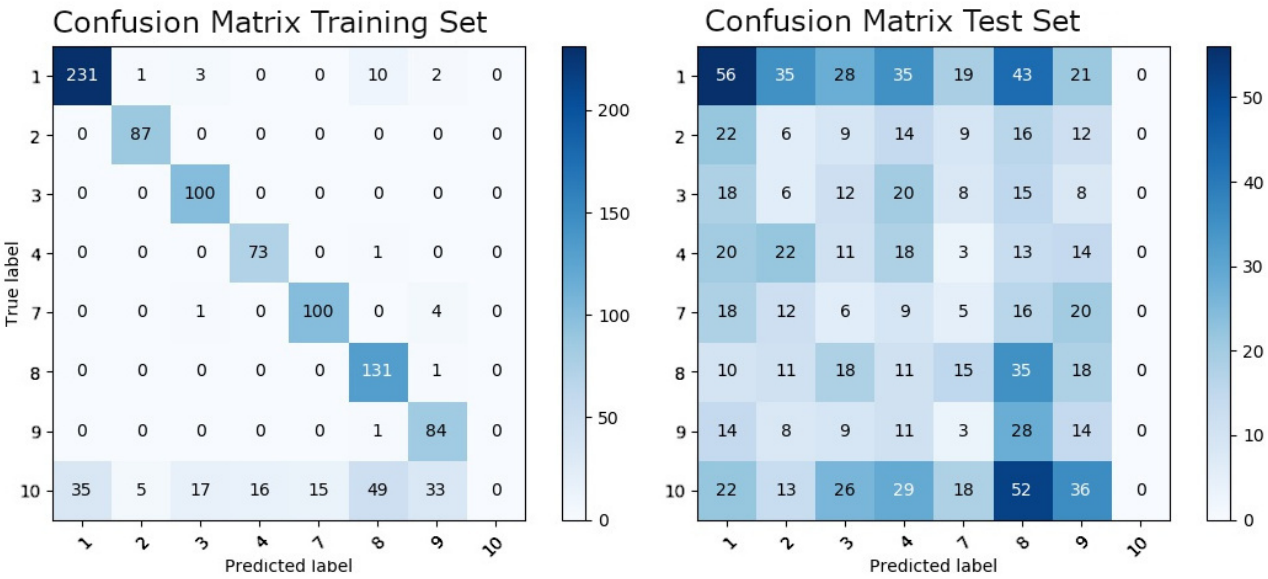

Figure 25: Confusion matrix for both data sets

When looking at the confuson matrices for both the training and test set (fig. 25), one gets a very similar picture. The classifier learns to assign the right labels to the training set, yet fails to apply its learned patterns to the test set.

After a range of different test runs and simplifications of the classification model, e.g. by reducing the output layer, it was finally discovered that the classifier only read the first couple of words from every movie review in the training set. When this bug was fixed and the model was rerun, the classifier was finally able to generalise and showed positive results on the test set (fig. 26).

Having spent a great amount of time on fixing this error, it was then decided to abandon the work on the movie review data set in favor of working on the A-sections, which became available in the meantime. After all, the main purpose of using the movie review data set was to bridge the implementation gap and to get some initial test results before pursuing the main goal of this dissertation.

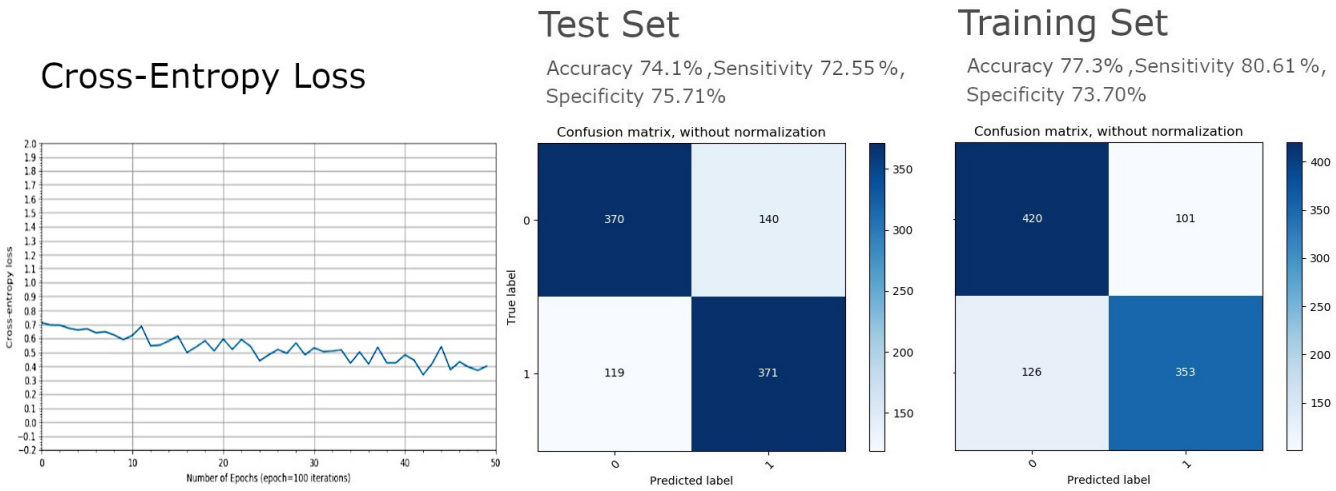

Figure 26: Results of rerunning the model using the fixed code

6 Running the model on the RoS data set

With the results obtained from the IMDB data set being positive and the RoS data having been transferred onto Cirrus, i.e. Edinburgh University's supercomputing facilities, work commenced on getting the classification process for RoS up and running.

6.1 RoS data set

The Registers of Scotland provided a total of 5,000 A-sections on houses and 5,000 A-sections on flats from both Glasgow and Edinburgh. The entire data set was randomised, i.e. there was no specific order associated with the A-sections. As outlined in the problem definition section, there are four types of ownership that RoS is concerned about, namely, exclusive strata, exclusive solum, common strata and common solum. In the case of houses, these four types of ownership get reduced to exclusive and common solum, since there exists no strata to begin with.

6.1.1 Tagging Process

Before any form of training and classification can be done, every document requires its colour-pairs to be mapped onto their respective ownership labels. An example of this process can be seen in figure 27, where a bespoke annotation tool is used to assign and save said labels.

Figure 27: Annotation tool in use

In addition to the four types of ownership, there are also text boxes for specifying the type of

property, the tenement steading and additional information, e.g. the cross reference to other documents such as supplementary plans. While these labels are not of direct importance to the colour pair classification process, they do provide information that may be used later within a different context, for example when investigating potential reasons as to why certain documents were misclassified. For instance, certain documents mentioned identical colour-pairs more than once, but every time this occurred the colour-pair was either referring to a supplementary plan or the original A-section. Investigating these types of cross-references and potentially coming up with another set of classifiers to deal with them could be of potential interest to RoS.

In about two weeks, 200 documents representing houses were annotated, along with the same amount of documents representing flats. Given the overall complexity of flatted properties, it was decided by RoS that annotating houses first would provide sufficient training in domain knowledge to be able to shift to flats at a certain point. Having 200 samples of data, especially when partitioned into a training and test set, is not quite enough to train a classifier with solid generalisation capabilities. However, it turned out to be sufficient for getting a representative distribution of tags and a good grasp of the recurring patterns in the legal text.

6.1.2 Distribution of Properties

When RoS created these two data sets, an algorithm for reliably detecting different properties did not exist . Thus, a simple heuristic technique was used to select flats and houses based on parameters such as document length, complexity and certain keywords. Surprisingly enough, the accuracy for selecting either houses or flats was very high, albeit not without error.

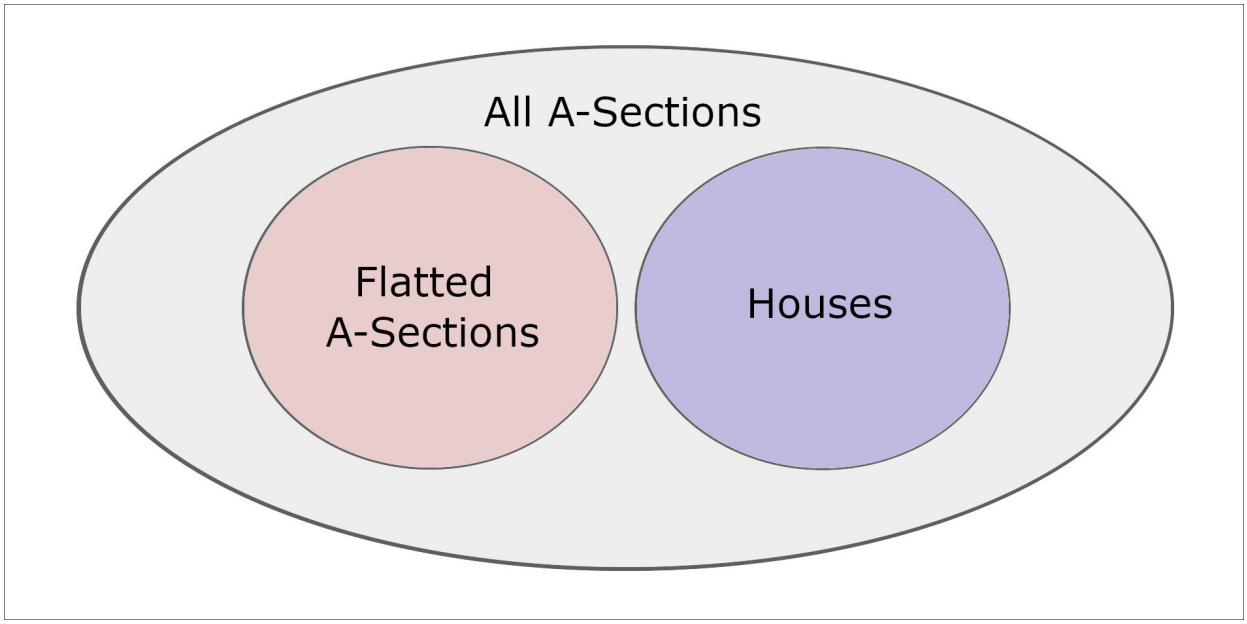

Figure 28: Flats and houses in relation to other types of A-sections

The data sets of 5,000 houses and flats only represent a small fraction of the total amount of A-sections. Currently, the exact proportion of houses, flats and other types of properties is unknown. When scaling this system up, a more fine-grained separation and training procedure has to be taken into consideration to ensure that all properties are properly categorised.

Houses		**Flats**	
Property Type	Frequency	Property Type	Frequency
House	185	Flat	197
Estate	9	Shopping Centre	2
Land	4	Unknown	1
Flat	2		

Table 6: Distribution of property types for 200 houses and flats

When going through the house samples manually, it was found that out of the 200 house samples, 185 samples in fact represented houses (tab. 6). Still, nine A-sections were describing estates, four of them were describing pieces of land and two of them were flats. On the flatted side, the situation looked slightly better with only three samples out of 200 not being flats. Regarding the classification process for flats and house, this opens up two possibilities. The first one would be to simply take both data sets without annotating anything and simply staring the training process. Here, the main advantage is the availability of a sufficiently large data set without having to spend time on manually annotating A-sections as flats or houses. Conversely, the downside of this approach would be the acceptance of a small percentage of training samples carrying the wrong label. The second approach would use the much smaller data set of 200 annotated flats and houses, with the upside being the absolute certainty concerning the correctness of the labels. In the end, both approaches were put to the test.

6.1.3 Distribution of Tags

Figure 29 and 30 depict the distribution of tags for all of the four types of ownership for the aforementioned 200 flats. It is clearly evident from these figures that the overall data set is highly imbalanced, with the majority of tags for every type of ownership being either *none* or *verbal*. More specifically, 147 out of 200 exclusive strata tags are *verbal*, 166 out of 200 exclusive solum tags are *none* and 188 out of 200 of tags are *none* for common strata. Even common solum, an ownership type often described with multiple colours at a time has a large overhead of 137 none-type tags.

Both *verbal* and *none* describe a state where no colour pair exists to describe the type of ownership in question. The difference between the two is that *verbal* is an indicator for a textual description of ownership being present within the document, while *none* simply states that a given type of ownership does not occur within a document at all. Altogether, the given distribution of ownership tags turned out to be a problem for the original colour-pair

matrix classification model as stated in the section on implementation design.

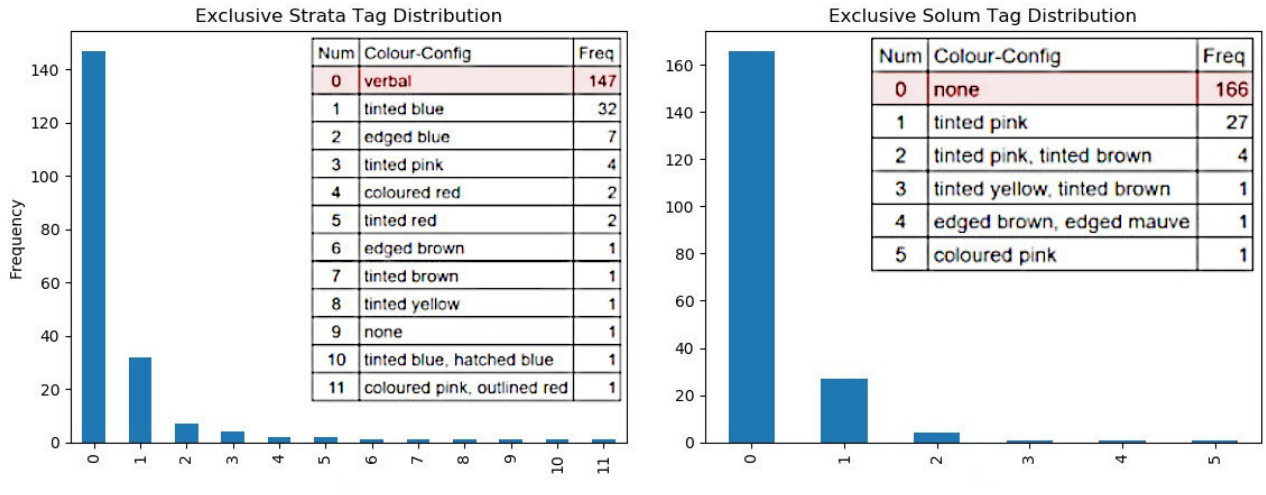

Figure 29: Exclusive ownership tag distribution for the first 200 annotated flats

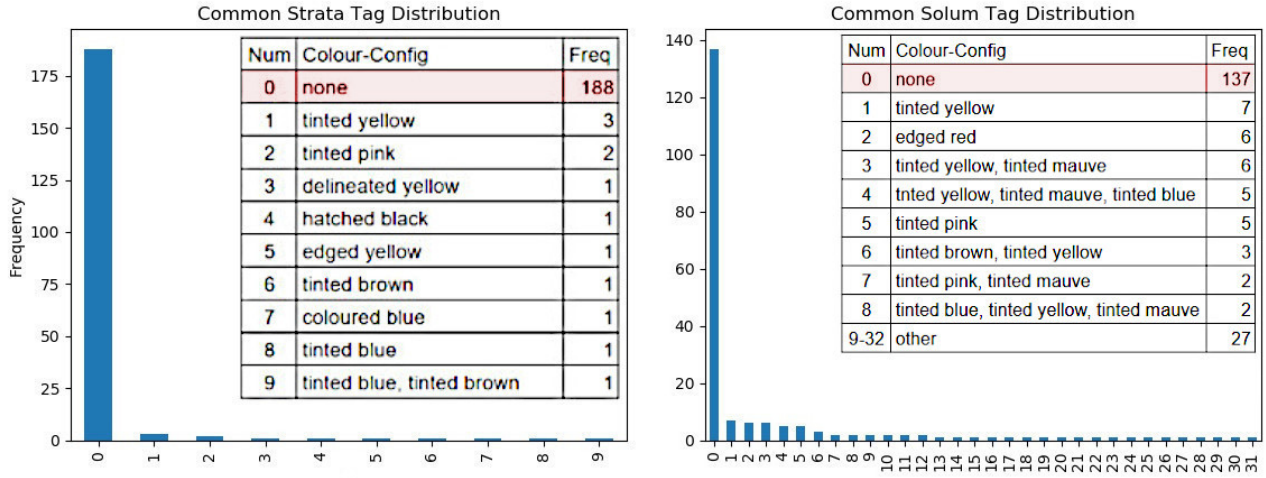

Figure 30: Common ownership tag distribution for the first 200 annotated flats

Firstly, the combination of imbalance and the low number of samples leads to a significant level of sparsity among the remaining colour pairs. Secondly, 11 different configuration tags and 14 different colours were observed during the annotation process, resulting in an output matrix of size 154. Thirdly, with so few colour pairs to speak of it is likely that these colour pairs will be linked to the specific type of ownership that they are observed with during the training process. The latter is problematic in the sense that the classifier will not be able to pick up the context around said colour pairs and will instead assume that the mere occurrence of a specific colour pair is indicating the presence of a specific type of ownership.

One way of mitigating the issue of context is to increase the total number of documents by creating copies of said documents with randomly generated colour-pairs (algorithm 1). However, despite the usefulness of this approach, the problem of sparsity and overly large output matrix still remains, which can be seen in table 7. In this particular example, 33 documents guaranteed to contain colour-pairs denoting exclusive strata were selected from the original 100 A-sections within the training set. Subsequently, 31 different copies were generated for all of the 33 documents, resulting in a training set of size 1023. Still, the number of examples per single colour-pair remains very low and sometimes is even zero.

In conclusion, one would need very large hidden layers due to the size and complexity of the output matrix along with a large number of training samples to address the sparsity problem. For one, a lot of time would have to be spent on annotating at least several thousand documents. Additionally, the computational resources required to facilitate training for a model of this complexity are simply not available for a project of this scope. Thus, it was decided to abandon the output matrix approach in favor of using a series of binary classifiers.

Algorithm 1: Duplicating A-sections and randomly replacing colour-pairs

```
duplication_factor = 10;
for i = 0; i < duplication_factor; i++ do
    for every A-section in the training set do
        copy <- duplicate the current A-section;
        copy_labels <- duplicate the current A-section's labels;
        for every colour-pair in copy do
            new_colour_pair <- randomly generate colour-pair;
            replace current colour-pair with new_colour_pair in copy;
            replace current colour-pair with new_colour_pair in copy_labels;
        end
        write copy and copy_labels back to memory;
    end
end
```

Config/Colour	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
edged	4	11	3	11	5	4	2	2	4	13	4	0	2	6
tinted	10	1	13	5	9	7	3	7	3	16	3	2	13	5
coloured	7	5	3	12	6	11	19	3	14	6	11	3	5	7
hatched	10	3	18	6	8	3	19	9	6	11	5	6	7	10
cross hatched	5	14	1	4	2	6	8	4	9	3	4	4	2	4
squared	6	9	10	0	4	1	1	5	8	11	3	8	7	3
mottled	9	4	5	8	3	9	6	7	3	11	3	4	4	6
delineated	3	10	7	4	11	4	8	12	12	7	2	2	3	1
outlined	7	3	7	11	9	6	5	5	9	13	5	5	0	7
shaded	3	8	12	3	6	5	12	3	24	14	5	4	4	7
dotted	9	15	19	2	9	3	11	7	15	6	11	3	14	13

Table 7: Sparsity and size problem of colour-configuration pair output matrix

However, this is not to say that the mechanism for creating copies of A-sections while randomly generating new colour-pairs was not put to into practice (algorithm 1). Apart from allowing to test the hypothesis of specific colour-pairs being associated with specific types of ownership, it can also be used as a tool to increase the total amount of training samples. Given the problem of sparsity concerning colour-pairs (fig. 29 and 30), artificially increasing the total number of samples actually proved to be very useful.

6.2 Vectorisation Process

Before any form of classification can commence, the words contained with the A-section have to be converted into vector representations. When this process was applied to the RoS data set, it became quite apparent that the total amount of unique words for a similar number of samples was drastically lower compared to the IMDB data set (fig. 8). All of the word embeddings, i.e. the number of unique words, and the total number of words were drawn from 5,000 samples of flats, houses or movie reviews to make sure that these metrics are comparable.

Houses		**Flats**		**IMDB**	
Embeddings	Words	Embeddings	Words	Embeddings	Words
6,259	380,813	8,222	1,179,267	46,591	1,116,359

Table 8: Vector embedding sizes and number of words for houses, flats and movie reviews

Interestingly enough, the number of total words encountered in 5,000 flatted A-sections is slightly higher than the same number for movie reviews. Yet, the number of word embeddings for movie reviews is larger than the one for flatted A-sections by a factor of 5.7. This points to a important difference between these two data sets: A-sections in general, whether they are representing flats or other types of properties, are far more repetitive and structured compared to movie reviews. The latter has a strong tendency to contain a lot of typos, slang and overall a great deal of everyday language which does not appear in the narrowly defined vocabulary used to describe properties.

Another interesting observation was made when a dictionary of vectors was trained on 5,000 flats and then used to convert A-sections of houses into their vector representations. Previously, when training the vectors on movie reviews and using them on an untouched test set, there was always a certain percentage of words for which no vector existed. In the case of training vectors on flats and using them on houses, there was not a single word without a corresponding vector. Evidently, there are no words within the 5,000 A-sections concerning houses that do not appear in 5,000 flatted A-sections (fig. 31). It is important to note, however, that this only occurred when the preprocessing methods outlined at the beginning of section four on implementation design were carried out.

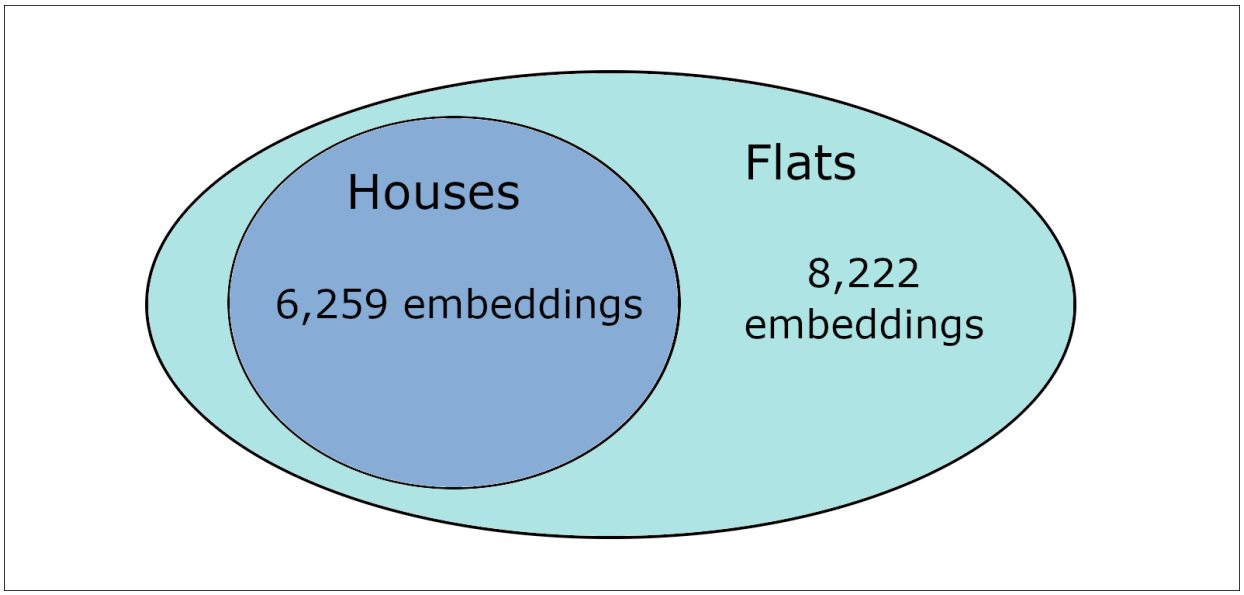

Figure 31: Embedding sets for houses and flats

Nevertheless, the fact that the embeddings for houses are a perfect subset of flatted embeddings is quite striking. This is once again a strong indicator of a highly structured language with a limited vocabulary size.

The vector embeddings used for the movie reviews were ranging from 100 to 300 dimensions and were trained on window sizes of either two or five. (expand here once results from imdb are confirmed) To keep the training process manageable, the initial embedding size was chosen to be 50 with a context window of size 7 to make up for the loss of information.

Training set	5000 flatted A-sections
Words	1,179,267
Embeddings	8,222
Embedding dimension	50
Padding dimension	25, i.e. 14 colours and 11 styles or configurations
Final dimension	75
Window size	7
Learning rate	0.01
Number of epochs	100
Iterations per epoch	5024

Table 9: Vector Training Hyperparameters

6.3 Classifiying A-sections as Houses or Flats

The first step in the classification process is the separation of houses and flats. As mentioned previously, outliers such as estates or shopping centres were not considered as a third category due to the general lack of training data on said outliers. Previously, two approaches concerning the structure of the training set were discussed. The first would be to use a larger, not manually annotated training set with a small percentage of wrongly labelled samples, whereas the second one would create its training and test set from the 400 samples of annotated houses and flats with no wrong labels. Both approaches were tested and will be discussed in this section.

6.3.1 First Experiment Setup

1. **Training set**:
   500 vectorised flats and 500 vectorised houses
2. **Test set**:
   50 vectorised flats and 50 vectorised houses. Training and test set are completely separate and do not overlap
3. **Training data sampling method**:
   A random integer was generated between 0 and 1000 and used as the file index. Since the data set is balanced, the distribution of flats and houses shown to the classifier during training is roughly equal over large enough periods of time
4. **Training duration**:
   2000 epochs and 100 iterations per epoch
5. **Learning rate**:
   The learning rate was chosen to be 0.0005, which is very low. However, higher learning rates resulted in unstable learning behaviour.
6. **Network Layers**:
   The input layer is of size 75 and the hidden layer, i.e. the LSTM layer, was chosen to be either of size 30 or 50 depending on the experiment. The output layer has only two nodes, since this is a binary classifier
7. **Number of hidden layers**:
   In this experiment, one, two and three layers were tried in conjunction with different layer sizes

6.3.2 First Experiment Results

Two different layer sizes were tried as well as three possible network depths to get an overview of what a good architecture might look like for this particular task. In short, the different

LSTM setups are referred to as runs.

The general performance of this classifier was mostly satisfactory. From the graphs depicting the accuracy on the test set for every particular run over time (fig. 32), it can be seen that the learning process itself is still somewhat unstable. Often times the accuracy climbs to a peak and drops off sharply before bouncing back again, even though a deliberately low learning rate was chosen. When comparing the accuracy curves with their respective cross-entropy loss over time (fig. 32), this problem becomes even clearer. Still, there are certain times where the accuracy reaches stable levels over extended periods of time. The highest accuracy observed for every single run is shown in table 10 along with the epoch where it occurred. Also, specificity represents houses, whereas sensitivity represents flats.

LSTM Setup			**Best Overall Results for Accuracy**			
Run	Layer Size	Layers	Sensitivity	Specificity	Accuracy	At Epoch
1	30	1	0.92	0.92	0.92	1,631
2	30	2	0.94	0.92	0.93	1,724
3	30	3	0.94	0.86	0.90	1,966
4	50	1	0.94	0.84	0.89	1,976
5	50	2	0.92	0.94	0.93	1,559
6	50	3	0.92	0.86	0.89	1,567

Table 10: House/Flat Classifier hyperparameters and the test set results

The reason why it was decided to find the spot with the optimal level of accuracy instead of specificity or sensitivity is very simple: High levels of specificity have a strong tendency of resulting in low levels of sensitivity and vice versa. Also, accuracy is a reasonable measure in this particular instance, since the data set is perfectly balanced. This is not going to be the case with the classifiers looking for ownership patterns, a problem that can easily understood by looking at the distribution of colour-pairs (fig. 29 and 30).

In general, it can be said that the performance results are very similar for every single run (tab. 10). Combined with the high levels of instability, it is very difficult to say with certainty if additional layers, differently sized hidden layers or a combination of both actually have a significant impact on performance. In order to arrive at a conclusive answer, the classifiers would have to be trained at least several times to be able to observe any pattern that would surface as a result of averaging the results, e.g. via cross-validation. What will be done, however, is rerunning the experiments on the manually annotated data set to see if the problem of instability is also linked to the occurence of wrongly labelled samples.

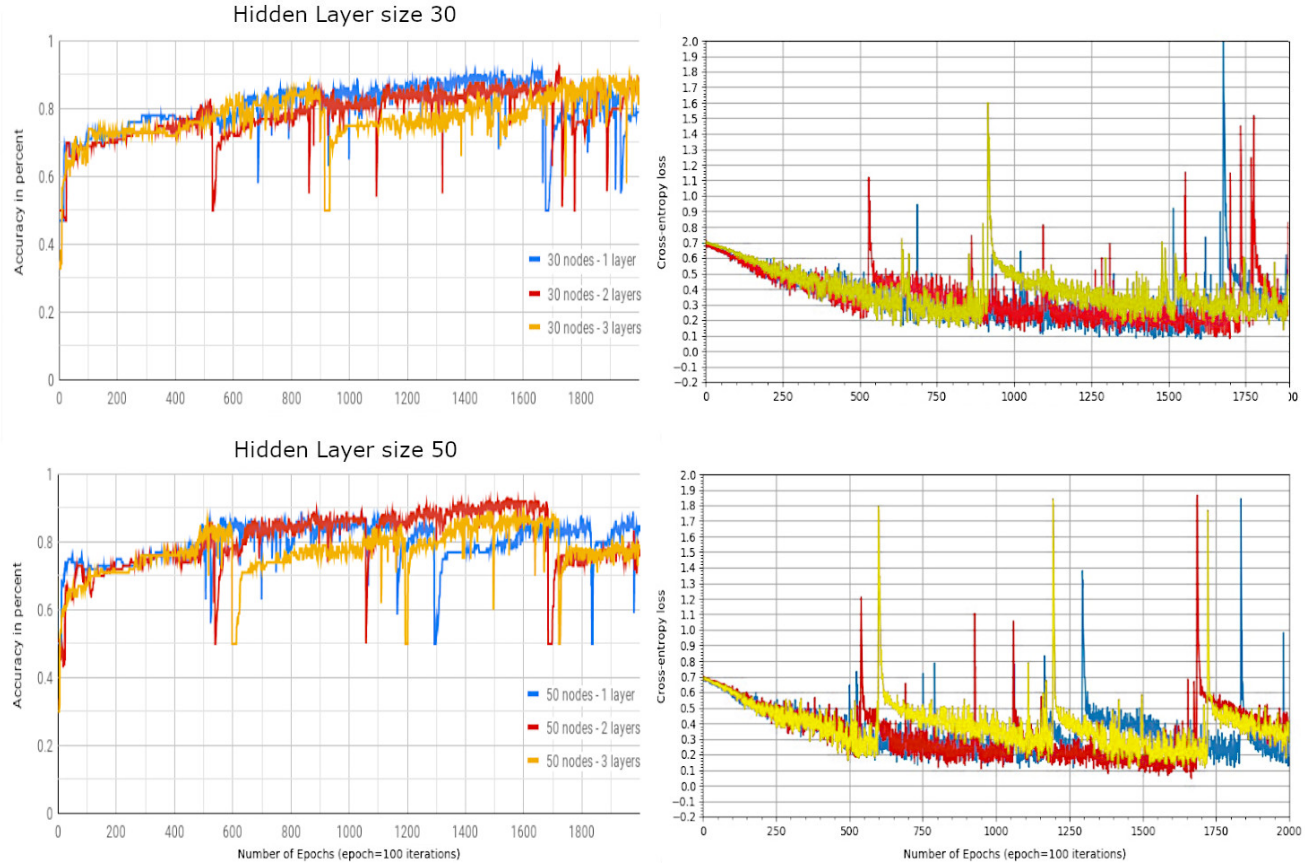

Figure 32: Accuracy/Loss of every classifier with hidden layer size 30/50 in table 10

6.3.3 Second Experiment Setup

All hyperparameters from the first experiment remain the same, with the exception of those outlined in the list below:

1. **Training set**:
   148 vectorised flats and 138 vectorised houses from the manually annotated data set - the outliers were removed

2. **Test set**:
   50 vectorised flats and 47 vectorised houses - the outliers were removed. Once again, training and test set are completely separate and do not overlap

3. **Training duration**:
   1000 epochs and 100 iterations per epoch

4. **Learning rate**:
   The learning rate was chosen to be 0.001, which is twice as high as in the previous

experiment. Learning stability is expected to be less of a problem given the cleaner training set. Increasing the learning rate allows faster training and uses less computational resources

6.3.4 Second Experiment Results

Compared with the previous experiments where a larger data set was used at the expense of training data quality, it can be seen that both losses and accuracies tend to be slighly more stable despite the higher learning rate (tab. 11 and fig. 33), albeit not nearly as much as originally hoped. Further, using the smaller but cleaner data set yielded very good results as well, with accuracies ranging from 89 % to 94 % on the test set.

LSTM Setup			**Results at the end of the training session**		
Run	Layer Size	Layers	Sensitivity	Specificity	Accuracy
1	30	1	0.94	0.85	0.90
2	30	2	0.82	0.94	0.88
3	30	3	0.92	0.96	0.94
4	50	1	0.88	1.00	0.94
5	50	2	0.92	0.94	0.93
6	50	3	0.82	0.96	0.89

Table 11: House/Flat Classifier hyperparameters and test set results

Yet, there is not enough training stability for a conclusive judgement to be made concerning the performance of different layer sizes and depths. Looking at the loss graphs from figure 33, perhaps it can be said that increasing layer size from 30 to 50 results in slightly more stable learning behaviour and overall better generalisation capabilities given the hyperparameters of this experiment. It should also be noted that even though the vector sizes are set at 75, only 50 dimension are actually trained on the text corpus. The additional 25 dimensions carry essentially no information because they are used as a padding to enable the use of one-hot vectors for the different colour-pairs. Therefore, the hidden layer size of 50 corresponds to the trained vector dimensions, which is also the reason why it was chosen as the upper benchmark. Having said that, the hidden layer size 50 may not be the best overall option for this particular task, although its performance is already reasonably good. Further experiments targeting optimisation, e.g. with layer sizes 60 or 70 and vectors with dimensions bigger than 75, may prove to be even more fruitful.

When it comes to picking different layer depths, it is also very difficult to deduce reliable patterns from these experiments. All of the graph losses for layer size 30 and 50 look very similar (fig. 33), making it close to impossible to tell if different layer sizes have any impact on performance at all. One way to get a more conclusive answer would be to run these experiments several times and look at their average performance and loss. Unfortunately, doing this within the time constraints of the dissertation was not feasible.

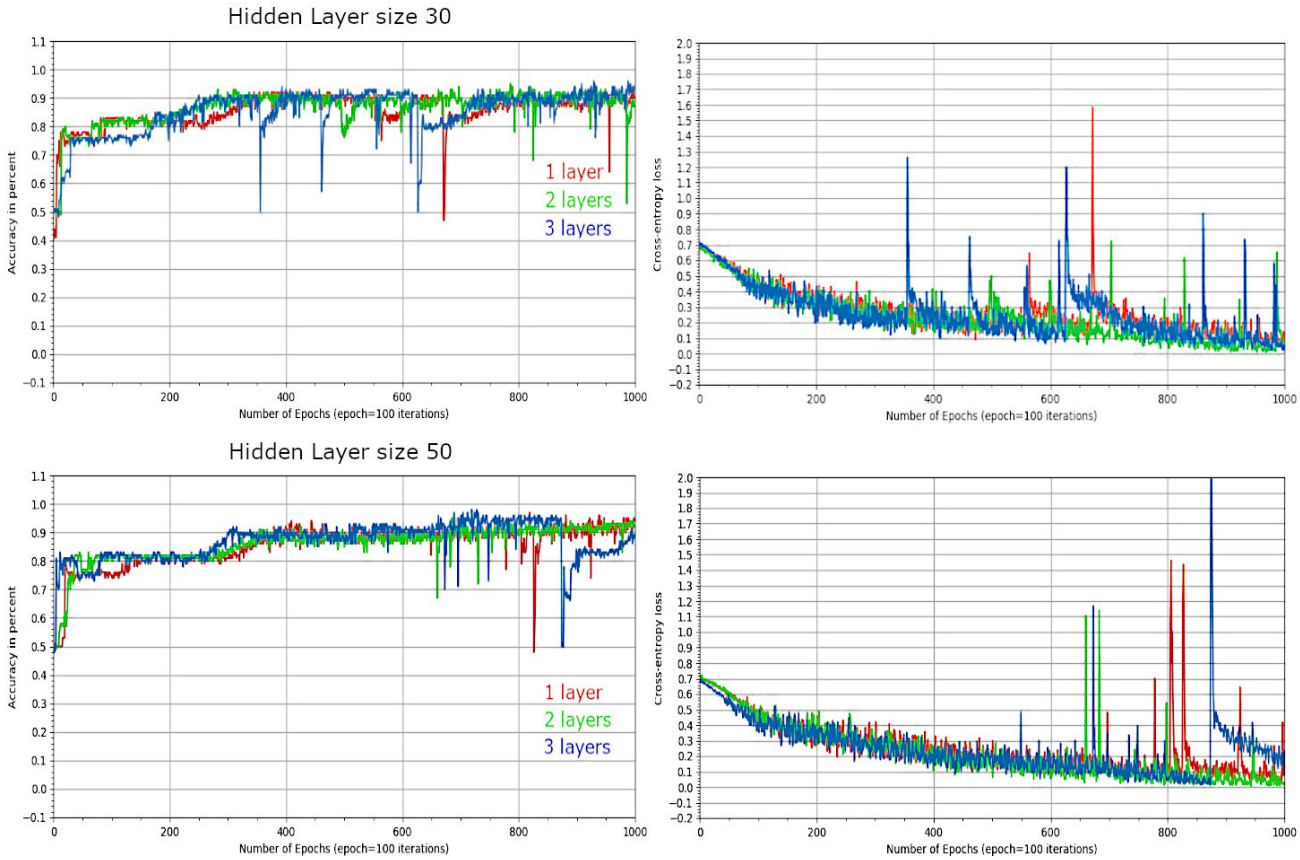

Figure 33: Accuracy of every classifier with hidden layer size 30/50 in table 11

Having ruled out small portions of wrongly labelled A-sections as the sole reason for instability, the main takeaway from these experiments is that flats and houses seem to have a wide range of heterogenous structures. Very likely, certain flats may resemble houses in a lot of ways and vice versa, making it very difficult for the classifiers to pick the right label. The latter may also be the cause for the sudden upward spikes observed in the loss graphs during training. To conclude, further investigation would be needed to provide a solid answer to these speculations.

6.4 Detecting Existing Ownership Types

Unlike with classifying houses and flats, four binary classifiers specialised on one of the ownerships are required to detect whether or not their given type of ownership in fact exists. With these classifiers all being binary, a positive label indicates that a specific type of ownership exists within the given A-section and is also accompanied by one or several colour-pairs. Conversely, a negative label indicates that said ownership does not have a corresponding colour-pair within the text. In the case of exclusive strata, this always means that there exists a verbal description of it without a colour-pair. For all other types of ownership, a

negative label simply means that the ownership does not exist within the text. When it comes to training separate classifiers for each type of ownership, however, the distinction between *verbalised* and none does not require any modifications to the binary classifier.

6.4.1 Experiment Setup

The additional challenge of identifying existing types of ownership in flatted A-sections compared to separating flats and houses lies within the highly imbalanced distribution of tags (tab. 12). This has two implications. Firstly, drawing training samples randomly cannot be done anymore, because the classifiers would be exposed to far more negative than positive samples. This could severely impact the learning process. To mitigate this problem, a simple algorithm for drawing samples was proposed (alg. 2), ensuring a balanced exposure to positive and negative samples during training.

	Training Samples		**Testing Samples**	
Ownership Type	Positive	Negative	Positive	Negative
Exclusive Strata	34	66	19	81
Exclusive Solum	16	84	18	82
Common Strata	10	90	2	98
Common Solum	34	66	29	71

Table 12: Number of training and test samples for each type of ownership

Algorithm 2: Sampling method for training data

```
positive_samples = 0;
negative_samples = 0;
while training do
    if positive_samples <= negative_samples then
        randomly draw a positive sample from the training set;
        positive_samples++;
    else
        randomly draw a negative sample from the training set;
        negative_samples++;
    end
end
```

Secondly, accuracy as an overall measure of performance is not going to be sufficient. For example, in the extreme case of 98 testing samples with no common strata and 2 positive samples with a colour-pair for common strata, one could push the accuracy to 98 % on this test set by simply classifying everything as negative (tab. 12). Thus, the classifiers' performance should be measured by sensitivity and specificity instead to account for the highly

imbalanced sample distribution.

Thirdly, there is the general lack of training data. With flats and houses, both annotated sets combined provided 300 training samples and 100 test samples. This is not a lot of data either, but still sufficient to get reasonable results. On the other hand, separating merely 200 flats in half into a training and test set is actually problematic, since there are only 100 A-sections to train with. As a result, it was proposed to artificially increase the number of training samples by a factor of 10 using the previously described algorithm (alg. 1) for duplicating and randomly generating colour-pairs. To observe the effects of this upsampling method, the classifiers were trained on both the original training set of 100 samples and on the upsampled training set with 1000 samples and scrambled colour-pairs.

Another possible optimisation would be to aim for a more effective distribution of samples in the training and test set. For example, exclusive strata has 34 positive and 66 negative samples in the training set, while its test set has 19 positive samples and 81 negative samples. This is essentially the default distribution of samples if one simply declares the first 100 samples out of the 200 flatted A-sections to be the training set and the latter 100 samples to be the test set. One possibility would be to ensure that the ratio of positive to negative samples is roughly equal in both the test and training set, e.g. by deliberately selecting 26 positive samples for the training and 27 positive samples for the test set in the case of exclusive strata.

Altogether, this idea was not implemented because from an implementation standpoint it is very time intensive to get this mechanism to work. On the one hand, the concept in itself may sound simple, yet it often turned out that integrating seemingly simple concepts into the existing code architecture proved to be very tedious. Also, while the given distribution of labels is not perfect, it turned out to be good enough from a performance point of view when combined with the upsampling technique used on the training set. Therefore, this would have been a potential optimisation which could have been implemented as the next logical step to improve the initial results. Unfortunately, this never materialised due to time constraints.

Lastly, the hyperparameters of each of the four classifiers were set in a similar fashion compared to the houses-flats classifier. The learning rate was increased to *0.002* to speed up learning, while the number of epochs was set to *700*. Upon receiving the results, the hyperparameters would be adjusted if deemed necessary, e.g. in a situation where 700 epochs were not sufficient to fully train the classifier or if the learning rate lead to unstable learning behaviour.

LSTM Hyperparameters for all four classifiers	
Number of epochs	700
Iterations per epoch	100
Learning rate	0.002
Input layer size	75
Hidden layer size	30
Number of hidden layers	1
Output layer size	2

Table 13: LSTM Training Hyperparameters

6.4.2 Results

Generally speaking, it can be said that even with limited training data at hand most classifiers delivered a solid baseline performance which can be improved upon with more annotated data and further fine-tuning of the hyperparameters.

	No Upsampling		**With Upsampling**		**Improvement**	
Ownership Type	Sens.	Spec.	Sens.	Spec.	Sens.	Spec.
Exclusive Strata	0.79	0.79	0.79	0.83	+ 0.00	+ 0.04
Exclusive Solum	0.67	0.90	0.79	0.94	+ 0.12	+ 0.04
Common Strata	1.00	0.91	1.00	0.99	+ 0.00	+ 0.08
Common Solum	0.56	0.89	0.69	0.92	+ 0.13	+ 0.03

Table 14: Results for the four ownership classifiers with and with no upsampling

In addition, the results in table 14 indicate that upsampling the data and randomising the colour-pairs did result in a significant improvement in performance. Here, it is important to note that both sensitivity and specificity were tested at the end of the training sessions. Also, the number of epochs was reduced from 700 to 400 when using the upsampled training data, since it became evident that 700 epochs were too much to begin with when training the classifier on the original 100 A-sections. This can be seen when looking at the two loss graphs for the exclusive strata classifier in figure 34. Essentially, the loss graph reaches zero at around epoch 200 for both versions of the classifier, albeit bouncing back for the classifier using upsampled data shortly after. In any case, the main point here is that reducing the number of epochs did not visibly compromise the test results. However, reducing the number of epochs was very helpful in reducing the runtime required for training the second version of the same classifier.

Despite the improvements achieved with upsampling, this technique also has its downsides. In figure 34, it very clear that the overall training process is more unstable compared to training on the unaltered one hundred A-sections. In order to achieve the best possible results, setting a fixed number of epochs and saving the state of the classifier at the end may

not be sufficient on its own. For instance, when trying to train the classifier in a deployment scenario it may be advisable to save the training state periodically before proceeding. At the end, the state with the best performance can be picked for deployment. This approach was not used for any of the experiments, because it would slow down training considerably in a scenario where the main objective is testing different architectures, i.e. hyperparameters and sampling algorithms.

Lastly, the classifiers for exclusive solum, common strata and common solum exhibited similar behaviour during training, which is the reason why their graphs are not explicitly shown in this section. It should be noted, however, that sensitivity for common solum is comparatively low at 0.56 % when trained with no upsampling and 0.69 % when upsampling is applied. This is a problem specific to common solum, with the main cause being the distribution of colour-pairs (fig. 30) and the fact that common solum is often denoted by multiple colour-pairs.

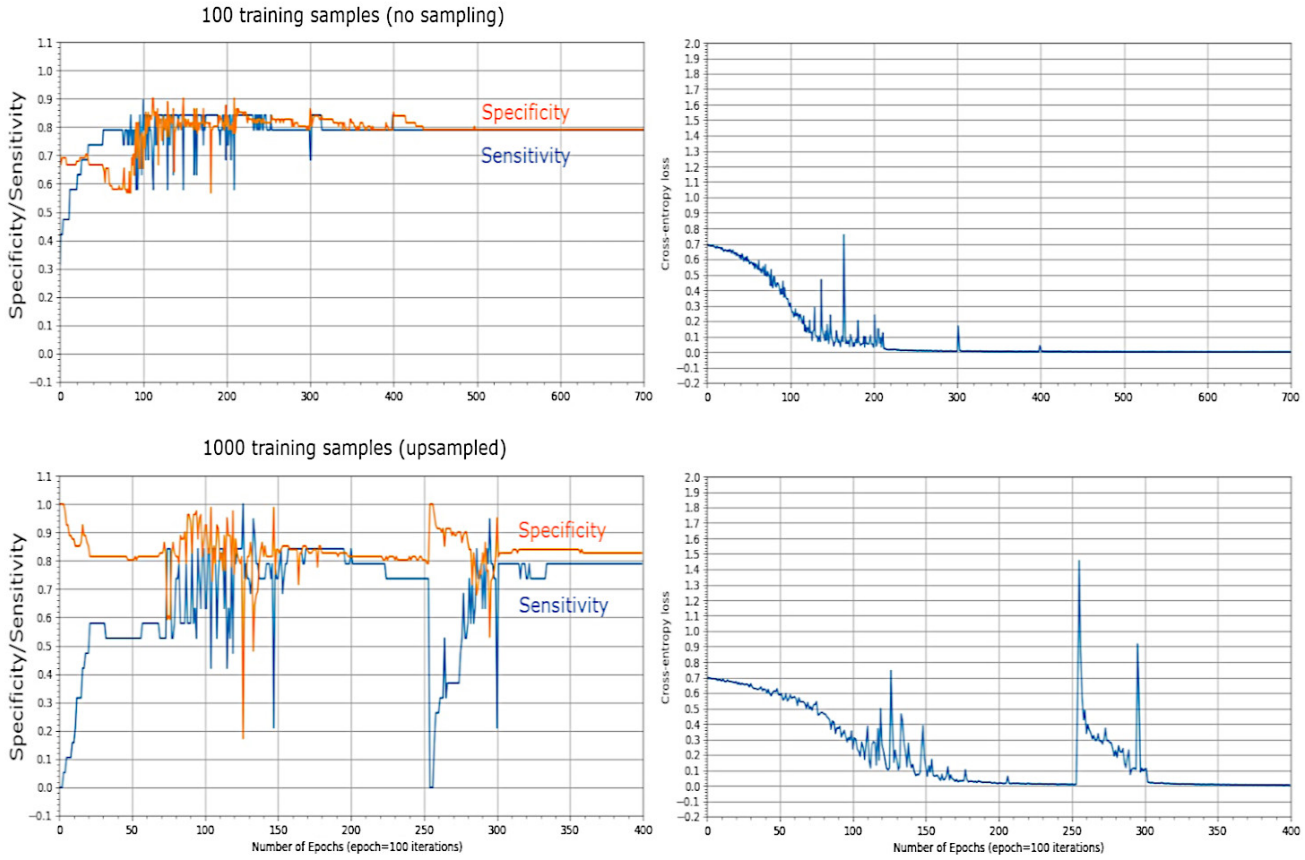

Figure 34: Sensitivity/Specificity over time for exclusive strata classifier

6.5 Mapping Colour-Pairs to Ownership

The last classification step involves iterating through every colour-pair in a given A-section and tagging it with a *yes/no* label depending on the classifier's type of ownership. Once again, four classifiers in total are needed to map every ownership type to one or more colour-pairs in said A-section.

6.5.1 Experiment Setup

Unlike with the previous experiments on separating properties and confirming the existence of ownership types, preparing the training and test sets turned out to be rather complicated in this case. For instance, the individual steps taken to construct the training and test sets for exclusive strata were:

1. Take the 200 flatted A-sections and split them evenly into a provisional training and test set
2. In both sets, remove all A-sections which do not contain any colour-pairs corresponding to exclusive strata. This means that every remaining A-section is going to have at least one colour-pair corresponding to exclusive strata. This means that one A-section does not always equate to one training sample, it can be more
3. To get the total number of positive samples, count the number of colour-pairs corresponding to exclusive strata within every document (table 15)
4. To get the total number of negative samples, i.e. those colour-pairs that do not correspond to exclusive ownership, count all of the remaining colour-pairs in every remaining A-section (table 15)

During training, the simple algorithm that was drawing the training samples for the ownership detection classifiers (alg. 2) was used here as well. The only difference in this case is that positive and negative samples are not the A-sections themselves, but the different colour-pairs within the A-sections. Therefore, selecting a random sample involves randomly picking an A-section and then randomly picking one of the positive or negative colour-pairs within that A-section.

Once again, the tradeoff for quick implementation is the acceptance of an unbalanced distribution of positive and negative labels in both the training and test set. In this case, the argument for simplicity is in fact even more prevalent due to the additional level of complexity of handling several training samples per A-section. Another implication of this problem is that a fully implemented balancing mechanism for the ownership detection classifiers is not even transferable to this type of problem, making implementation even more difficult. In a potential deployment scenario, on the other hand, this problem needs to be taken very

seriously.

	Training Set		Test Set	
Ownership Type	**Postives**	**Negatives**	**Positives**	**Negatives**
Exclusive Strata	34	101	20	39
Exclusive Solum	21	57	19	35
Common Strata	10	38	3	9
Common Solum	85	59	54	27

Table 15: Actual number of positive and negative colour-pairs

Similar to those classifiers trying to identify existing types of ownership, there is a general lack of training data. Once again, to get more out of the existing data, the upsampling algorithm from the previous classifiers (alg. 2) was applied to the training data in such a way that it resulted in 1,000 A-sections in the training set and 100 A-sections in the test set (tab. 16). With at least one colour-pair per A-section per ownership, this actually resulted in the training and test sets being slightly bigger in reality (tab. 17).

	A-sections		Upsampling factor		A-sections	
Ownership Type	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Exclusive Strata	34	19	30	6	1,020	114
Exclusive Solum	19	18	63	6	1,008	108
Common Strata	10	2	100	50	1,000	100
Common Solum	34	29	30	4	1,020	116

Table 16: Upsampling the remaining A-sections to 1000 and 100

	Training Set			Test Set		
Ownership Type	**Postives**	**Negatives**	**Total**	**Positives**	**Negatives**	**Total**
Exclusive Strata	1,020	3,030	4,050	120	234	354
Exclusive Solum	1,323	3,591	4,914	114	210	324
Common Strata	1,000	3,800	4,800	150	450	600
Common Solum	2,550	1,770	4,320	216	108	324

Table 17: Increase in colour-pairs as a result of upsampling (see table 16)

Realistically speaking, the training sets contain roughly 4,000 and 5,000 samples, while the training sets are set between size 300 and 600 (tab. 17). Again, this is good enough for the time being as a quick way to artificially increase the number of training samples.

When it came to selecting the training hyperparameters, empirical evidence, i.e. trial and error, showed that about 500 epochs at a learning rate of 0.001 were sufficient to reduce the classifiers' loss to practically zero. To speed up learning, layer size 30 was chosen instead of 50.

LSTM Hyperparameters for all four classifiers	
Number of epochs	500
Iterations per epoch	100
Learning rate	0.001
Input layer size	75
Hidden layer size	30
Number of hidden layers	1
Output layer size	2

Table 18: LSTM Training Hyperparameters

6.5.2 Experiment Results

Other than for common solum, all classifiers were reasonably good at tagging colour-pairs as either belonging to their type of ownership or not (tab. 19). Compared to the classifiers trying to detect the existence of ownership, the learning process was also far more stable. In figure 35, 36 and 37, the loss graph always reaches zero reliably almost without any sudden upward spikes.

Ownership Type	**Sensitivity**	**Specificity**
Exclusive Strata	0.92	0.98
Exclusive Solum	0.90	0.80
Common Strata	1.00	1.00
Common Solum	0.74	0.76

Table 19: Classifiers' performance at the end of the training session

It may seem surprising that the classifier tasked with tagging colour-pairs denoting common strata achieves a sensitivity and specificity of 1.0. The problem with these figures is essentially that there are only ten original colour-pairs in the training and three in the test set. Hence, these numbers should be taken as a mere indication rather than a conclusive statement concerning the performance of a real-world classifier.

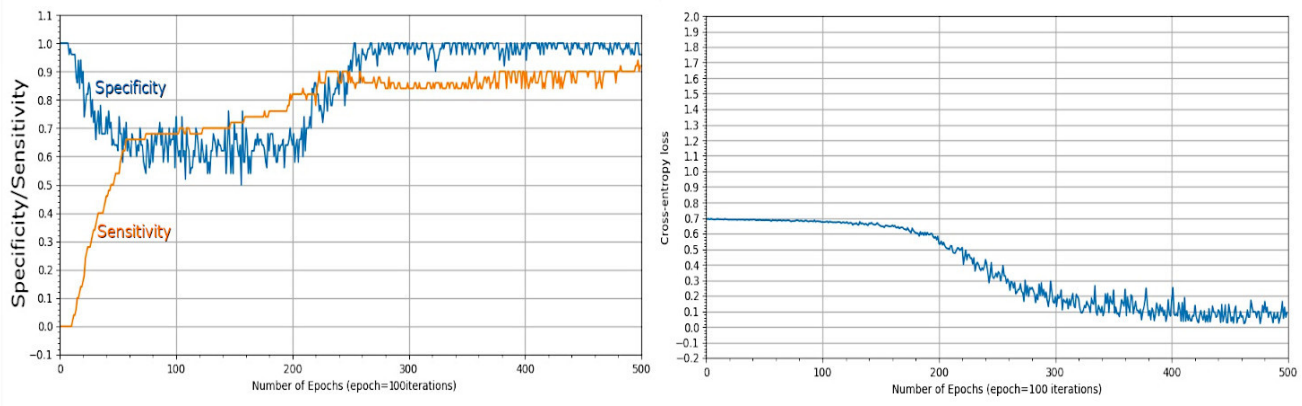

Figure 35: Exclusive Strata

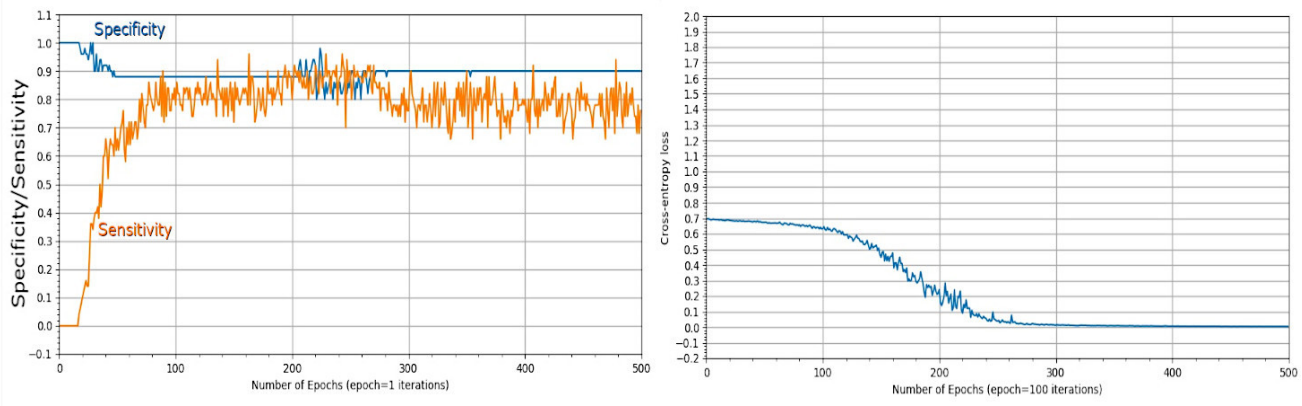

Figure 36: Exclusive Solum

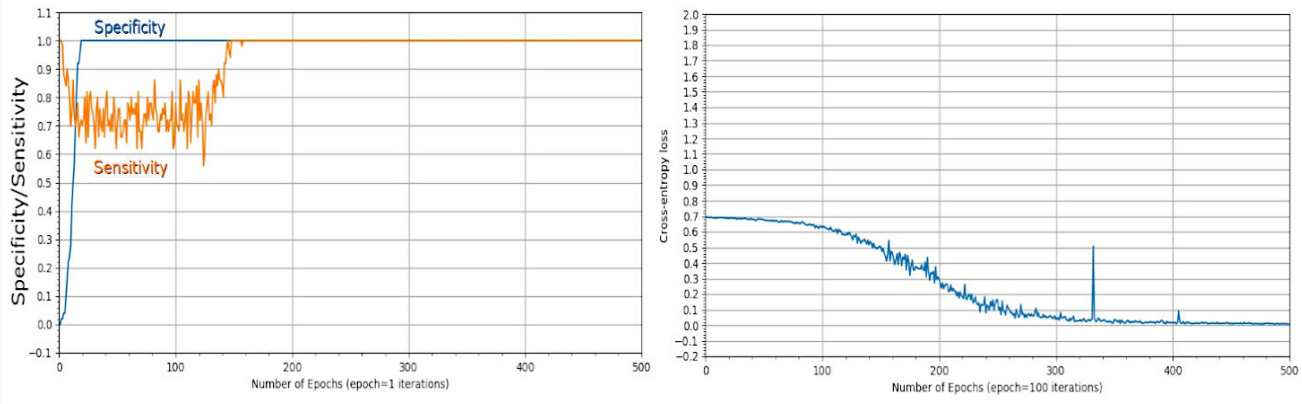

Figure 37: Common Strata

The one exception in this suite of classifiers is the one concerning common solum. Not only are sensitivity and specificity down at 0.74 % and 0.76 % (tab. 19), the learning process

itself is much more unstable compared to the other classifiers (fig. 37). The most likely reason for the relatively poor performance lies within the colour-pair distributions observed with common solum. More specifically, common solum is far more likely to be described by multiple colour-pairs at a time compared to other types of ownership (tab. 20).

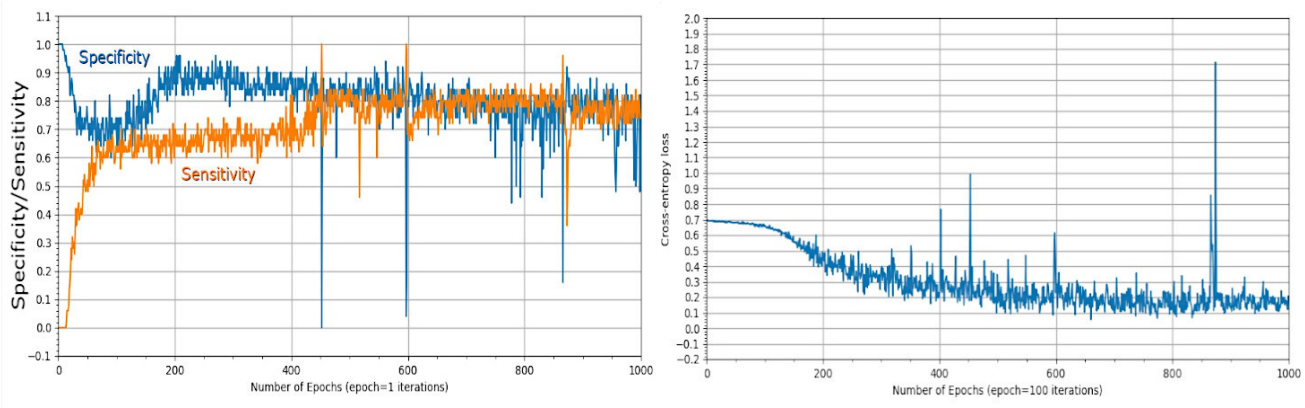

Figure 38: Common Solum

Ownership Type	**Exclusive Strata**	**Exclusive Solum**	**Common Strata**	**Common Solum**
1 Colour-Pair	51	28	11	22
2 Colour-Pairs	1	6	1	19
3 Colour-Pairs				10
4 Colour-Pairs				11
5 Colour-Pairs				1

Table 20: Number of colour-pairs belonging to one type of ownership per A-section

7 Conclusion

Finally, this section will look into things that could have been done differently and will give an outlook concerning the potential next steps that RoS could undertake to elevate this model into a production environment.

7.1 Potential Improvements

Given the benefit of hindsight, a couple of things were identified which could have been done better. To be more specific, the training process could have been more rigorous by using cross-validation in the case of separating houses and flats, the colour-pair tokens could have been designed more carefully when it came to identifying existing ownership and some of the coding process could have been more structured, e.g. by doing more thorough testing.

7.1.1 Questionable use of colour-pair tokens for identifying existing ownership

The classifiers concerned with identifying existing ownership were trained on an upsampled data set with randomly generated colour-pairs. Later on, it was realised that using this method was arguably unnecessary, since there is no need at this stage to map any colour-pairs to their respective ownership. Instead, the colour-pairs could have been replaced by one specific vector signifying that a colour-pair exist in this place, e.g. a complete zero vector. The latter would have also made the additional padding of the other vectors obsolete, reducing vector sizes from 75 to 50 as a result. With that, training speed and stability might have been improved.

7.1.2 Using a naive sampling method

When separating the A-sections into a training and test set, the A-section were split in half without looking at the distribution of colour-pairs or ownership patterns. One the one hand, this approach was very fast and allowed for quick implementation. On the other hand, this resulted in an unbalanced and random distribution of samples across the training and test set. If more time was available, more sophisticated sampling methods could have been put into place, increasing the quality of the classifiers as a result.

7.1.3 Training the classifiers

It has already been mentioned how the lack of data impacted the training process. Yet, there is a method that could have potentially been used to gain some deeper insights concerning the classifier's performance, which happens to be cross-validation. This would not have

been feasible with the classifiers detecting and mapping the colour-pairs to ownership due to the lack of data, but it certainly would have worked with houses and flats. With 500 flats and houses making up a training set of size 1,000, one could have separated this set into 900 training samples and 100 test samples. Subsequently, ten different classifiers could have been trained on this data set using a different portion as the test set every time (fig. 39).

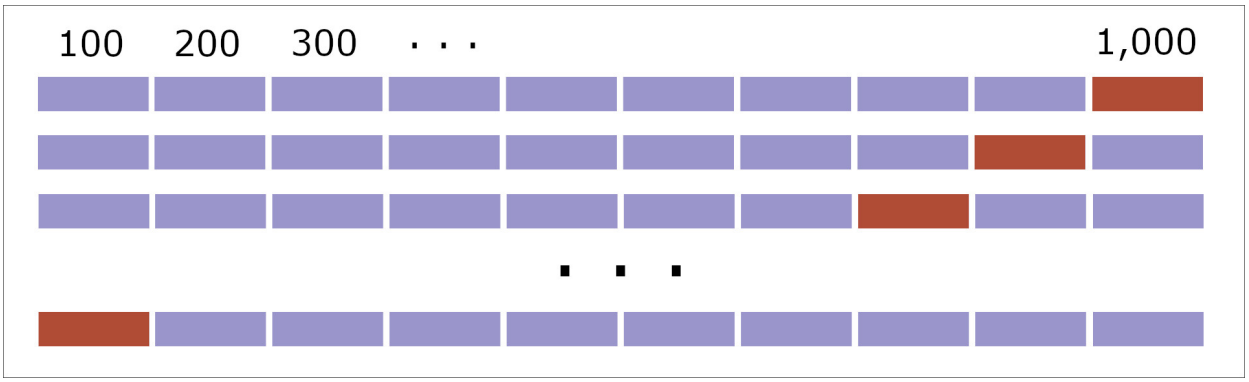

Figure 39: Cross-validation visualised - the blocks tinted red represent the test set

Overall, this method could have had the potential to uncover patterns concerning choice of different layers sizes and depths. For instance, there would have been no bias towards one specific test set. Further, having ten different results would have enabled the averaging of graphs and the creation of error-bar charts showing the differences in accuracy, sensitivity and specificity for every training session. This would have resulted in a potential recommendation to RoS as to what type of architecture may be appropriate going into the future.

At the same time, cross-validation is computationally expensive. In this particular example, the amount of runs required would be multiplied by a factor of ten. With two layer sizes and three different layer depths to test, using this method would have required 60 separate runs. Given that this would have only been done on the house-flat classifiers, however, it is safe to say that this could have been attempted. Moreover, instead of ten different runs per architecture a smaller number of runs could have been considered as well, e.g. seven or five.

7.1.4 Availability of data

Another issue was the delay in getting the RoS data due to the time spent on designing, signing and approving the non-disclosure agreement. At the end, RoS agreed to transfer the data onto Cirrus at risk without the legal process having finished. Because of the delay, the data was not available until the end of June, which happened to be the half-way mark for the dissertation. Furthermore, an alternative data set containing movie reviews had to be used to implement and test the model. If the data had been available earlier, some time could have been saved by working on the RoS data set directly.

7.1.5 Problems with bugs during the coding process

When the model was implemented and tested on the movie review data set, the results described in various papers could not be reproduced. After a lot of test runs, it was eventually discovered that the model suffered from a bug preventing it from taking the entire length of the movie reviews into consideration. Another contributing factor was the use of code for the Word2vec and LSTM model written by other people. This particular problem could have been potentially avoided by using more rigorous software engineering practices, e.g. more testing, instead of opting for quick implementation.

7.2 Future Work

The project's main intention was to provide a proof of concept to RoS that machine learning based techniques can be a part of the solution to their problems. Naturally, the existing work can be improved upon going forward in to the future. In this section, a couple of recommendations will be given as to what one could do to push the performance of the existing classification model.

7.2.1 Data Annotation

200 annotated flats proved to be sufficient to deliver a lot of valuable insights and proved that a machine learning based approach can be a viable solution for the colour-pair mapping problem. At the same time, there are also a couple of limitations which can only be overcome by annotating a far greater data set. Firstly, a lot of instability was observed with some of the classifiers during training, a problem sometimes further exacerbated by the upsampling method used to artificially increase the amount of training data. Furthermore, the lack of data resulted in sparsity for certain types of ownership, most notably common strata. The latter was only observed 12 times within the entirety of all 200 annotated A-sections. Secondly, the classifiers' generalisation capabilities could not be put to the test, for example by using cross validation due to lack of data, making a direct comparison of different network architectures very difficult.

It should also be noted that annotating the A-sections without prior domain knowledge is difficult, but possible provided there is sufficient training by domain experts. The downsides are the initially slow annotation process during training and an inevitable margin of error due to said lack of domain knowledge. On top of that, by looking at a wide variety of samples a lot of previously unknown concepts and structures were discovered, leading to a gradual adaptation of the existing model. To give an example, the concept of strata and solum for flats and houses was not known before the annotation process. Then, the existence of colour-pairs denoting rights of access or the very common citation of cross-references to other documents further increased the complexity of the annotation process. While the latter

aspects were not directly relevant to classifying ownership patterns, this does not mean that classifiers specialising on rights of access or cross-references are not possible. Hence, some of the non-ownership related information within the A-sections was included into the tagging process as well, especially since it did not require a lot of additional effort to do so.

To conclude, if RoS should decide to go down this path, spending a considerable amount of time and resources on creating an annotated data set of high quality and quantity is absolutely crucial. This can be done by either the domain expert or by a specifically trained person or team. To speed this process up, the already existing classifiers can be used to automatically annotate a range of A-sections, whereas the human annotators can look through the annotated samples and correct potential errors. Subsequently, the classifiers can be retrained once a new batch of samples had been correctly annotated. With this approach, one can also observe the effects of gradually adding more training data on the classifiers, while at the same time gradually refining the hyperparameters.

7.2.2 Optimisation

One of the important things to take into consideration is that by using a cascade of classifiers, small amounts of classification errors are going to accumulate over time. This problem is illustrated in table 21, albeit in a slightly oversimplified fashion:

Classification Step	A-Sections	Classified Correctly	Classified Wrongly
Separating Houses and Flats	100	90	10
Identifying ownership	100	81	19
Colour-pairs to ownership	100	≈ 73	≈ 27

Table 21: Propagation of misclassified samples over time assuming an accuracy of 90%

Assuming an accuracy of 90% for every single classifier and a hundred unknown A-Sections consisting of only flats and houses, 90 A-sections will be labelled correctly as either flats or houses and another 10 will be labelled incorrectly. With that in mind, it is save to say that identifying existing types of ownerships for these 10 samples is going to go wrong, since the misclassified flats will not be tested on exclusive and common strata. Likewise, the misclassified houses will be sent through the classifiers for exclusive and common strata, even though the concept of strata does not apply to houses. In addition, only 90% of the remaining 90 A-sections will be classified correctly, resulting in a total of 81 correctly classified samples. This figure is going to decrease even further when it comes to mapping colour-pairs to ownership.

Thus, when it comes to optimising the general performance of the entire model, a top-down approach is the most reasonable way forward. By optimising the classifiers at the top first, i.e. the upper classification nodes in the classification tree, the propagation of error down to

the lower classification nodes can be minimised. In practice, this process has to start with the property type classifier first, then continue with the ownership identification classification suite. Once the performance is good enough for RoS's needs, the next and ultimate step would consider optimising the process of mapping colour-pairs to ownership.

Another point to take into consideration is that not every node in the classification tree has to be an LSTM-based classifier. Originally, flats and houses were separated using a heuristic method involving keywords and the number of floors. Altogether, the performance of this heuristic mechanism is comparable to the levels of accuracy observed when using the LSTM model. This leaves RoS with the choice to either continue refining the heuristic approach or to continue with the LSTM classification model. Ultimately, a decision can be made once a decent amount of annotated A-sections is put into place and a more realistic comparison can be implemented. This is not to say, however, that every LSTM classifier necessarily has an equivalent heuristic method of the same quality. Separating flats and houses according to keywords may be relatively straightforward, interpreting context around colour-pairs certainly is not.

A Code Structure

Given the relatively short period of time and an emphasis on testing a wide variety of ideas on how to improve the classifier, quick implementation cycles constitutedthe most important requirement concerning programming language and framework choice. In addition, training the models in question by means of hardware acceleration such as GPUs without resorting to writing hardware specific code was of equal importance.

As a result, the LSTM classifier and Word2Vec implementations are written in **Python** 3.7.3 and make extensive use of the **PyTorch** framework.

PyTorch is a library built specifically for fast implementation of machine learning models within the realm of research. It achieves this by automating a wide variety of steps that would otherwise require a detailed low-level implementation. A selection of those steps includes automatic handling of tensor meta data by the Variable class, automatic backward propagation by invoking a simple *backward* function supported by the said class, optimised memory management and execution on GPU's as well as a wide range of pre-programmed loss functions (Paszke et al., 2017).

In addition to using PyTorch, the basic implementations of both the Word2Vec and the LSTM models were borrowed from two GitHub users and incorporated into the implementation. The specific details of using GitHub are detailed in the section below.

A.0.1 GitHub Repositories

Repository 1 contains the main source code of the classifier, whereas repositories 2 and 3 contain code borrowed from the GitHub users emadRad (emadRad, 2018) and Andras7 (Andras7, 2018):

1. Main GitHub repository:
   https://github.com/AlexRiepenhausen/MastersDissertation
2. Implementation of LSTM and GRU for classifying the MNIST dataset:
   https://github.com/emadRad/lstm-gru-pytorch/blob/master/lstm_gru.ipynb
3. Simple and fast Word2vec implementation with Negative Sampling + Sub-sampling:
   https://github.com/Andras7/word2vec-pytorch
   This particular implementation makes use of Skip-grams, Batch updates and also offers GPU support, a crucial feature for accelerating training

It is important to note that different types of classifiers are positioned on different branches. More specifically, the classifier distinguishing houses from flats is located on a different branch

compared to the classifier detecting existing types of ownership. This separation allowed for quick implementation and potential rollback if something did not work out as planned. Further, this prevented the code from becoming too complex, since the different classifiers had different procedures on handling the training and tests sets. Implementing these in one branch would have resulted in too many different options to keep track of. If RoS decides to turn this concept into a fully functional application, engineering a maintainable piece of software would be the way forward. However, the latter was not the main concern of this project to begin with. With that being said, a list of relevant branches is given below:

1. Generic Master branch:
   https://github.com/AlexRiepenhausen/MastersDissertation/tree/master
2. Separating houses and flats:
   https://github.com/AlexRiepenhausen/MastersDissertation/tree/houseflat
3. Identifying existing types of ownership:
   https://github.com/AlexRiepenhausen/MastersDissertation/tree/verbal_non_verbal
4. Colour-pairs to ownership:
   https://github.com/AlexRiepenhausen/MastersDissertation/tree/c2o_preselected
5. Sentiment classification on IMDB movie review data set:
   https://github.com/AlexRiepenhausen/MastersDissertation/tree/imdb

Looking at this list, it may seem surprising that number 5 talks about doing sentiment classification on a data set containing movie reviews. As a matter of fact, there were delays in getting access to the RoS data, meaning that the model had to be tested and developed using a publicly available data set.

A.0.2 High-Level Code Structure Overview

Down below are two lists of classes and methods containing a high-level description of both the word2vec and lstm implementation. As mentioned previously, these two implementations were taken from the web and adjusted to meet the needs of this project. Other parts of the code, e.g. helper classes and methods will not be mentioned here to avoid unnecessary clutter.

A.0.3 Word2Vec Implementation

The Word2Vec package consists of the following components:

1. **dataReaderDoc.py**
   (a) Class DataReader
      i. init - reads training files from either .txt or .ndjson and constructs auxiliary data structures to support *trainer.py* when training the model
      ii. readWords - constructs a word histogram and replaces unique words with integer tokens to save memory space
      iii. initTableDiscards - identifies words to be discarded during training process (optional)
      iv. initTableNegatives - puts the concept of negative sampling into practice
   (b) Class Word2vecDataset
      i. getitem - PyTorch specific method which fetches the current training sample for *trainer.py*
2. **dictionaryMerger.py**
   (a) init - takes file paths of two distinct dictionaries
   (b) replaceVectors - replaces specific vectors in first, e.g. colour-pairs, with vectors from second dictionary
   (c) writeVectors - writes new dictionary two file
3. **trainer.py**
   (a) init - sets training data file paths, the number of training epochs and the Skip-gram model's hyperparameters
   (b) train - runs the current word with its surrounding words through the Skip-gram model and propagates error back for a set number of epochs. At the end, *def save_ embedding* from word2vec.py is invoked
4. **word2vec.py**
   (a) init - initialises Skip-gram model and sets hyperparameters
   (b) forward - forward the one-hot vector through the Skip-gram architecture
   (c) save_embedding - extracts vectors from hidden layer and writes them to a dictionary file

A.0.4 LSTM Implementation

The LSTM package consists of the following components:

1. **dataReaderDoc.py**
   This is classifier specific. The exact data sampling methods used during training will be discussed in a later section. Mainly, this class decides on a sample, reads its vectorised version and passes it to *trainer.py*

2. **file2VecConverter.py**

   (a) init - specifies file paths for both the vector dictionary and the text files to be converted to their vector representations

   (b) convertDocuments - takes the vector dictionary and converts specified text files into their vector representation; these files are written back to storage at the end

   (c) unknownWordReplacement - if an unknown words is encountered, i.e. no vector exists for this word, the word is either replaced by a token or skipped altogether

3. **lstm.py**

   (a) Class LSTMModel

      i. init - initialises LSTM model provided by PyTorch and sets hyperparameters

      ii. forward - takes vectors from current A-section and puts them through the LSTM layer until end of file is reached. In other words, loops over *LSTMCell.forward*

   (b) Class LSTMCell

      i. init - sets size of input layer (vectors) and hidden layers, i.e. the input, output and forget gate as well as the cell itself

      ii. forward - takes one vector and forwards it through the LSTM architecture. Is called repeatedly in the LSTMModel's forward method

4. **trainer.py**

   (a) init - sets training data file paths, the number of training epochs and iterations and the LSTM's hyperparameters

   (b) train - runs the vectorised A-sections through the LSTM model and propagates error back for a set number of iterations and epochs. Note that one epoch can contain multiple iterations. At the end, graphs denoting the loss over time and the classifier's accuracy/sensitivity/specificity are produced

B Submission Details

This section provides an overview of the entities submitted aside from the main report.

B.1 Code

Three different code branches were submitted along with the report, namely, the code branch concerning the classifier for separating houses and flats, the code for detecting existing types of ownership as well as the code for mapping colour-pairs to their respective ownership types. Given that the code used on the IMDB data set was used for testing and development purposes only, this particular branch will no be included in the final submission.

B.2 Data

As stated by the non-disclosure agreement signed by the student, the data concerning A-sections must not leave Edinburgh University's supercomputing facilities (Cirrus). For this reason, no RoS data cannot be part of the submission process. As for the imdb data set, it is publicly available at https://ai.stanford.edu/~amaas/data/sentiment/ (Maas et al., 2011b).

References

Andras7 (2018). *Extremely simple and fast word2vec implementation with Negative Sampling + Sub-sampling Source Code (Version 1.0)*. URL: https://github.com/emadRad/lstm-gru-pytorch/blob/master/lstm_gru.ipynb (visited on 05/29/2019).

Baroni, M., G. Dinu, and G. Kruszewski (2014). "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors." In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Vol. 1. Baltimore, Maryland, USA: Association for Computational Linguistics, pp. 238–247. URL: https://www.aclweb.org/anthology/P14-1023 (visited on 06/27/2019).

Beck, Anthony (2019). *A-sections: structural (and linguistic) breakdown*. Registers of Scotland. Meadowbank House, 153 London Rd, Edinburgh EH8 7AU, UK.

emadRad (2018). *Implementation of LSTM and GRU Source Code (Version 1.0)*. URL: https://github.com/emadRad/lstm-gru-pytorch/blob/master/lstm_gru.ipynb (visited on 05/29/2019).

Graves, A. et al. (2008). "A novel connectionist system for unconstrained handwriting recognition." In: *IEEE transactions on pattern analysis and machine intelligence, 31(5)*, pp. 855–868.

Grosse, R. (2017). *Lecture 4: Training a Classifier*. University Lecture. University of Toronto. URL: http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/readings/L04%5C%20Training%5C%20a%5C%20Classifier.pdf (visited on 08/11/2019).

Hochreiter, S. and J. Schmidhuber (1997). "Long Short-Term Memory". In: *Neural computation 9(8)*, pp. 1735–1780.

Johnson, R. and T. Zhang (2016). *Exploiting Similarities among Languages for Machine Translation*. URL: https://arxiv.org/abs/1602.02373 (visited on 08/08/2019).

Maas, Andrew L. et al. (2011a). *Large Movie Review Dataset*. URL: https://ai.stanford.edu/~amaas/data/sentiment/ (visited on 06/27/2019).

— (2011b). "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, pp. 142–150. URL: http://www.aclweb.org/anthology/P11-1015 (visited on 06/27/2019).

McCormick, C. (2016). *Word2Vec Tutorial - The Skip-Gram Model*. URL: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/ (visited on 06/20/2019).

Mikolov, T., I. Sutskever, et al. (2013). "Distributed representations of words and phrases and their compositionality." In: *Advances in Neural Information Processing Systems 26, NIPS 2013*. Published by Neural Information Processing Systems Foundation Inc., pp. 3111–3119. URL: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf (visited on 06/10/2019).

Mikolov, T., Q. V. Le, and I. Sutskever (2013). *Exploiting Similarities among Languages for Machine Translation*. URL: https://arxiv.org/pdf/1309.4168.pdf (visited on 06/05/2019).

Paszke, Adam et al. (2017). "Automatic differentiation in PyTorch". In: *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA*. Pp. 1–4. URL: https://openreview.net/pdf?id=BJJsrmfCZ (visited on 06/17/2019).

Rong, X. (2014). *Word2vec parameter learning explained.* Cornell University, arXiv preprint. URL: https://arxiv.org/pdf/1411.2738.pdf (visited on 06/10/2019).

Sidorov, G. et al. (2014). "Soft similarity and soft cosine measure: Similarity of features in vector space model." In: *Computación y Sistemas, 18(3)*, pp. 491–504.

Tang, D., B. Qin, et al. (2015). *Effective LSTMs for target-dependent sentiment classification.* Cornell University, arXiv preprint. URL: https://arxiv.org/pdf/1512.01100.pdf (visited on 06/10/2019).

Tang, D. and M. Zhang (2018). "Deep Learning in Sentiment Analysis". In: *Deep Learning in Natural Language Processing, Deng, L. and Liu, Y.* 152 Beach Road, 21-04 Gateway East, Singapore 189721: Springer Nature Singapore Pte Ltd, Edition 1, pp. 219–253. DOI: 10.1007/978-981-10-5209-5_8. URL: https://doi.org/10.1007/978-981-10-5209-5_8.

Wang, X. et al. (2015). "Predicting Polarities of Tweets by Composing Word Embeddings with Long Short-Term Memory". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*. Beijing, China: Association for Computational Linguistics, pp. 1343–1353. URL: https://www.aclweb.org/anthology/P15-1130 (visited on 08/08/2019).