MASTER'S THESIS 2021

Exploring the suitability of explainable AI for binary conflation of POI data

Emil Aminy, Marcus Lissner

ISSN 1650-2884
LU-CS-EX: 2021-09

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2021-09

Exploring the suitability of explainable AI for binary conflation of POI data

Undersökning av förklarbar AI för sammanslagning av POI-data

Emil Aminy, Marcus Lissner

Exploring the suitability of explainable AI for binary conflation of POI data

Emil Aminy
emilaminy@outlook.com

Marcus Lissner
dat15ml1@student.lu.se

May 28, 2021

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Frank Camara, frank.camara@afry.com

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

A *point of interest*, or POI, is a geographic location which someone could find useful or interesting, such as a restaurant or tourist attraction. Information regarding such locations exists in large quantities and can be sourced from numerous data aggregators and vendors. It is however not guaranteed that a source provides data that is complete and completely correct. Collecting and conflating data from multiple locations can improve the accuracy of such information. This is not a trivial task and can require complex systems to solve well, creating an uninterpretable black box of reasoning between input and output.

In this thesis, we aim at exploring a solution which can conflate POI data from two different sources while still maintaining a transparent decision making process. To accomplish this, we employ decision trees, a highly explainable machine learning model. Our data consists of POI data gathered from *Eniro*, *Foursquare*, *Hitta*, *OpenStreetMap* and *Tripadvisor*. The results indicate that explainable AI models such as decision trees are a viable and useful tools for POI data conflation; achieving 60-90% accuracy, depending on which property is being conflated, while maintaining a high level of interpretability.

Acknowledgements

We would like to thank, our supervisor, Pierre Nugues for his help with the machine learning aspects of this thesis. We would also like to thank Frank Camara, Dick Max-Hansen, and AFRY for making this thesis possible. And lastly we would like to express our gratitude to Mattias Levin for his support, knowledge, and advice.

Contents

Chapter 1
Introduction

1.1 Background

1.1.1 Point of interest

A point of interest, or POI, is a group term for any geographic location that a human might find useful or interesting. This may be anything from hotels and restaurants to car parks and public toilets. Being geographic locations, these POIs have geographic data, such as coordinates, connected to them. However, POI data may also contain metadata of the POIs, information such as the name of the locations, their contact information, user reviews, or opening hours (Papadopoulos et al., 2015). In Figure 1.1 a number of POIs are presented with their locations and names.

1.1.2 Conflation

The process of merging multiple sets of data into one is referred to as *conflation*. The goal is to get an end result which is more complete and accurate than any of its parts. The main challenge of conflation is the existence of overlapping and conflicting information in the sets of data that are being merged. Overlapping data has to be evaluated in order to select the most correct option from all the values available in the data sets. For this, some sort of decision making process has to exist which can accurately decide which option is the most correct. In this thesis, we will try to create such a process using machine learning algorithms.

1.1.3 Machine learning

Machine learning, or ML, is the field of study which looks at computer algorithms that can improve through experience, without human involvement. It is often considered to be part

Figure 1.1: Examples of different POIs in Lund, Sweden. Source: Screenshot taken from Google Maps

of the wider discipline of artificial intelligence, though it is also closely related to statistics, data mining, and mathematical optimisation.

ML algorithms are algorithms that learn to perform tasks without being explicitly coded to do so. However, the learning process varies from algorithm to algorithm. In general these can be divided up into three categories: these are *supervised learning*, *unsupervised learning* and *reinforcement learning* (Chollet, 2017).

Supervised learning is by far the most common type of ML algorithm. To learn, it requires a set of sample data, also called *training data*, which provides input data as well as the desired output of that data for the algorithm to learn. By being provided answers, it can independently create a model which learns to map the input data to the output data. Once a model is created it can be given new data, not before seen by the algorithm, and make predictions.

Oftentimes this *training data* will be the product of so called human annotation, meaning that humans have gone through all of the input data and manually provided the desired output of that data. This can be seen as one of the biggest downside of supervised machine learning since the creation of annotated data sets is a very time and labour intensive process (Chollet, 2017).

Unsupervised learning as the name might suggest, can almost be considered the opposite of supervised learning algorithms. These methods are referred to as not supervised because no exact purpose is taught when creating them. Data is simply inputted and the algorithm is tasked with grouping together data points into clusters based on any discernible patterns. As such, the most common uses of unsupervised learning are related to data analytics. A benefit of unsupervised learning algorithms is that no training data is required for the creation of models (Chollet, 2017).

Reinforcement learning is entirely different to the above mentioned types of ML. It learns by being rewarded or punished based on how it responds to input. This process creates a model which aims at maximising rewards. To accomplish this, there has to exist an *evaluation*

function which can appraise the responses of the model in order to know when a response deserves a reward or not. The quality of the model created is entirely dependent on how well the *evaluation function* can evaluate responses.

Reinforcement learning has had most of its practical success in learning to play games. This is because games have very clear and defined goals which can be relatively easy to evaluate and score. Having such goals greatly simplifies the creation of an *evaluation function*, a much more difficult task in real world scenarios (Chollet, 2017).

1.1.4 Explainable artificial intelligence

An issue that has arisen with the development of newer and more complex machine learning algorithms is that they have gradually become more and more obscure. Because of their self learning nature, no human involvement is required to create ML models. This leads to the creation of systems that no human has ever been part of to design and that cannot be easily explained and justified by a human. Thus, modern ML algorithms are often referred to as "black boxes".

As a response, the study of methods and techniques focused on explainability has emerged in recent times. This field has quite appropriately been named the study of explainable AI, often shortened to XAI. Many techniques for gaining an understanding of the reasoning behind the decisions of ML algorithms have been developed (Molnar, 2019).

The most straightforward XAI techniques are those that utilise interpretable ML algorithms, i.e. ML algorithms that do not suffer from the "black box" problem. In general, these are slightly older algorithms that create simpler models which can be more easily visualised and understood by humans. Examples of these include *regression*, *decision trees* and *Bayesian classifiers*. While being inherently more interpretable, they often fall short of the results non-interpretable algorithms can achieve due to their simplicity (Molnar, 2019).

To give more advanced ML algorithms some level of explainability, techniques which try to separate the explainability from the algorithms have been developed. These techniques provide a greater flexibility for the choice of algorithm since they can be applied to all of them. In general, they utilise the data an algorithm interacts with to gain some insight into a model's reasoning. This is done by manipulating the data and observing how outputs change in response. Examples of these techniques are for instance *LIME* and *SHAP* (Molnar, 2019; Ribeiro et al., 2016; Lundberg and Lee, 2017).

Lastly, there are algorithm-specific techniques, techniques which can provide significant interpretability but only to specific ML algorithms. *Layerwise relevance propagation* (LRP) is one such example. It is a technique that grants some explainability to *neural networks*, a popular ML algorithm which is highly un-explainable without such tools (Montavon et al., 2019).

1.1.5 Decision tree

A decision tree is a tool which can be used to make decisions based on conditional control statements arranged in a tree-like structure. Each node in such a tree represents some form of condition and all outcomes of said condition will either lead to the next condition or to an end node. The end nodes contain decisions based on the path followed through tree.

Decision trees can be utilised as models for supervised machine learning algorithms. By analysing the training data, these algorithms can recursively split the data based on its features until some level of homogeneity is achieved. This creates the series of conditions and outcomes that characterise a decision tree and can be used to make predictions of future data.

Basing a ML model on a decision tree allows for easy visualisation of the models as tree graphs, making them highly explainable. The simplicity of the models does however also make them prone to overfitting (Russell and Norvig, 2009).

1.1.6 Overfitting

Supervised machine learning algorithms construct models from training data, which allows them to make predictions when shown new data. The accuracy of these predictions depends on how well constructed the models are, also referred to as how well the models have been *fitted* to the data.

Issues can arise when these models are not properly fitted. The most problematic instance of this is when a model is fitted too well to the training data, also called *overfitted*. Since the training data is finite, the ML algorithm can, if given enough time and training, find overly complex and arbitrary patterns to make perfect predictions from the training data. An overfitted model will however not be applicable to any data set beyond the training data since the rules it has learned are too specific and narrow for any other piece of data. (Chollet, 2017).

1.2 Purpose

The purpose of this thesis is to explore the possibilities and limitations presented by explainable AI when used for conflation of POI data gathered from multiple sources. The goal is to create a system which can perform this non-trivial task well while still maintaining a high level of transparency, granting human insight into the system.

1.3 Problem statement

The questions we seek to answer in this thesis can be condensed down into the following:

- How accurate is an approach utilising explainable AI? Can it reliably conflate POI data? Are results comparable to black box machine learning approaches? Are results comparable to human-level performance?

- How usable is an explainable AI approach for conflation of POI data? Do the explainability yield any valuable information or insight that can be used to improve future conflation attempts?

1.4 Related work

1.4.1 Clustering

While the aim of this thesis is to explore data conflation, there is an element of data clustering involved as well. The clustering of POI data has been attempted in several other papers.

Scheffler et al. (2012) attempt to cluster POIs from *Facebook Places* and *Qype* to their counterparts from *OpenStreetMap* (OSM). The proposed approach utilised a metric which took into account the geographic distance between POIs as well as the edit distance between their titles. The data set consisted of 100 POIs achieving results of approximately 70-80% cluster accuracy.

McKenzie et al. (2014) attempt to cluster POIs from *Yelp* to their counterparts from *Foursquare*. The proposed approach performed multiple clusterings using a different property of the POIs as matching criteria in each. The properties were given a weight based on the accuracy achieved by their respective clustering. A final clustering was performed using multiple properties together modified by the previously calculated weight. The data set consisted of 200 POIs achieving a result of over 95% cluster accuracy.

These two papers approach the clustering slightly differently than us. In their approach, they find a limited number of POIs from a dataset. They then manually find the corresponding POIs from another data set and attempt to create a matching algorithm which can identify these POI pairs. This differs from our approach since their data is constructed such that all POIs have exactly one other matching POI. Rather than taking multiple sets of POI data and identifying clusters of matching POIs, they take a set of matching POI clusters and try to see how many of the clusters they can identify.

Novack et al. (2018) attempt to cluster POIs from *Foursquare* and *OpenStreetMap*. The proposed approach utilised a metric which took into account the geographic distance between POIs, the edit distance of their names as well as the semantic distance between words in their names as measured by the English semantic network *WordNet*. A bidirectional graph was constructed from these similarity values, with nodes being the various POIs and similarity values being the edges. Graph theory was applied to create clusters from the interconnected nodes. The data set consisted of 21786 POIs, with the best result reaching 94% accuracy.

This approach is much more resemblant to ours. It takes two sets of POIs and try to find clusters in a very similar manner to us. The main difference between our approach and theirs is that we used a larger number of sources for our data, which added a level of complexity to our clustering process since clusters could contain more than two POIs.

Piech et al. (2020) attempt to cluster POIs from *Foursquare*, *Yelp*, *Facebook Places* and *OpenStreetMap*. Clustering was done using multiple machine learning algorithms, namely k-nearest neighbour, decision tree, random forest, isolation forest, perceptron, and feed-forward deep neural network. The data set consisted of 4000 POIs with results ranging from 80-90% depending on the machine learning algorithm used.

This paper uses more than two data sources, much like ourselves. Their approach, however, is more advanced than ours, utilising various machine learning algorithms for the clustering. Such an approach was considered by us, but since the clustering was not the main focus of this thesis we decided to use a simpler rule based approach.

1.4.2 Conflation

Several papers have also tested various approaches to conflation of map data. Hastings (2008), Adams et al. (2010) and Anand et al. (2010) all take rule based approaches of conflation. However unlike our work, these focus on geospatial objects such as lakes, roads, and buildings rather than POIs. As such, the purpose is to conflate data relating to the geometric shape and geographic positioning of the objects. Our goal of conflating POIs focuses less on such data and rather aims at resolving meta data conflicts.

The most similar paper to ours is probably Yu et al. (2016) which also focuses on POI conflation. A rule based approach is also taken here. Several difficulties relating to conflation are brought up; such as missing values, outdated data, neighbouring POIs that should stay separate and inconsistent data. A *conflation framework* is created to process, improve, and normalise the data set to finally cluster and conflate POIs according to a predefined number of rules. The purpose of the paper was to present a POI conflation method and generate ontologies based on it, more so than actually evaluating the accuracy of an implemented conflation algorithm.

Chapter 2
Theory & techniques

This section defines and explains some terms and concepts used in this thesis. They are in no way exhaustive explanations, but will at least provide the reader with enough knowledge to be able to sufficiently understand the rest of the content. References for respective concept are provided for further reading.

2.1 Web scraping

Web scraping is a term for collecting data from websites. This is achieved by downloading a web page and parsing data of interest from certain HTML elements. It is commonly used in automated processes like search engine indexing to generate databases of searchable websites, or to parse data from several websites for the purpose of summarising and comparing products, sensor data and the like.

2.2 Levenshtein edit distance

Levenshtein edit distance is a way to measure the difference of two strings. The distance is calculated as the number of single-character edit operations that are required to transform one string into another. The edit operations permitted by Levenshtein are insertion, deletion, and substitution (Levenshtein, 1966). The distance between two strings a, b can therefore be expressed as shown in Figure 2.1.

2.3 Agglomerative clustering

Agglomerative clustering is a bottom-up hierarchical clustering method. It is initiated by placing all nodes in separate clusters. The algorithm then measures the distance between

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) & \text{if } a[0] = b[0], \\ 1 + \min \begin{cases} \operatorname{lev}(\operatorname{tail}(a), b) \\ \operatorname{lev}(a, \operatorname{tail}(b)) \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) \end{cases} & \text{otherwise.} \end{cases}$$

Figure 2.1: Levenshtein edit distance expressed mathematically, using the recursive method.

each cluster and merges them if close enough, forming a single combined cluster. This is repeated until all nodes are connected or a specified distance threshold is crossed. Figure 2.3 shows a finished tree, built from the individual nodes and where the branches represent the hierarchical clustering.

The distance metric can be determined in multiple ways. Three of the most common methods are *complete-linkage clustering*, *single-linkage clustering* and *average-linkage clustering*. Complete-linkage clustering measures the distance between nodes in separate clusters farthest from each other, single-linkage clustering measures the distance between nodes in separate clusters closest to each other, and average-linkage clustering measures all distances between two clusters and uses the average distance. These distances are defined in 2.1, 2.2 and 2.3, respectively.

$$\max\{d(x,y) : x \in \mathcal{A}, y \in \mathcal{B}\} \tag{2.1}$$

$$\min\{d(x,y) : x \in \mathcal{A}, y \in \mathcal{B}\} \tag{2.2}$$

$$\frac{1}{|\mathcal{A}| \cdot |\mathcal{B}|} \sum_{x \in \mathcal{A}} \sum_{y \in \mathcal{B}} d(x,y) \tag{2.3}$$

2.4 ID3

ID3, or *Iterative Dichotomiser 3*, is an algorithm for building a decision tree invented by Quinlan (1986). The algorithm uses the *entropy* and *information gain* metrics to split branches optimally in an iterative manner. The involved steps are as follows:

Given a data set S, calculate the entropy

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x) \tag{2.4}$$

where X is the set of classes and $p(x)$ the proportion of class x present in set S, i.e. $\frac{\#(x)}{\#(S)}$. Entropy is a metric used for describing the level of uncertainty in a set. If $p(x) = 1$, we know all outcomes and the resulting Entropy is $H(S) = 0$. On the other hand, if all classes are equally present in the set, the entropy is instead $H(S) = 1$, as there's no indication of what the variable's outcome will be.

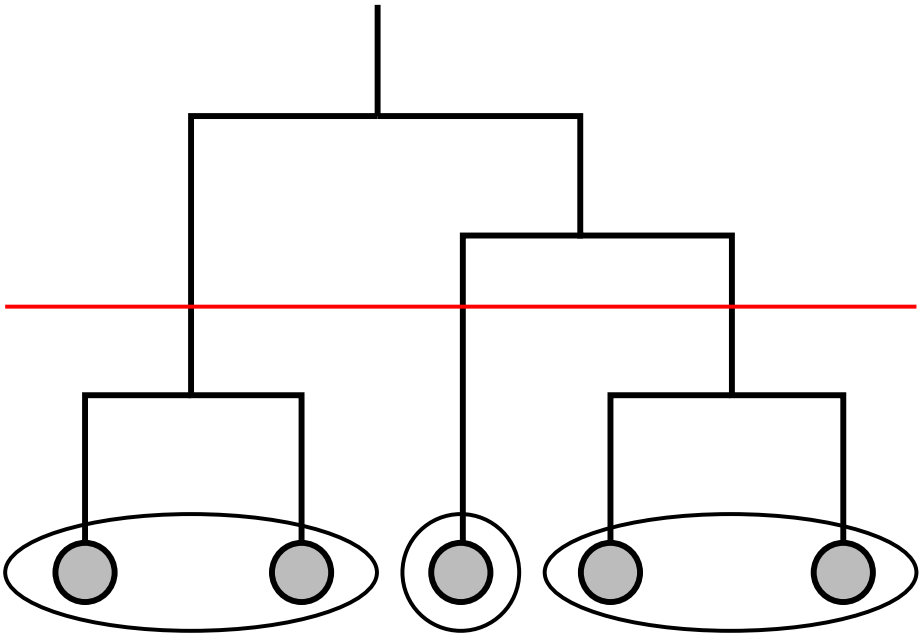

Figure 2.2: A tree displaying the agglomerative clustering process. Nodes are hierarchically clustered to their nearest neighbour. The red line marks the distance threshold, halting the clustering at a certain point. This creates the circled clusters of nodes.

Then, for every subset of feature, calculate the information gain

$$IG(S, A) = H(S) - \sum_{t \in T} p(t)H(t) \tag{2.5}$$

where A is the feature which S is to be split on, $H(S)$ is defined in Equation 2.4, T is the set of subsets derived from splitting set S on feature A, $p(t)$ is the proportion of the number of values in the feature subset t compared to the number of values in set S, and $H(t)$ is the entropy of the subset. The information gain is a metric on the difference in entropy before and after the set S is split on feature A.

The set S is then partitioned into subsets based on the feature split with the highest information gain. The feature is removed from the currently available in set X and added to the decision tree as an intermediate node. The process is then repeated for every subset created from the split.

The tree building is halted if either of the following criteria are fulfilled:

- The subset contains only one class. A leaf node is added with the class at the current position in the decision tree.
- The set X of currently available features is empty. A leaf node is added with the class most common in the subset.

Most ID3 implementations do however include additional stopping criteria to avoid unnecessary leafs which can cause overfitting.

Figure 2.3 shows a trained decision tree with the features *Outlook*, *Humidity* and *Windy* returning a binary classification, *Yes* or *No*.

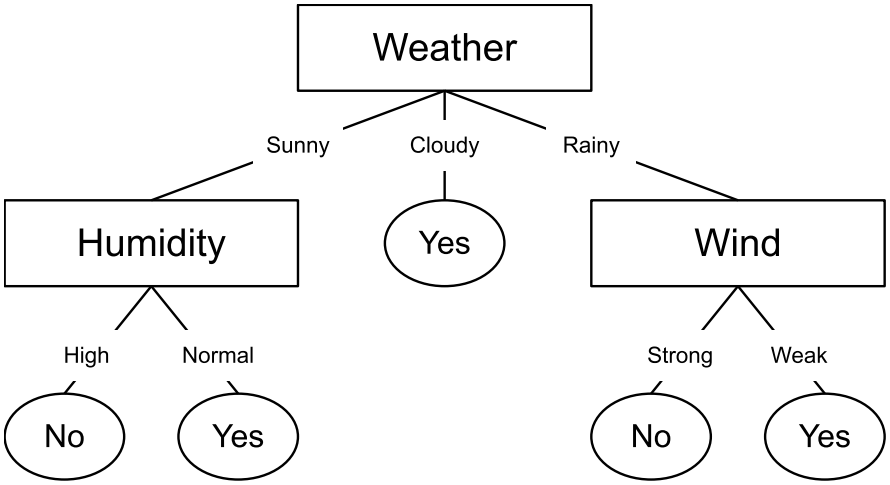

Figure 2.3: An example of a trained decision tree model predicting if the weather is favorable for outdoor activities.

ID3 is a greedy algorithm, meaning it only considers the features available at the current node. This results in a locally optimal decision tree. There is also a risk of overfitting, making the decision tree model excessively precise for a specific data set, but unreliable for classifying new unseen data.

Typically ID3 decision trees can only be applied on data sets that exclusively contain categorical feature values. The decision tree algorithm used in this thesis did however modify the ID3 algorithm to also have the ability to handle numerical values as input. In essence, this is achieved by turning numerical values into a categorical ones by creating a threshold value. This implicitly divides the numerical values into two categories, those greater than the threshold and those lesser. To find a numerical threshold, a split is done for each of the values in the set. The value that achieves the split with the highest information gain is then selected as the threshold. This modification is one of those added in C4.5, the successor to the ID3 algorithm, also invented by Quinlan (1993) (Hssina et al., 2014).

2.5 Cross validation

Machine learning models are commonly trained on a set of training data and tested on a different set of testing data. This is to avoid high results due to overfitting. In some cases, models need to be validated before being tested. Validating a model on the training data will reintroduce the risk of overfitting. Therefore a third set of data is required, a validation set. Partitioning a data set into three parts can however be an issue if data is limited, as was the case in this thesis. A solution to this problem is cross validation.

Cross validation evaluates the trained model by creating having the annotated data split into a predetermined number, k, of smaller subsets, where one subset is used as the validation set and the rest, $k-1$, as training sets. This is repeated so that all subsets act as the validation set once, as seen in Figure 2.4. The overall performance score is calculated from the average of all iterations (Pedregosa et al., 2011; Hawkins, 2004).

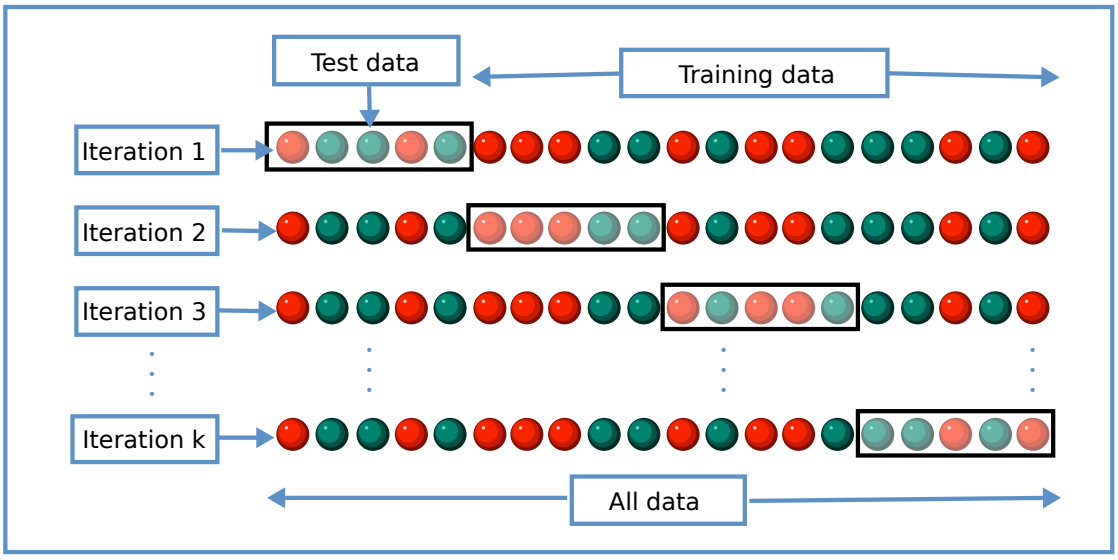

Figure 2.4: Cross validation with k folds. Source: Gufosowa, CC BY-SA 4.0, via Wikimedia Commons

2.6 Performance metrics

To actually evaluate models, some kind of statistical measure is needed to be able to analyse and compare results in a meaningful way. There exists many types of metrics that give insight into the qualities of the evaluated classifications, but some of the commonly used ones in machine learning are *precision*, *recall* and *F1 score* (Hossin and Sulaiman, 2015).

Let TP, TN, FP and FN be defined as the following:

- *True positive* (TP) is when the model correctly classifies the input as positive.
- *True negative* (TN) is when the model correctly classifies the input as negative.
- *False positive* (FP) is when the model incorrectly classifies the input as positive.
- *False negative* (FN) is when the model incorrectly classifies the input as negative.

With this new terminology, we define precision and recall in Equations 2.6 and 2.7.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.6}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.7}$$

F1 score is the harmonic mean of the precision and recall. The measure can be seen as a combined performance evaluation of the classification.

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.8}$$

F1 scores for all classifications can further be combined into a *weighted average* for a performance metric on the model overall. Let x_i be the class and w_i be the score to be averaged. The weighted average is then calculated as

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}. \tag{2.9}$$

Chapter 3
Method

3.1 Data collection

Since this project is about machine learning, we required data to train our machine learning model. Our first step was to identify which sources of data we could use to gather data from. We found large number of websites and databases that tracked certain specific types of POI, for instance *Skånetrafiken* which only kept track of bus stops and train stations, or *Tripadvisor* which only collected hotel and restaurant data.

However, since the goal was to create a model for conflating data, we required data that was conflatable. In other words, we required sources which provided data of the same type of POI. Because of this, we had to constrain ourselves to the types of POI which we found to be most abundantly available from multiple sources. The types of POI that we settled on were restaurants and hotels. This not only gave us a larger number of sources to pick from, but also sources which provided a larger quantity of POI metadata.

Additionally, due to our familiarity with Swedish services such as *Eniro*, *Hitta* or *Merinfo* we simply had access to more sources which exclusively provided data of Swedish POIs. Because of this we constrained ourselves to only Swedish POIs since this allowed us to find a larger number of sources.

With these two constraints we ended up with the following five sources: *Eniro*, *Foursquare*, *Hitta*, *OpenStreetMap*, and *Tripadvisor*.

Eniro is a Swedish local search service which specialises in aggregating map data and data of individuals and companies in Sweden, Norway, Denmark, and Finland. They acquire this data from a number of companies, governmental agencies and open databases (Eniro, 2021b,c). They provide access to this data freely through their website but also offer an API service for users that pay a monthly subscription fee (Eniro, 2021a).

Upon inspection we found that all of the POI data was available in a JSON-object in the HTML page source for each POI. Since this simplified the amount of work necessary to

web scrape the data from their website, we chose to do this rather than use the API. For this purpose we used the Beautiful Soup library for Python 3.

The main issue we encountered during this process was the way *Eniro* displayed their POIs. Search queries were displayed in pages of fifteen POIs each, however only 400 pages were accessible for each search query. Therefore only 6000 POIs could be scraped for each search query. To circumvent this we had to search for the POIs in smaller regions. Fortunately *Eniro* permitted searches within a rectangular area defined by the latitudinal and longitudinal coordinates of its corners. By doing this we were able to divide up Sweden into smaller regions which yielded fewer than 400 pages of POIs and collect them in multiple batches. To avoid missing POIs in the borders between the rectangular areas a small overlap was used after which duplicate POIs were removed.

Foursquare is an American local search service with a global focus. The company offers a consumer product for local searches, as well as several different enterprise API services (Foursquare, 2021b). One of their business products is *Places by Foursquare*, an API service which provides desired data from user defined areas (Foursquare, 2021c). The service comes in three versions: a free non-commercial version as well as two different paid versions, depending on business needs. The difference between these versions is mainly the number of API requests allowed.

We chose to use the free version, which came with a daily limit of 100,000 API requests (Foursquare, 2021a). For the purpose of finding POIs, the API required longitudinal and latitudinal coordinates of a center point, a search radius around this point and one or several POI categories to search for. Each query returned a maximum of 50 data points regardless of the actual number of POIs in the area covered, limiting the chosen radius of the circular search areas (Foursquare, 2021d). This restriction made searching for POIs in all of Sweden quite difficult and time consuming. Therefore we strictly focused on the ten biggest cities in Sweden by population, allowing us to collect a sufficient number of POIs in a reasonable amount of time.

In order to retrieve POI data from an entire city, we generated a grid of circles to cover the area. The radius of the circles varied between 100 and 200 meters depending on the population and density of the city in question. The circles were positioned in such a way that no circles overlapped. The result of this did however not create a sufficient coverage of the city as gaps occurred in between each group of four circles. This was solved by adding a second grid of circles, offset half a position on both the x- and y-axis. Figure 3.1 shows the difference between the coverage with and without intermediate points. Because of the overlap, any duplicate POIs returned from the API calls were removed before further data processing.

Hitta is another Swedish local search service which specialises in aggregating map data and data of individuals and companies in Sweden. The data they collect is provided by a number of companies, governmental agencies and open databases (Hittapunktse AB, 2021). They provide access to this data freely through their website or through an API for users that pay a monthly subscription fee (Hitta, 2021).

As with *Eniro*, the POI data could be found as JSON objects in the page source of their website. Because of this we chose to web scrape *Hitta* as well. Unlike Eniro there was no upper limit to how many POIs a search returned which simplified the process somewhat. However

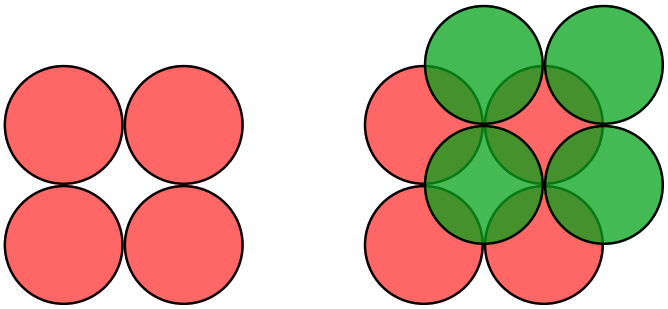

Figure 3.1: A comparison of grid layouts with and without the offset circles.

the page only fetched the JSON object with the POI data using JavaScript. *Beautiful Soup* does not use JavaScript, therefore we had to use the *Selenium* library instead. Apart from this the data collection from *Hitta* was very resemblant to that of *Eniro*.

OpenStreetMap (*OSM*) is a community driven project to make free map data available for personal and professional use worldwide as an alternative to other proprietary, and in some cases locally limited, solutions. Users are encouraged to contribute with local knowledge in order to expand, update, and verify the map and its content (OpenStreetMap, 2021).

There were several ways of exporting POI data from *OSM*. The official website provided the possibility to export data from all map layers within the currently visible map region to an XML file, but this could not be done for areas with more than 50,000 nodes because of their server bandwidth limits. Several other external mirrors and APIs were also available for larger downloads (OpenStreetMap, 2020). We used the *Overpass API* since it was recommended by *OSM* for large-scale data exports.

Tripadvisor is an American online travel company and booking site. Among other products they host a website which provides a wide range of travel related services. It allows users to search and make bookings for transportation, lodgings, restaurants, and travel experiences. The information regarding these services is gathered through user submissions (Tripadvisor, 2021). Access to this data is available freely through their website or through an API.

Since the API did not allow for large-scale downloads, we made the decision to once again web scrape the information we required. Unlike our other data sources the pages for hotels and restaurants were built differently. This meant that creating a single script for web scraping them both was not feasible. Furthermore, the amount of data acquired would also vary between the hotels and restaurants.

For collecting the hotel data, we could find no JSON objects within the HTML document containing this. Instead, the data had to be extracted from multiple different HTML tags on the pages of the POIs. This worked sufficiently well for most of the POI data, but the hotel websites caused an issue. In many cases, the website *Tripadvisor* referred to was not the actual website of the hotel but an affiliate link which would redirect the user to the actual hotel website. Since we could not discern any pattern in the affiliate links which would hint at what the actual website was, the web scraping program had to simply follow the affiliate links to get redirected to the correct site. This significantly increased the time required for the scraping.

The restaurants on the other hand did contain a JSON object within the HTML page source. While this did facilitate the web scraping, restaurants were not displayed in a single list. Instead they were separated by which Swedish county they were located in. The URLs for these county specific lists of restaurants did not follow any kind of pattern and had to be manually collected before they were scraped.

For both the hotel data and the restaurant data the *Selenium* library for Python 3 was used since *Tripadvisor* extensively utilised JavaScript on their website.

Other sources besides the above mentioned five were explored.

We looked at many Swedish websites similar to *Eniro* and *Hitta*, such as *Merinfo* or *Allabolag*. The issue with these was that they often did not categorise their POIs in the same way that *Eniro* and *Hitta* did. This resulted in many incorrect search results. Another issue was that these sites often listed the information of the owner of a POI rather than the information of the POI itself.

We also looked at sites more similar to *Tripadvisor*, like *Momondo* or *Hotels*. While these did contain well categorised POIs with relevant data, they lacked an efficient way of extracting such data. The most common reasons for this was that they lacked a way of listing POIs for larger regions. Web scraping these would have taken too much time for this project. Due to time constraints we concluded that they were not viable options.

Finally we investigated public municipal data. When available, these were easily downloadable. However availability and quality varied substantially between different municipalities. Furthermore. they rarely contained information regarding hotels or restaurants. Because of this we decided not to use them either.

3.2 Normalisation

The data we had gathered required some curating before it could be utilised. This process involved geocoding, reverse geocoding and developing a standardised data format to which every POI was to be normalised to.

Geocoding

Geocoding is the process of converting an address into a geographic coordinate. If, for example, a building is located at the intersection of two streets, its location could be described with two different addresses. Such issues did occur in the data we had collected. By acquiring the coordinates of a POI we gained a way of calculating distance between POIs, granting us an additional metric to use when clustering the POIs from our different sources. For this purpose we decided to use Google's *Geocoding API*.

The data we had collected varied greatly in terms of how complete, correct, and extensive the POIs were. The same POI from multiple sources often had different formatting on names, numbers, websites, and addresses; without any of them necessarily being incorrect. Some examples of variations seen in our source data for included letter capitalization, abbreviations, incorrect spelling, or Swedish characters being inconsistently rewritten using English characters, "ä" becoming "ae" for instance. In Listing 3.1 we see an example of this

Listing 3.1: An example of unprocessed POI data using “oe” and “ae” instead of the preferred “ö” and “ä” from Tripadvisor.

```
...
"@type": "PostalAddress",
"streetAddress": "Sankt Mikaels vaeg 5",
"addressLocality": "Eslov",
"addressRegion": "Skane County",
...
```

Listing 3.2: An example of a street address attribute with the street number included when a separate property already exists for it.

```
...
"zipcode": 41308,
"street": "Linnegatan 55",
"number": "55",
"addressLine": "Linnegatan 55 55",
...
```

where the such a replacement was made. Listing 3.2 shows another example, where the street name component of a POI occurs twice.

Solving these kinds of issues was an additional benefit of using Google’s API. It contained some of Google’s search engine functionalities, making it more lenient and allowing it to parse partially incorrect addresses. We could therefore use it to both acquire coordinate of POIs and standardise their address.

A request to the API had to include a formatted postal address. The API response would then return said address separated into its components, the corresponding coordinates of the address as well as the type of location the address pointed to (Google, 2021). In cases where a faulty address was inputted, the API would either correct the address if possible or return the smallest geographical area it could identify within the input. For example, if it could not find or parse a street number in the address, it would return information regarding the entire street. These geocoded address components and coordinates were used as the primary location data of the POIs in our final data set.

Geocoding was only done to a POI if it had sufficiently complete address data to pinpoint its location. An address was deemed sufficient if it contained a street name and either the postal code or municipality of the POI. Since street names are unique within their municipality or postal code, only one of these was deemed necessary to make an address valid. We did not consider street numbers as required for an address to be valid since some addresses did not contain them. POIs with addresses that did not fulfill these criteria were discarded.

As mentioned previously, the API would not only return the address components and its coordinates, but also the type of location. If the API could not parse an entire address from the input, due to missing or incorrect data, it would return the address of the smallest parseable area instead. The location type of the query response would reflect this. Therefore, if the location type of an address claimed it was an address of a larger region rather than that of a specific place, it would also be discarded from the data set. For example, if the address

"Drottninggatan 103, 113 60 Stockholm" was queried and no location could be found at street number 103, the API would return the address components "Drottninggatan", "113 60" and "Stockholm" with the location type "route". This would signify that no specific location could be found at that address and that the returned address components specify the location of an entire road. Table 3.1 shows a list of the types which were considered too large.

Table 3.1: Location data returned from the *Geocoding API* with one of these location types were discarded because they were not specific enough to pinpoint a location.

`route`
`locality`
`political`
`postal_code`
`postal_town`
`sublocality`
`colloquial_area`
`sublocality_level_1`
`sublocality_level_2`
`administrative_area_level_1`
`administrative_area_level_2`

Data from *OSM* and *Foursquare* lacked addresses completely. Fortunately, all of them did contain a coordinate. For these we completed their data using reverse geocoding. As the name suggests, this is when a coordinate is used to find an address. These were subjected to the same scrutiny as the geocoded POIs and if the Google API did not consider them to be specific enough they were also discarded.

Format

For our final normalised data structure we used GeoJSON (Butler et al., 2016). GeoJSON is a format built on JSON, but specifically designed for describing geographic features. We only focused on individual locations defined by one point and as such only managed `Feature` objects, as defined in the reference documentation. Each such object must have the following members:

- `type` with the value `"Feature"`.
- `geometry` describing a `Geometry` object. This contains a `type` member, which in our case always had the value `"Point"`, and a `coordinates` member containing a list of coordinates. When the `type` is `"Point"`, this is always a pair of coordinates.
- `properties` contains any JSON object. It is typically used to provide metadata for the GeoJSON object. For a POI, that metadata could be a name, an address, contact information and so on.

Listing 3.3 shows one POI example of such an object. We chose our structure according to the GeoJSON format for compatibility reasons. These files could, if needed, be used directly

in applications for visualising the data onto maps. Before this could be done the data required some additional cleaning.

The properties of the POIs which were of interest to us and would later be involved in the conflation process included the the names, phone numbers and website URLs of the POIs as well as how our sources categorised the POIs.

A POI category described what type of POI it was. For instance, an Italian restaurant could have the categories “restaurant”, “italian cuisine” and “pizza”. However, the data sources used different sets of categories for their POIs. To standardise the category names these were translated to a common set of names. Some categories with smaller scopes, such as “brew pub” and “beer garden”, were merged into “bar” for example. This significantly decreased the number of categories to a more manageable set which could be more easily compared between different POIs.

The website URLs also had to be adjusted. We stripped down all URLs to the format `[sub-domains].[domain].[top-level domain]/[path]`, removing unnecessary URL components such as `www` and `https://`. An exception was made for *Facebook* domains since a large number of POIs used *Facebook* as their primary website. These often required the full URL to function and were therefore left as they were.

The phone numbers had to adjusted as well since they often contained unnecessary spaces, dashes, and country codes. These were removed or replaced such that the final phone numbers only contained numeric characters.

The names of the POIs were left as they were since restaurant and hotel names do not follow any standardised format.

Beyond these four properties, we were also interested in the address and coordinate properties we had obtained from the geocoding process since these could be used when clustering the POIs.

3.3 Clustering

To conflate our data we required multiple POIs which referred to the same location to be grouped together. This meant that we had to evaluate all POIs within our data set and based on similarities between POIs create clusters. An issue that we ran into early on in this process were "chained" matches. For instance, if we had 3 POIs: A, B, and C, and it was found that A was similar to both B and C, but B and C were not found to be similar to each other, how would these POIs be clustered? To solve this, we would have had to create a distance matrix with the similarities of the POIs and make a decision based on their relative distances to each other. However, doing so for all POIs within our data set was not feasible with the resources we had at hand and it would have been a highly time and memory intensive process.

To get around this issue we decided to divide our clustering process into two steps. In the first step we would get avoid the issue of "chained" matches entirely by ignoring it, i.e. we considered all POI-chains to be valid. This allowed us to forego the distance matrix. Instead, all POIs which could be matched through some sort of similarity-chain were simply put into the same cluster. We also decided on using more lenient thresholds during this first step. Our goal with this step was to create larger clusters of POIs which were perhaps not exactly the same, but somewhat similar of each other.

Since each of these first-step-clusters had been constructed with very lenient thresholds

Listing 3.3: An example of a normalised POI, complying with the GeoJSON format.

```
{
    "type": "Feature",
    "geometry": {
        "type": "Point",
        "coordinates": [18.0224161, 59.30936759999999]
    },
    "properties": {
        "id": "31219",
        "source_id": "122721645",
        "source_name": "eniro",
        "name": "Espresso House",
        "categories": [],
        "phone": "",
        "website": "espressohouse.com",
        "address": {
            "street_nbr": "38",
            "street_name": "Nybohovsbacken",
            "postal_code": "11763",
            "postal_town": "Stockholm"
        }
    }
}
```

and relationship criteria, we considered all POIs outside of the cluster to not be of interest to those within. This allowed us to use the first-step-clusters in a second clustering process, treating each first-step-cluster as its own data set. We essentially aimed at taking each of the less accurate first-step-clusters and refine them into smaller more accurate second-step-clusters. Since the amount of POIs within a first-step-cluster was far smaller than our entire data set, the amount of comparisons between POIs dropped drastically. This meant that the memory requirements for the construction of similarity matrices was far more manageable.

Point-based clustering

The first clustering step utilised a point-based approach, each POI in our data set was compared to every other POI. POIs were compared using five of their properties. These were the address, coordinates, name, phone number and website URL. A point was granted to a POI pair for each value that was resulted in a positive match. Many POIs lacked one or more properties and if this was the case for either POI in a pair, the property would not be considered. If the score of a pair equaled or exceeded 75% of the considered properties, the POIs in the pair were matched and placed in the same cluster.

For the address, phone number and website URL the compared POIs were required to contain exactly equal values. The coordinates, being a continuous value, allowed us to calculate positions nearby each other rather than requiring absolute equality. Lastly, the names were compared to each other using Levenshtein edit distance (Levenshtein, 1966). This was done because the name of a POI could vary significantly and requiring absolute equality would yield very few matches.

Agglomerative clustering

The second step was an agglomerative clustering. In this step, the only POI property that was considered was the name of the POI. However, unlike in the previous step, the names were pre-processed before comparison. Words that we found were often added to the POI names were removed so that they would not create a false similarity. Examples of such words are for instance *restaurant*, *hotell*, *café* etc. Furthermore, municipality and street names were also removed. For example, "Flipping Burger Bar & Restaurang" or "McDonalds Årsta" would be reduced to only "Flipping Burger" and only "McDonalds". These removals reduced the POI names to their most unique components, allowing for more precise comparisons.

As in the previous stage, Levenshtein edit distance was used for evaluating name similarity. The names of all POIs were once again compared pairwise against all other POIs within a cluster produced in the first step. However, in this step a distance matrix was constructed with these values. This matrix could then be inputted into an agglomerative clustering algorithm and produce its own clusters which were within a specified threshold distance from each other. We used the agglomerative clustering algorithm available in the *scikit-learn* library for Python 3.

3.4 Conflation

Annotation

Since we had created our own data set, none of the data was annotated, requiring us to do so ourselves. We had not considered the size of the clusters during the clustering process. Our data set did therefore contain clusters of two or more POIs. However, since our goal was to perform binary conflation we only annotated binary clusters, i.e. those containing only two POIs.

Our POIs had four properties of interest for the conflation process: *name*, *phone*, *website* and *categories*. We initially annotated the correctness of these properties in 1000 randomly selected POI clusters from our data set. The annotation was done such that each cluster received a label for each of the above mentioned properties. The label a property pair could receive was *left*, *right*, *both* or *neither*. The *left* and *right* labels were given in instances where one POI contained a more correct property value than the other. *Both* labels were given both POIs contained an equally correct property value and *neither* for when neither property value could be considered correct. The verification of property values was done online using *Google*.

After the first 1000 annotated clusters, we tested and evaluated the resulting models. For reasons discussed in Section 6, 300 additional clusters had to be annotated to improve the results of the *phone* and *website* conflation. However, these were not randomly selected but rather chosen based on if the POIs in the cluster had non-empty and conflicting values for the property to be annotated.

Decision tree

For the conflation we utilised the *ID3* decision tree algorithm. Our implementation did however include a slight modification to the *ID3* algorithm which allowed it to handle both categorical and continuous values. The trained models were used together with the *Graphviz* library to visualise their structure. A more in-depth description of the algorithm can be found at Section 2.4.

We chose to perform the conflation through creating a decision tree model for each of the four properties we were interested in. Each model was trained to look at the property values in both of the POIs and decide if either value could be considered more correct than the other or if both or neither could be considered correct.

Feature engineering

To find suitable features for training our models we took a very iterative approach of trial and error. Our work annotating our data set had given us some insight into possible patterns that might exist, however we continuously evaluated any feature we added or removed. The ability to visualise the decision trees also allowed us to analyse if features were used illogically due to, for instance, overfitting. The results produced by a model trained on a specific set of features was also considered during this process. Since we aimed at creating four different models, four different sets of features had to be decided upon.

Model evaluation

As mentioned, models were continuously evaluated during the feature engineering process. Due to our relatively small training set we employed the technique of cross validation for this evaluation. This allowed us to evaluate feature sets using a greater portion of our data.

Comparative performances

As we had created our own data set, there existed no performance values to compare against. For this purpose we decided on evaluating our data set using two additional methods; a neural network model and human-level performance.

The neural network models were constructed using the *Keras* API for Python 3. As this was merely a value to be used for reference and not the main focus of our project, we utilised a simple sequential model consisting of two dense layers. These models trained for 20 epochs each before being evaluated.

For the human-level performance, we simply classified the clusters ourselves. Unlike the annotation process, we did however not use any external tools such as *Google*. The classification was done with the same data as provided to the decision tree and neural network models. We took a sample of 100 annotated POIs for each of the four properties and tried to classify them in the same way that the model did. We then compared our classification to the actual annotated values to measure our performance.

Chapter 4
Data analysis

Our initial data collection yielded a total of 83,209 POIs. These were divided up such that 24791 came from *Eniro*, 14,850 from *Foursquare*, 15341 from *Hitta*, 8370 from *OSM* and 19,857 from Tripadvisor, as shown in Figure 4.1.

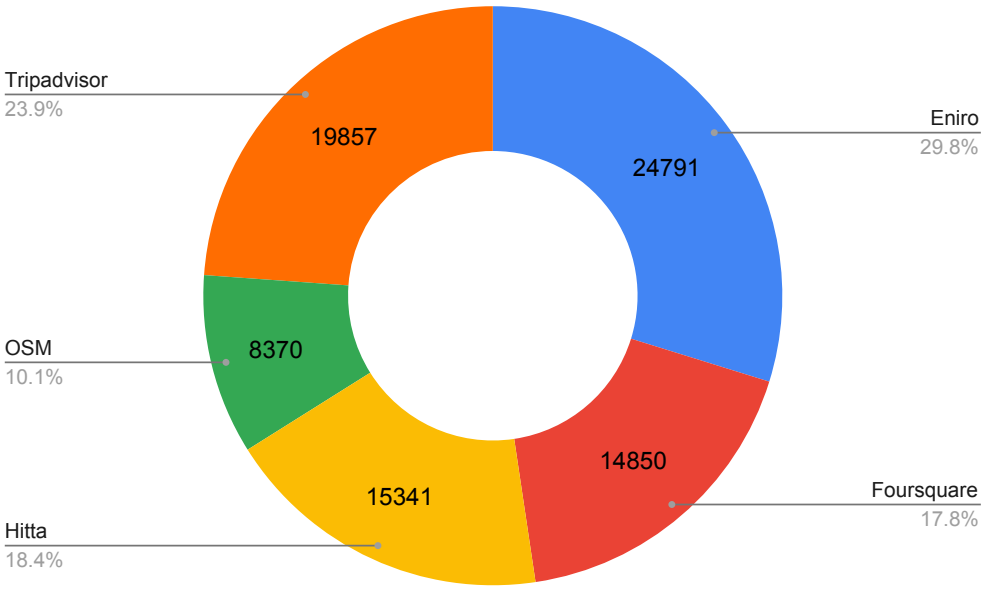

Figure 4.1: The number of collected POIs per source.

During the normalisation process a number of POIs had to be discarded due to incomplete or faulty data. As mentioned in Section 3.2, this was done in conjunction with the geocoding process where POIs with incomplete or incorrect addresses were removed. All in all the normalisation yielded 74,857 POIs.

The properties of these POIs which were of interest to us were *name*, *phone*, *website* and *categories*. The presence of these properties varied depending on the source, as can be seen in Figure 4.2.

It is worth mentioning that the lower averages of *Eniro* and *Foursquare* were caused by them entirely lacking data regarding the *categories* property in the case of the former and the *phone* and *website* properties in the case of the latter.

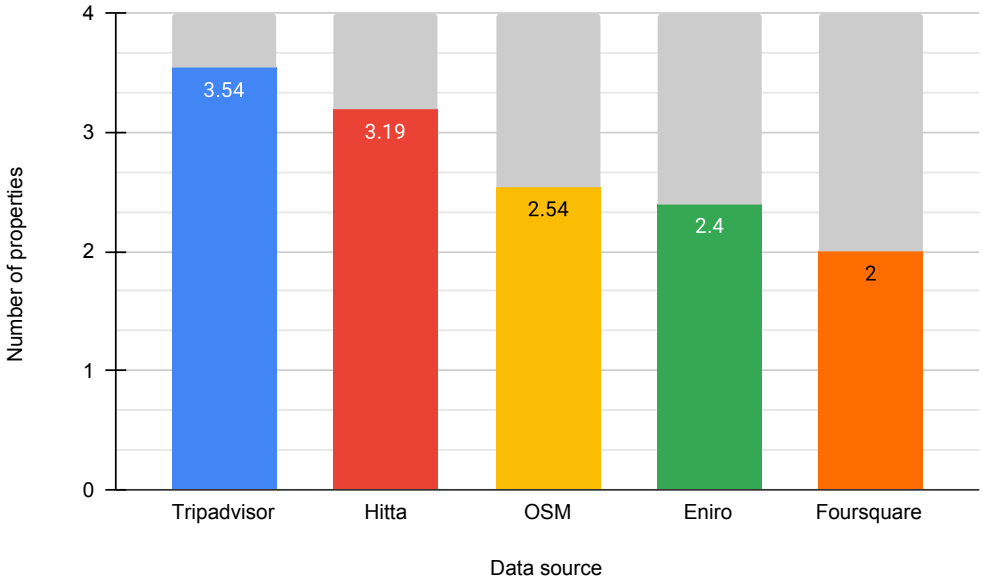

Figure 4.2: Average number of properties in POIs per source

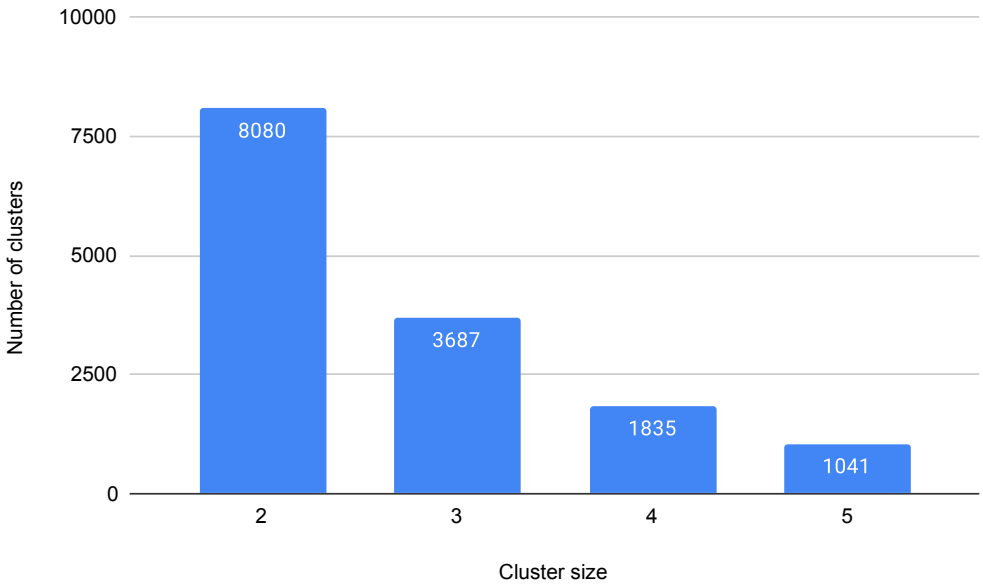

Figure 4.3: The distribution of the clusters according to their size

The clustering process resulted in 14,578 clusters, 8080 of which were binary, seen in Figure 4.3. Since our goal was to perform binary conflation of POIs, these 8080 were the ones of interest. However not all of these POIs contained all four properties and some properties were rarer than others as seen in 4.4.

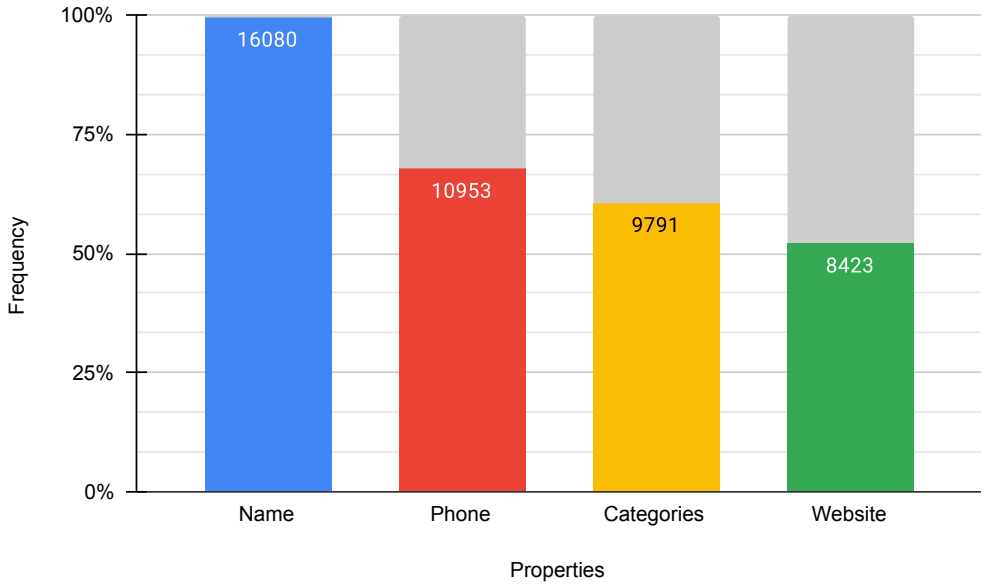

Figure 4.4: The number of POIs containing each property among only the binary clusters.

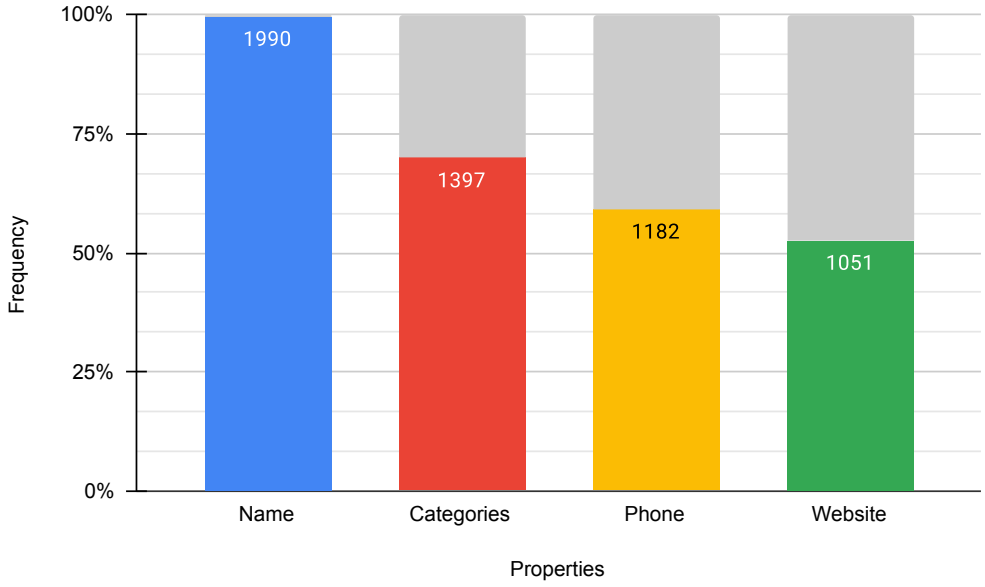

Figure 4.5: The number of POIs containing each property among the annotated binary clusters.

Of these 8080 we initially annotated 1000 randomly selected clusters. As can be seen in Figure 4.5 the distributions of properties remained relatively similar to those from the entire set of binary clusters. We therefore considered our annotated data to be fairly representative of the entire data set of binary clusters.

For reasons discussed in Chapter 6, 300 additional clusters had to be annotated specifically for the *phone* and *website* properties, 100 for the former and 200 for for the latter. These were however not entirely randomly selected, it was ensured that both of the POIs in the binary clusters contained the respective property and that the values of properties were not the same between the two POIs. This caused the training data of the models which conflated *phone* and *website* to have a slightly higher percentage of POIs containing those properties, as seen in Figures 4.6 and 4.7.

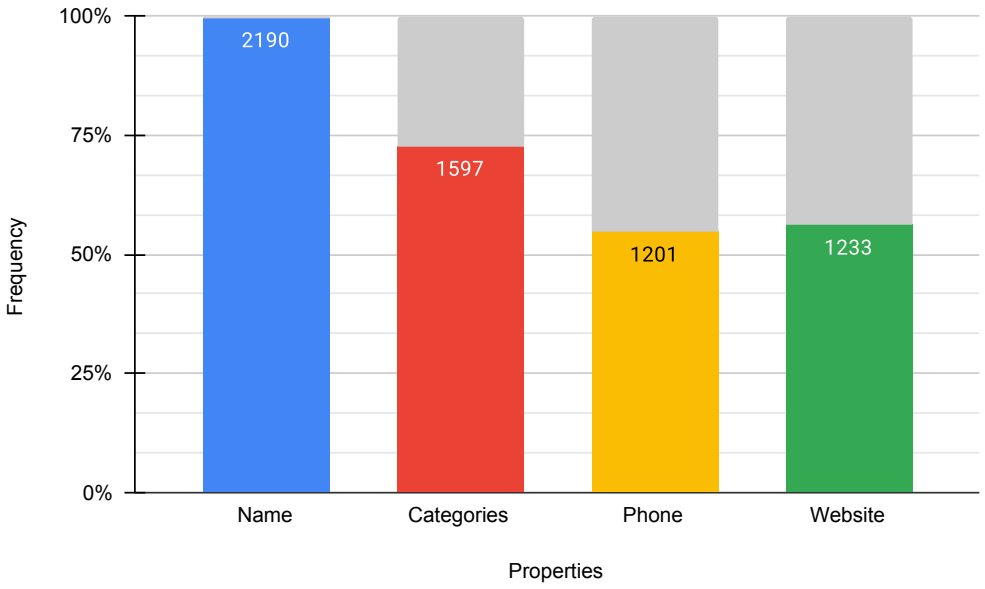

Figure 4.6: The number of POIs containing each property among the annotated clusters with the additional annotations done for the phone property.

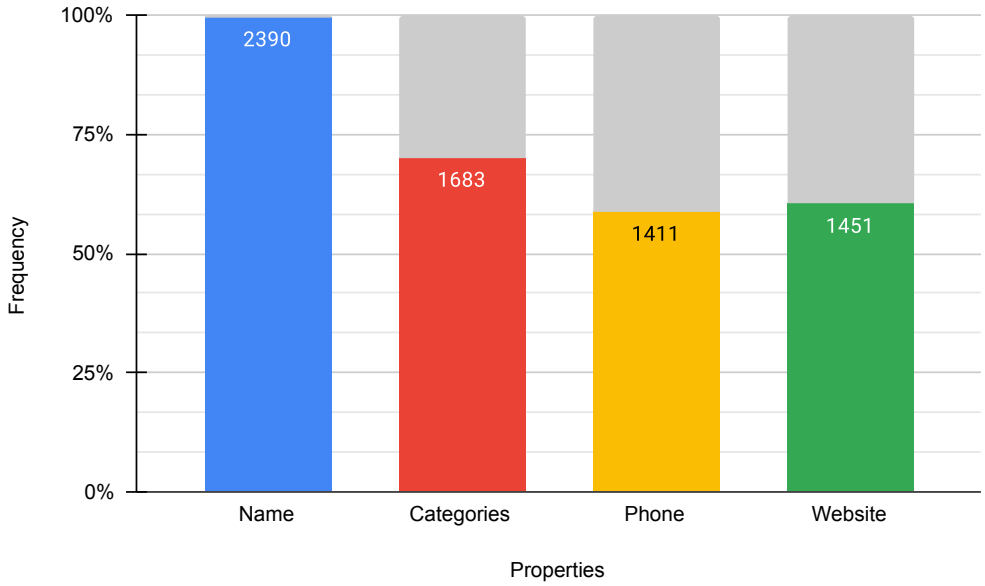

Figure 4.7: The number of POIs containing each property among the annotated clusters with the additional annotations done for the website property.

Chapter 5
Results

The final performance metrics for the decision tree models as well as those of the two comparative methods are presented in this chapter with tables on precision, recall and F1 score. These metrics are defined and explained in Section 2.6. The column *Support* in the performance tables refers to the number of occurrences for respective class in the training set. Additionally, bar charts have been included which display the comparative performances between the three methods.

Both decision tree and neural network models classify input as one of the following four classes: *left*, *right*, *both*, or *neither*. This is meant to designate which POI in the binary clusters containing the most correct value for the conflation. The POI in the first position is referred to as *left* while the second position is referred to as *right*. If either is more correct than the other, one of these two classes are used. If neither or both of the POIs in the cluster are correct the classes *neither* or *both* are used.

Furthermore, we have also added our four decision tree models visualised as tree graphs in the Appendix. We have also added tables containing the sets of features used to generate the training data for the four decision trees are together with short descriptions of each feature to the Appendix. It should be noted that more features are listed in the tables than visible in the tree graphs. This is because the ID3 algorithm reached the minimum amount of samples required to split before it got to use all features available to it. However, all features listed were included in the training data for the decision trees as well as for the neural networks.

5.1 Categories

The decision tree model for conflating the *categories* property performed the best of all the four we constructed. As shown in Table 5.1 we achieved over 90% precision and recall with this decision tree. This more or less matched the results achieved by the human-level performance (Figure 5.1). Surprisingly, the neural network model performance was slightly worse than the decision tree, though it still performed quite well overall. We did not expect this

result since neural networks, in general, perform better than decision trees.

The number of features used for this property was small in comparison to the other three. This is shown in Table 1 and is also reflected in the small decision tree produced by this model, seen in Figure 1 in the Appendix.

5.2 Name

The decision tree model for conflating the *name* property performed the worst of all the four we constructed. As shown in Table 5.2 we achieved around 70% precision and recall with this decision tree, though this varied a lot depending on the classification. The results are brought up a lot by the high precision and recall of the *both* class. The *neither* class did not appear that often in the training data resulting in the low accuracy for this class.

Unlike our other trees, the comparative performances seen in Figure 5.2 showed a large gap between the different methods. The poor results of the neural network model are especially unique among our results. It also surprised us again that the neural network performed worse than the decision tree.

The number of features used for this property was the largest of the four models, as can be seen in Table 2. The tree is however similarly sized to those of the *phone* and *website* properties, as seen in Figure 2 in the Appendix.

5.3 Phone

The decision tree model for conflating the *phone* property achieved around 80% precision and recall, as shown in Table 5.3. While not having as much variance in the performance scores as the *name* property, some variation is visible between classes. The precision for the classes *both* and *neither* were significantly higher than those of *left* and *right*. This gap was slightly smaller when looking at recall. Here however, *neither* performed worse than all three of the other classes.

The comparative performance seen in Figure 5.3 shows that all three methods performed very similarly to each other, though the decision tree model clearly produced the worst results. This was however what we expected, since we believed that both a neural network and us as humans would be able to outperform the decision tree.

5.4 Website

The decision tree model for conflating the *website* property achieved around 80% precision and recall, as shown in Table 5.4. Overall it performed very similarly to that of the *phone* property. The variance among the results for the different classes was however even smaller. The only value that stands out is the high precision of the *neither* class.

The comparative performance seen in Figure 5.4 shows, as with the *phone* property, that all three methods performed very similarly to each other, though the decision tree model again clearly produced the worst results. We did however expect such results, as mentioned above.

Method	Class	Precision	Recall	F1	Support
Decision tree	Left	0.873	0.948	0.906	382
	Right	0.915	0.863	0.883	336
	Both	0.987	0.97	0.976	81
	Neither	0.993	0.895	0.94	201
	Weighted average	0.92	0.911	0.911	1000
Neural network	Left	0.84	0.845	0.842	336
	Right	0.843	0.886	0.864	382
	Both	0.987	0.97	0.976	81
	Neither	1	0.895	0.943	201
	Weighted average	0.885	0.881	0.882	1000
Human-level	Left	0.923	0.947	0.935	38
	Right	0.938	0.938	0.938	32
	Both	0.857	0.75	0.8	8
	Neither	0.909	0.909	0.909	22
	Weighted average	0.919	0.92	0.919	100

Table 5.1: Performances of the three different methods for the conflation of the *categories* property

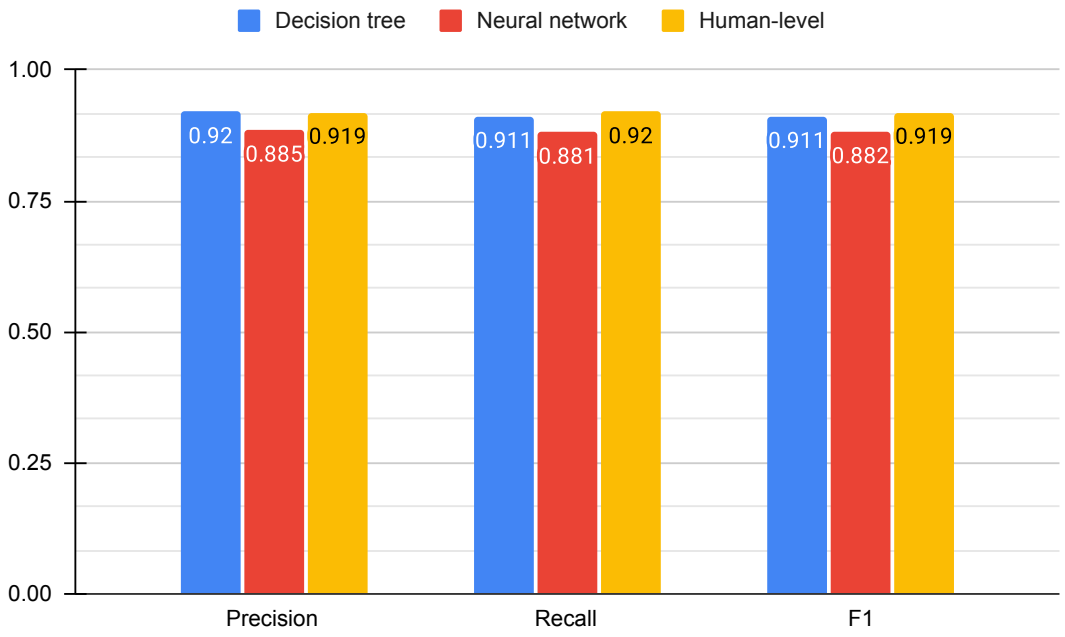

Figure 5.1: Bar chart displaying the weighted average performances of the three different methods for the conflation of the *categories* property

Method	Class	Precision	Recall	F1	Support
Decision tree	Left	0.617	0.55	0.564	300
	Right	0.574	0.623	0.582	316
	Both	0.957	0.908	0.931	360
	Neither	0	0	0	24
	Weighted average	0.711	0.689	0.688	1000
Neural network	Left	0.597	0.327	0.409	300
	Right	0.555	0.446	0.483	316
	Both	0.662	1	0.794	360
	Neither	0	0	0	24
	Weighted average	0.593	0.599	0.561	1000
Human-level	Left	0.679	0.704	0.691	27
	Right	0.718	0.757	0.737	37
	Both	1	1	1	33
	Neither	0	0	0	3
	Weighted average	0.779	0.8	0.789	100

Table 5.2: Performances of the three different methods for the conflation of the *name* property

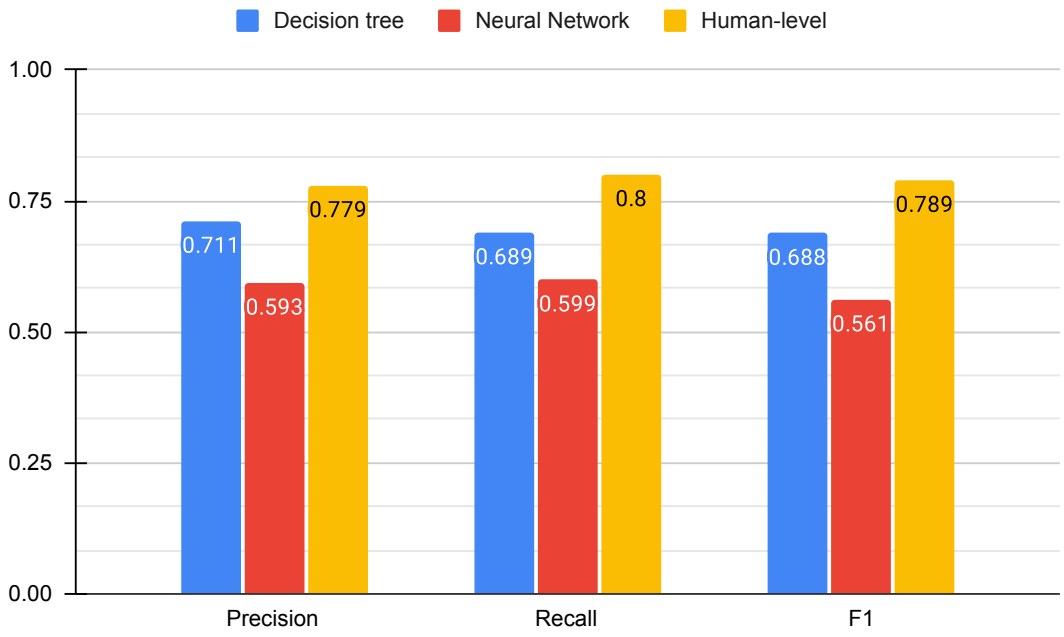

Figure 5.2: Bar chart displaying the weighted average performances of the three different methods for the conflation of the *name* property

Method	Class	Precision	Recall	F1	Support
Decision tree	Left	0.681	0.828	0.744	285
	Right	0.651	0.735	0.69	253
	Both	0.942	0.933	0.937	345
	Neither	0.992	0.479	0.641	217
	Weighted average	0.817	0.771	0.772	1100
Neural network	Left	0.719	0.86	0.782	285
	Right	0.683	0.775	0.723	253
	Both	0.942	0.936	0.939	345
	Neither	0.91	0.516	0.652	217
	Weighted average	0.818	0.796	0.792	1100
Human-level	Left	0.697	0.92	0.793	25
	Right	0.778	0.778	0.778	27
	Both	0.862	0.962	0.909	26
	Neither	1	0.58	0.734	22
	Weighted average	0.828	0.818	0.806	100

Table 5.3: Performances of the three different methods for the conflation of the *phone* property

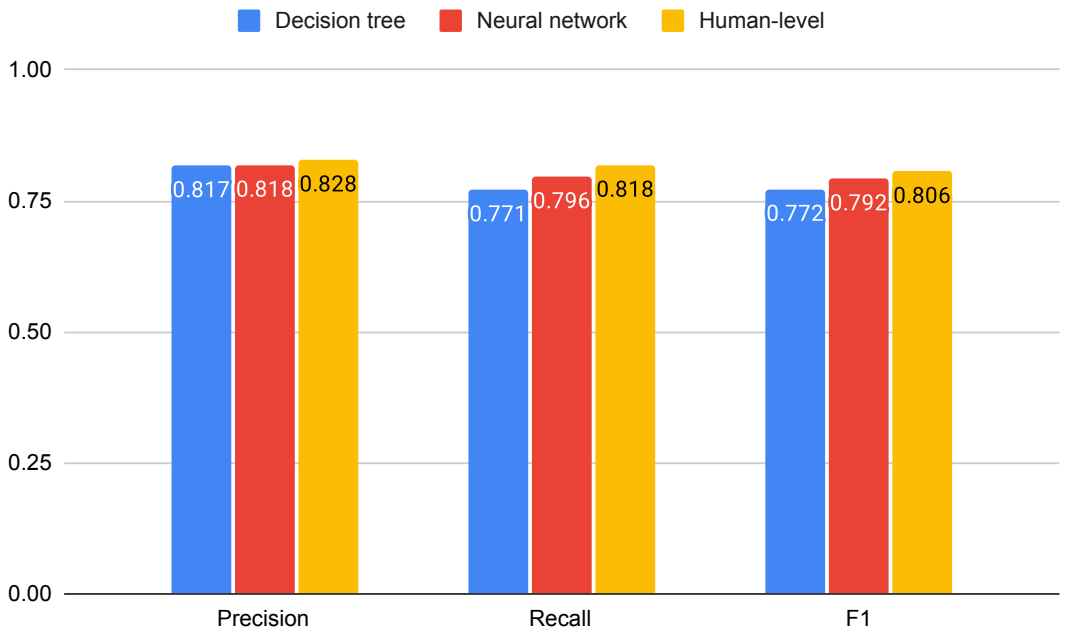

Figure 5.3: Bar chart displaying the weighted average performances of the three different methods for the conflation of the *phone* property

Method	Class	Precision	Recall	F1	Support
Decision tree	Left	0.771	0.743	0.746	307
	Right	0.691	0.876	0.769	322
	Both	0.774	0.763	0.765	255
	Neither	0.98	0.712	0.819	316
	Weighted average	0.805	0.775	0.775	1200
Neural network	Left	0.759	0.808	0.776	307
	Right	0.71	0.885	0.785	322
	Both	0.844	0.751	0.792	255
	Neither	0.953	0.706	0.808	316
	Weighted average	0.815	0.79	0.79	1200
Human-level	Left	0.783	0.844	0.812	32
	Right	0.794	0.823	0.808	22
	Both	0.941	0.889	0.914	18
	Neither	0.833	0.714	0.769	28
	Weighted average	0.828	0.811	0.817	100

Table 5.4: Performances of the three different methods for the conflation of the *website* property

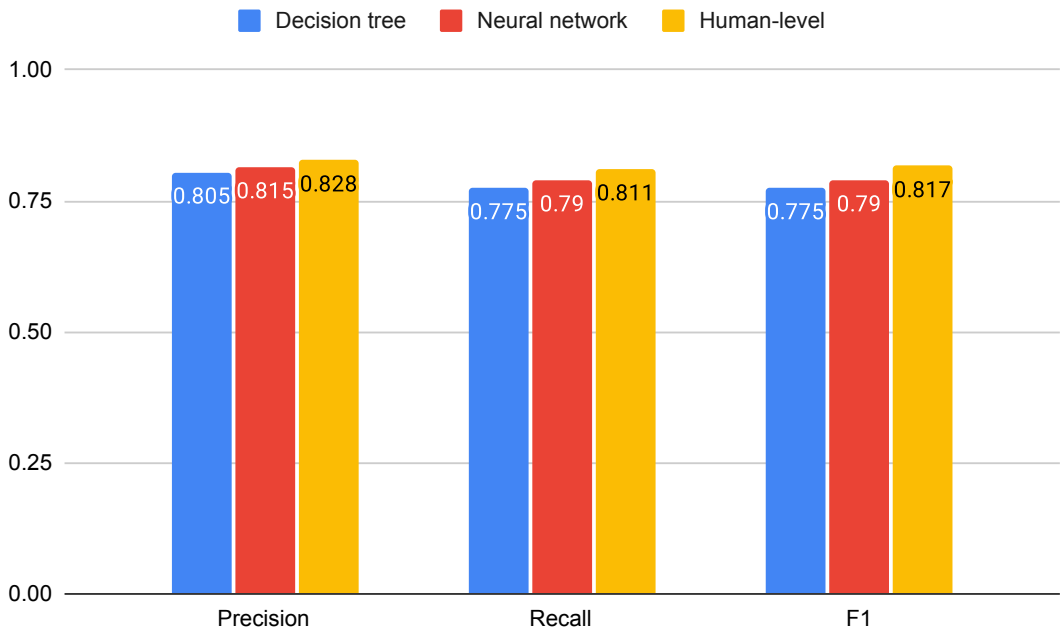

Figure 5.4: Bar chart displaying the weighted average performances of the three different methods for the conflation of the *website* property

Chapter 6
Discussion

6.1 Model performance

All our models were taught to look at a specific property in both of the POIs in the binary clusters and decide which value it considered more correct. Though our four models were each tasked with evaluating different properties, some common traits were noted between them.

During the annotation process we noted that many clusters had POIs which both contained the same property values. In these cases, the value presented by both POIs was almost always correct. In fact, most clusters which were annotated as *both* were of this type. This pattern was learned by all four of our models, as can be seen in the visualised trees presented in the Appendix and led to the models all having relatively high precision and recall scores for the *both* classification.

The fact that not all POIs contained all of their properties became significant. In many clusters, only one of the POIs in the pair would contain a property value while the other would be missing. This caused some issues during the training, since it ended up with the models lacking training data of clusters where both values existed. Because of this some additional annotation had to be made for some of the models to rectify this lack of training data.

Categories

The model trained to conflate the *categories* property produced the best results. The reason for this was that there existed a very simple feature to decide which *categories* property value was more correct. The POI in the cluster which contained more categories would almost always be the more correct one. For instance, if a cluster for a pizzeria contained the category “restaurant” in one of its POIs and “restaurant, pizza” in the other, usually the latter would be correct.

This would suggest that much of the information regarding the *categories* property was highly accurate. Since it was accurate, the more categories a POI contained the better it would most likely be. While we cannot be sure of the reason behind this accuracy, we believe it might have to do with the sources we selected. All of them provide some type of search feature on their websites for users to locate POIs. Categorisation of data by search term facilitates searching tasks and gives their users better results. Because of this we think that a lot of effort is placed on accurately categorising POIs on their part.

Due to the rarity of the *categories* property, the issue of missing values was present for this model. However since a such clear relationship existed between the number of categories and correctness we decided not to annotate more data to remedy this issue. We, as humans, concur with the relationship that our model found and believe little more could be learned from training a decision tree with additional data.

As to the alternative methods of classification used for comparison, they all performed quite similarly. The neural network surprisingly produced worse results than the decision tree. We believe this might be caused by the simplicity of the relationship lending itself well to the simple design of decision trees while at the same time confusing the neural network. The fact that it has to heavily favour the number of categories over all the other features means that the weights within the network have to be significantly adjusted. It is possible that the 20 epochs of training we provided were insufficient for such an adjustment, i.e. that the model was underfitted.

The human-level performance produced slightly higher results than the decision tree model. This shows that some additional work could be done to improve the model. What these improvements might include is further discussed in Section 6.3.

Name

Conversely to the *categories* property, the *name* property was the least accurate model of the four. A major obstacle with the *name* property was that the name of a POI could be highly subjective. For instance, a "McDonalds" could also be referred to as "Mc Donalds" or "McDonalds AB". If it were located in *Lund*, it could also be named "McDonalds Lund". There existed a lot of valid variation, and unlike the other three properties there was some grey area for what could be considered correct and incorrect.

Furthermore, it was challenging to extract any type of information from the name itself. The model which handled the *website* property could for instance gain some hints of a property's correctness from the top-level domain of websites since it was more likely that a Swedish POI had a `.se` domain rather than `.dk`. Since there is no format for restaurant and hotel names to follow, such useful features were harder to find.

The fuzzy nature of the *name* property made the annotation and feature engineering process for this model significantly harder. Overall, we believe the features that we found and used were lacking and this is reflected in both the poor results of the decision tree model and even more so by the neural network model. Based on the human-level performance, it becomes clear that significant improvements can be made. In Section 6.3 we discuss improvements that we consider necessary to make this model function.

Phone & website

As with the *categories* property, the *phone* and *website* properties were missing from many of our POIs. However, unlike the former tree, no clear pattern could be found within the initial 1000 annotated POIs. The model trained on these had an especially difficult time deducing the more correct property value when presented two POIs containing different phone numbers/URLs.

Because of this, we needed more training data. An additional 100 clusters were annotated for *phone* and *200* for *website*, however these were ensured to contain POIs with differing property values for the property in question. This markedly improved our results.

Overall, the final decision tree models performed quite well. We believe a reason for this is the fact that both phone numbers and URLs must follow a standard pattern. Such patterns allowed us to extract highly informative features for the models. Furthermore, the model showed that some correlation existed between the two properties. The presence of one in a POI often meant the other, if present, also was correct. Such a strong correlation had not been detected among the other properties.

When compared to the neural network models and the human-level results we can see that they all performed quite similarly. This would suggest that our results were nearly as high as they could get with the data we had gathered. Any further improvements would therefore have to come from improved data. Our thoughts on what type of data that could be used for this is further discussed in Section 6.3.

6.2 Explainability

The second aspect of our experiments which interested us, beyond the accuracy of the conflations, was the explainability of the decision trees and how usable such explainability could be.

As expected, the decision tree models proved to be highly explainable. The visualised trees clearly showed which features were used and in what way. This proved to be highly useful during the feature engineering process. Being able to analyse how a feature was used allowed us to quickly see the features' usefulness and helped us develop better models. It also gave us a level of trust and confidence in the model that it was doing its job correctly. In comparison, we have no way of knowing how the comparative neural network models functions.

6.3 Future work

Overall, we believe there are two areas in which improvements can be made to improve the performances of our decision tree models: data and natural language processing.

We believe the largest limitations of entire project came from the amount and quality of training data collected. All training data had to be manually annotated by us, a very time consuming process. Due to time constraints, this limited the amount of training data we could create, which in turn limited the amount of training we could do with our models. Furthermore, the data we collected was somewhat limited in quality. All our sources were quite bare-boned when it came to what data was provided. We can imagine there being many

different POI properties that could have been very useful. For instance, data regarding the recency of the information within POIs would have most likely been a highly informative feature. Data such as this were lacking in the data collected from all of our sources. We believe that if we perhaps had access to more resources and could pay for map data, instead of only collecting what is freely available, the quality of data would have increased.

The second area of improvement is related to the faults with the model for the *name* property. As previously mentioned, we believe there is some room for improvement for this model since the human-level performance exceeded both the neural network performance and the decision tree performance by a sizeable margin. We think the reason behind this is that we as humans have an understanding of the Swedish language which gives us significant hints on whether or not a *name* property is correct. Replicating this understanding in an AI with some sort of natural language processing system would, in our opinion, significantly improve the performance of the models. Such a system might also be useful on the model conflating the *categories* property. For instance, deducing that a restaurant named "New Delhi Tandoor" is an Indian restaurant is not difficult with the language skills a human possesses. An NLP system might also be able to do this.

Chapter 7
Conclusion

In this project we looked at what benefits explainable AI could provide for the purpose of binary conflation of POI data. We built a data set of POI data gathered from five different sources, focusing exclusively on restaurant and hotel POIs within Sweden. These where then clustered using a two-step clustering process and lastly conflated with a decision tree algorithm. The performance of the decision tree was compared to a neural network as well as to the human-level performance of the task.

The performance from all of our four decision tree models were overall comparable to the neural network and the human classifications used as references. This indicates that a decision tree algorithm is a viable option for this task. The resulting models could also be easily visualised, granting us a clear look at the reasoning used by the models. Such insight was useful during the feature engineering process, showing a tangible use of such explainability. Furthermore, the insight grants a level of confidence in the decision tree which the neural network was lacking.

References

Adams, B., Li, L., Raubal, M., and Goodchild, M. F. (2010). A general framework for conflation. *GIScience 2010*, Extended Abstracts Volume.

Anand, S., Morley, J., Jiang, W., Du, H., Hart, G., and Jackson, M. (2010). When worlds collide: combining ordnance survey and open street map data. In *AGI Geocommunity '10*. London, UK.

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., and Schaub, T. (2016). The GeoJSON format. RFC 7946, RFC Editor.

Chollet, F. (2017). *Deep Learning with Python*. Manning.

Eniro (2021a). Eniro pricing. `https://api.eniro.com/pricing`.

Eniro (2021b). Hjälp | eniro.se. `https://www.eniro.se/hjalp/kartor`.

Eniro (2021c). Kartor, vägbeskrivningar, flygfoton, sjökort & mycket mer på eniro.se. `https://kartor.eniro.se/`.

Foursquare (2021a). Getting started. `https://developer.foursquare.com/docs/places-api/getting-started/`.

Foursquare (2021b). Our story. `https://foursquare.com/about/`.

Foursquare (2021c). Places api. `https://developer.foursquare.com/docs/places-api/`.

Foursquare (2021d). Search for venues. `https://developer.foursquare.com/docs/api-reference/venues/search/`.

Google (2021). Get started. `https://developers.google.com/maps/documentation/geocoding/start`.

Hastings, J. (2008). Automated conflation of digital gazetteer data. *International Journal of Geographical Information Science*, 22(10):1109–1127.

Hawkins, D. M. (2004). The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12.

Hitta (2021). Api documentation. `https://www.hitta.se/api`.

Hittapunktse AB (2021). Om oss. `https://www.hitta.se/info`.

Hossin, M. and Sulaiman, M. (2015). A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5(2):1.

Hssina, B., Merbouha, A., Ezzikouri, H., and Erritali, M. (2014). A comparative study of decision tree id3 and c4.5. *International Journal of Advanced Computer Science and Applications*, 4(2):13–19.

Levenshtein, V. I. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707–710.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 4765–4774. Curran Associates, Inc.

McKenzie, G., Janowicz, K., and Adams, B. (2014). A weighted multi-attribute method for matching user-generated points of interest. *Cartography and Geographic Information Science*, 41:125–137.

Molnar, C. (2019). *Interpretable Machine Learning*. Unpublished. `https://christophm.github.io/interpretable-ml-book/`.

Montavon, G., Binder, A., Lapuschkin, S., Samek, W., and Müller, K.-R. (2019). *Layer-Wise Relevance Propagation: An Overview*. Springer International Publishing, Cham.

Novack, T., Peters, R., and Zipf, A. (2018). Graph-based matching of points-of-interest from collaborative geo-datasets. *ISPRS International Journal of Geo-Information*, 7(3).

OpenStreetMap (2020). Export. `https://www.openstreetmap.org/export`.

OpenStreetMap (2021). About. `https://www.openstreetmap.org/about`.

Papadopoulos, S., Popescu, A., and Kompatsiaris, I. (2015). *Tourism Knowledge Discovery in Social Multimedia*, volume 3. IGI Global.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Piech, M., Smywinski-Pohl, A., Marcjan, R., and Siwik, L. (2020). Towards automatic points of interest matching. *ISPRS International Journal of Geo-Information*, 9(5).

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Ribeiro, M., Singh, S., and Guestrin, C. (2016). "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 97–101, San Diego, California. Association for Computational Linguistics.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition.

Scheffler, T., Schirru, R., and Lehmann, P. (2012). Matching points of interest from different social networking sites. In Glimm, B. and Krüger, A., editors, *KI 2012: Advances in Artificial Intelligence*, pages 245–248, Berlin, Heidelberg. Springer Berlin Heidelberg.

Tripadvisor (2021). List a business on tripadvisor. `https://www.tripadvisor.com/GetListedNew`.

Yu, F., West, G., Arnold, L., McMeekin, D., and Moncrieff, S. (2016). Automatic geospatial data conflation using semantic web technologies. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '16, New York, NY, USA. Association for Computing Machinery.

Appendices

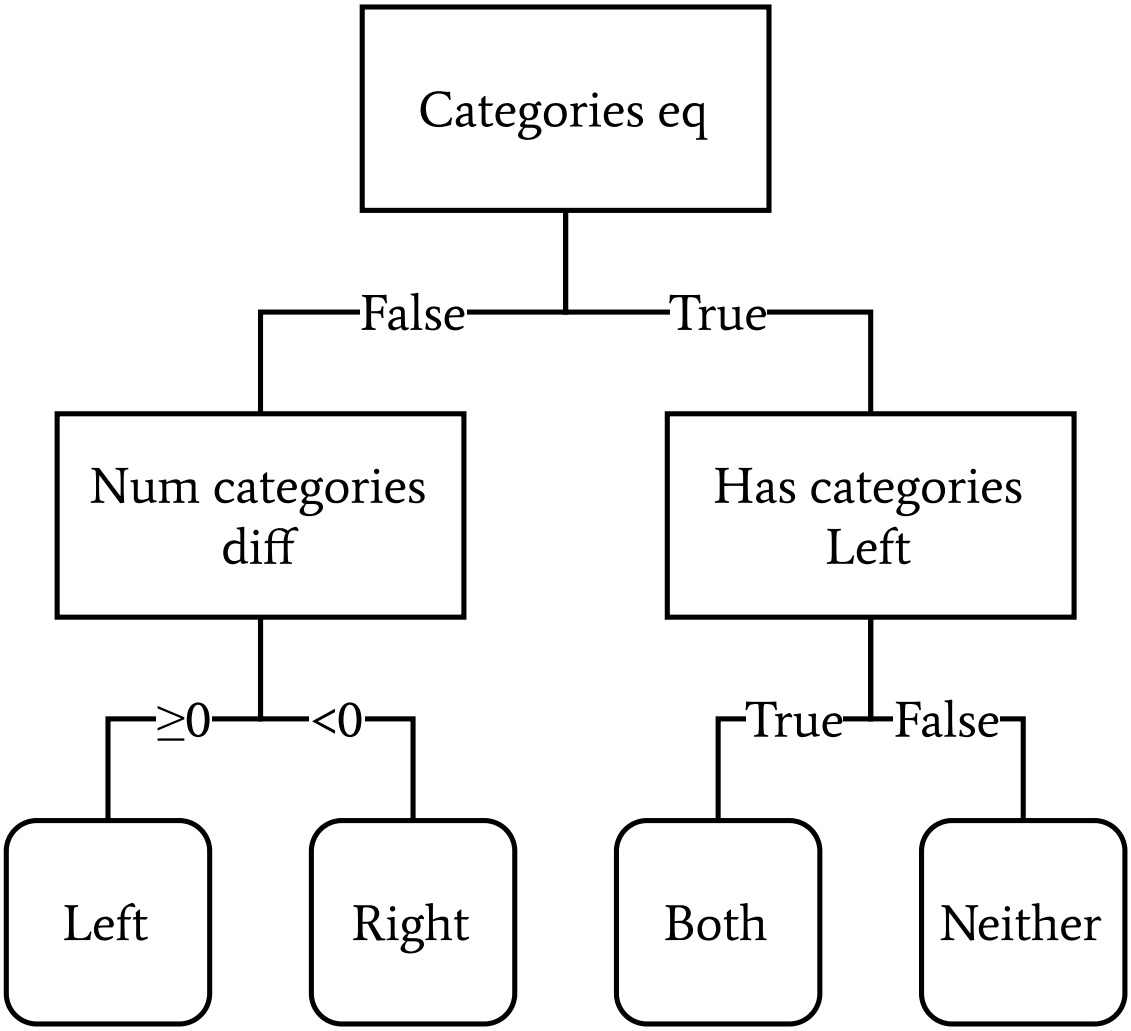

Figure 1: The decision tree graph of the model conflating the *categories* property.

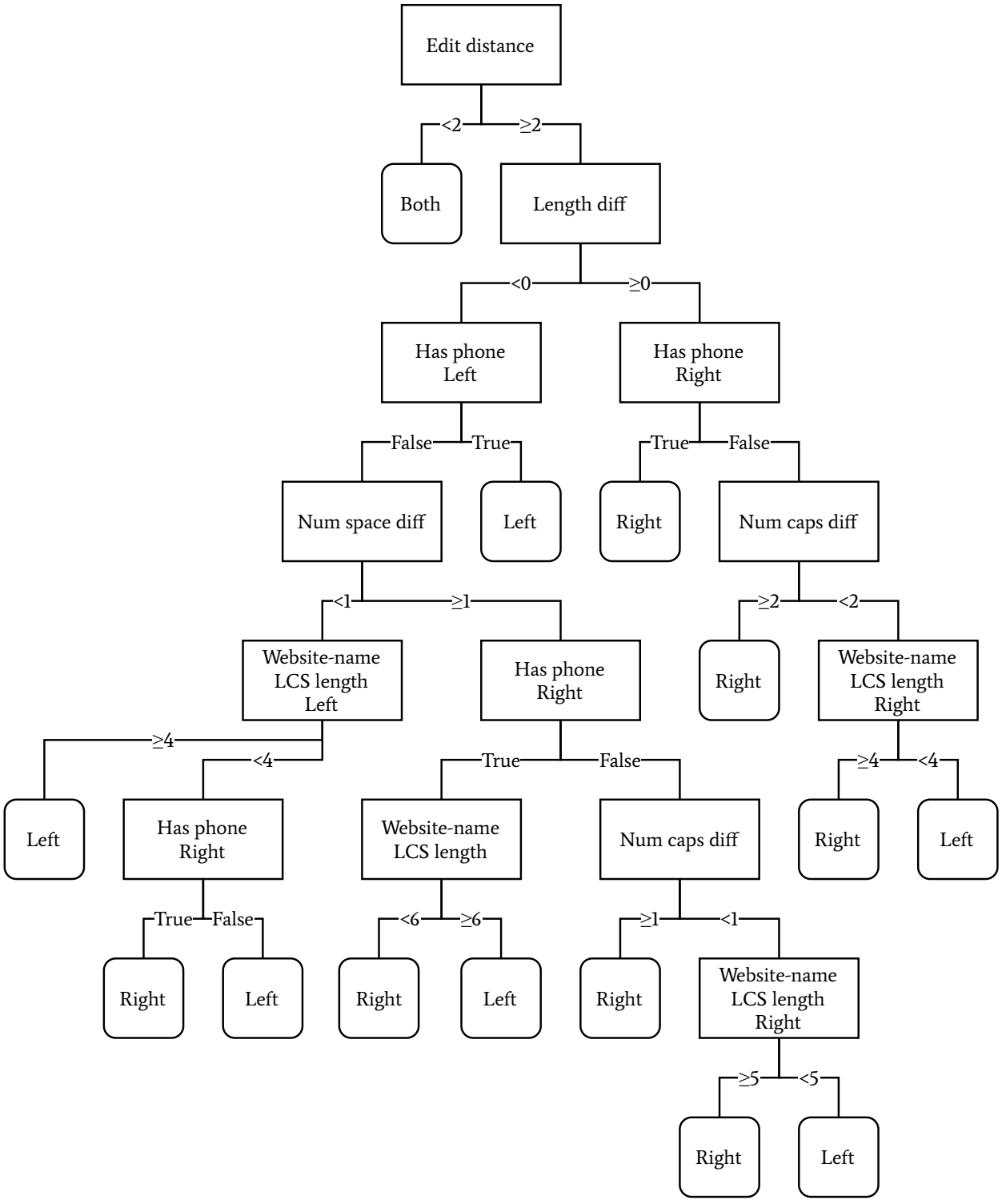

Figure 2: The decision tree graph of the model conflating the *name* property.

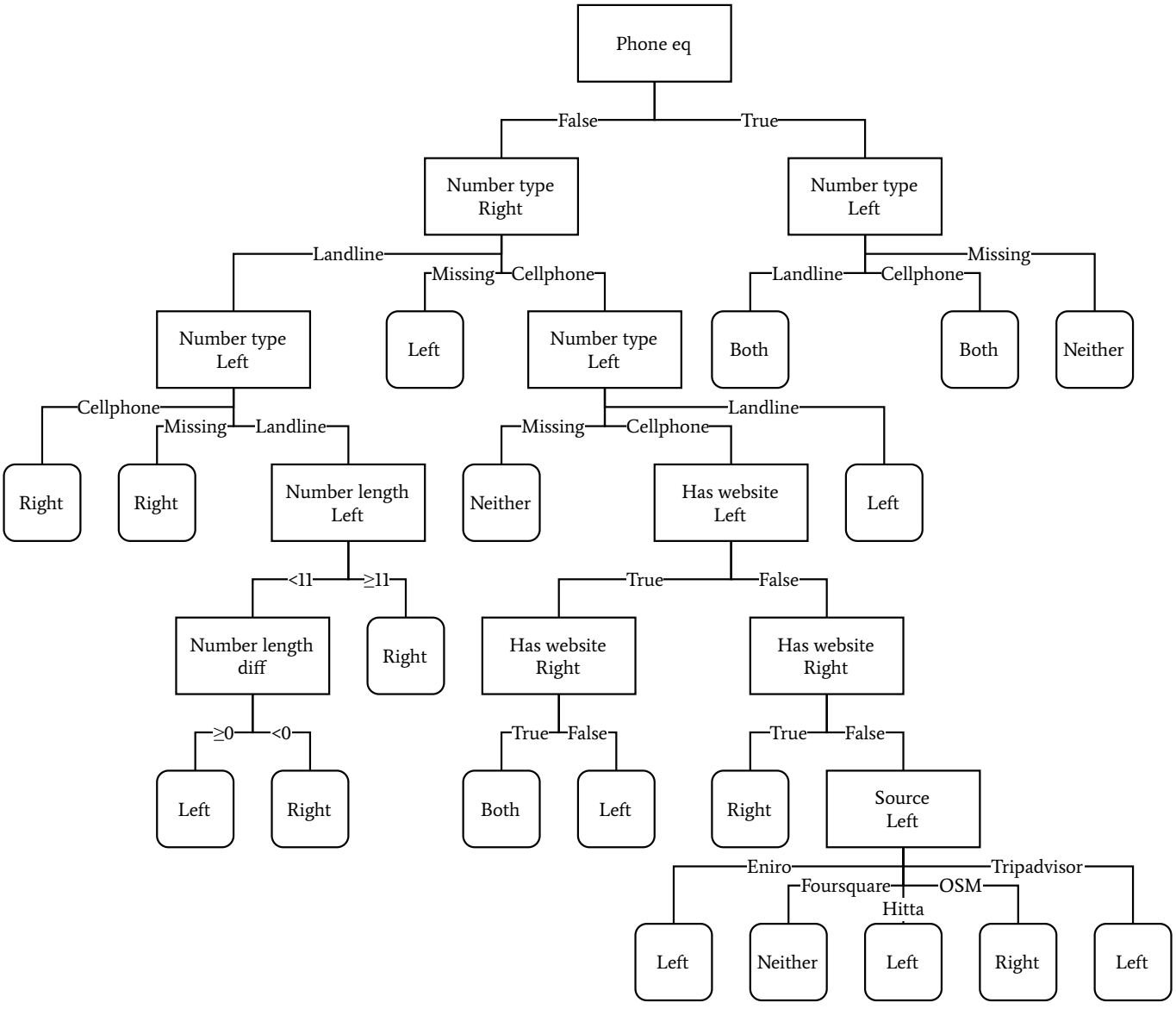

Figure 3: The decision tree graph of the model conflating the *phone* property.

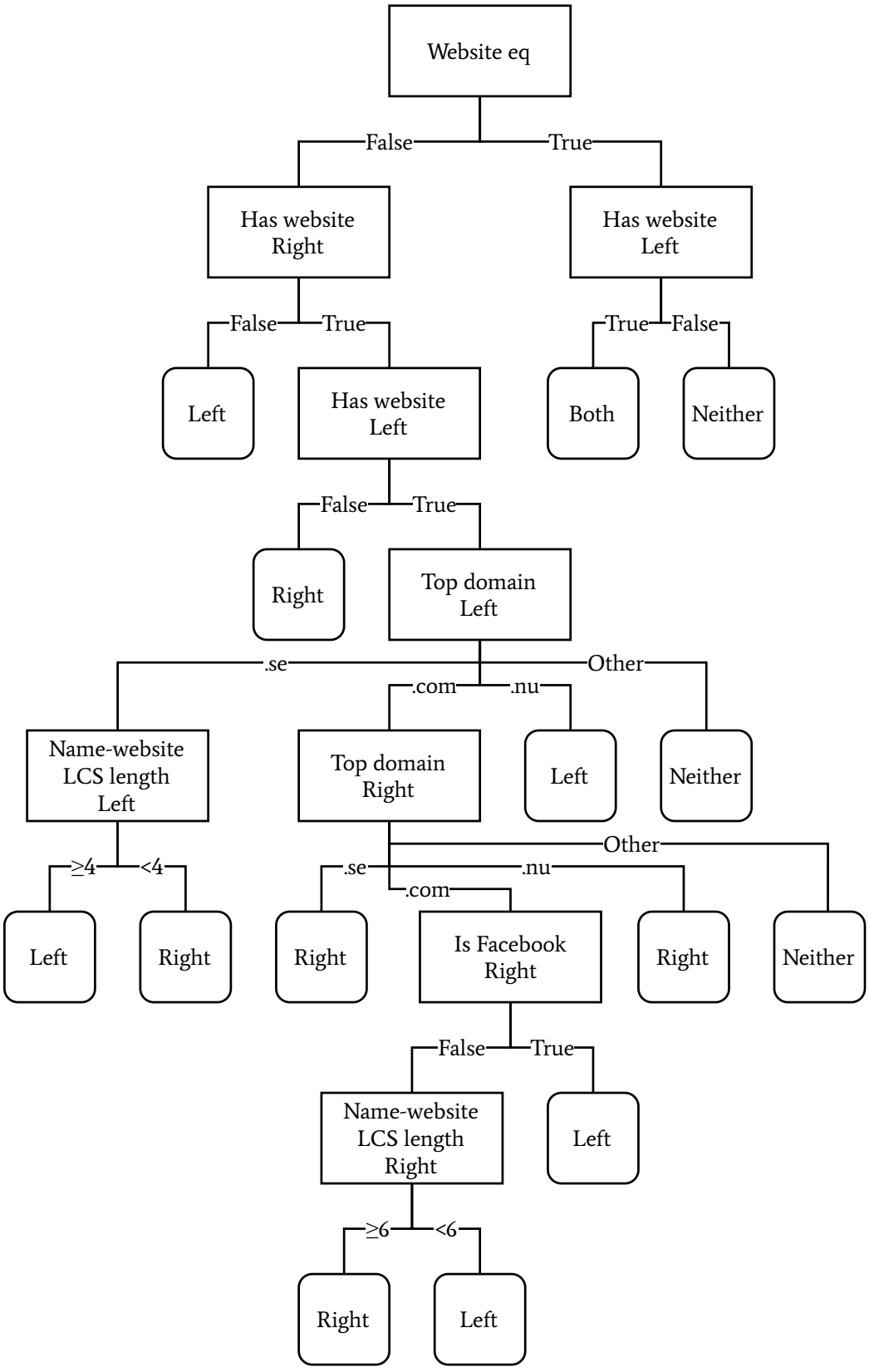

Figure 4: The decision tree graph of the model conflating the *website* property.

Feature name	Feature description
Categories eq	Whether both POIs contain the same *categories* property
Has categories *left*	Whether the POI in the *left* position has a non-empty *categories* property
Num categories diff	The difference in the amount of categories listed by the two *categories* properties

Table 1: Descriptions of the features used for the conflation of the *categories* property

Feature name	Feature description
Edit distance	The Levenshtein edit distance between the two *name* properties
Length diff	The length difference of the two *name* properties
Num caps diff	The difference in the amount of capitalised letter in the two *name* properties
Num space diff	The difference in the amount of spaces in the two *name* properties
Has phone *left*	Whether the POI in the *left* position has a non-empty *phone* property
Has phone *right*	Whether the POI in the *right* position has a non-empty *phone* property
Has website *left*	Whether the POI in the *left* position has a non-empty *website* property
Has website *right*	Whether the POI in the *right* position has a non-empty *website* property
Name-website LCS *left*	The length of the longest common string between the *name* and *website* properties of the *left* POI
Name-website LCS *right*	The length of the longest common string between the *name* and *website* properties of the *right* POI

Table 2: Descriptions of the features used for the conflation of the *name* property

Feature name	Feature description
Phone eq	Whether both POIs contain the same *phone* property
Source *left*	The source from which the POI-data of the *left* POI was collected
Source *right*	The source from which the POI-data of the *right* POI was collected
Number type *left*	Whether the phone number is a landline, cellphone or missing for the *left* POI
Number type *right*	Whether the phone number is a landline, cellphone, or missing for the *right* POI
Number length diff	The length difference of the two *phone* properties
Has website *left*	Whether the POI in the *left* position has a non-empty *website* property
Has website *right*	Whether the POI in the *right* position has a non-empty *website* property

Table 3: Descriptions of the features used for the conflation of the *phone* property

Feature name	Feature description
Website eq	Whether both POIs contain the same *website* property
Has website *left*	Whether the POI in the *left* position has a non-empty *website* property
Is facebook *left*	If the *website* property of the *left* POI is a Facebook URL
Is facebook *right*	If the *website* property of the *right* POI is a Facebook URL
Top domain *left*	The top domain of the *left* POI
Top domain *right*	The top domain of the *right* POI
Name-website LCS *left*	The length of the longest common string between the *name* and *website* properties of the *left* POI
Name-website LCS *right*	The length of the longest common string between the *name* and *website* properties of the *right* POI

Table 4: Descriptions of the features used for the conflation of the *website* property

EXAMENSARBETE Exploring the suitability of explainable AI for binary conflation of POI data
STUDENTER Emil Aminy, Marcus Lissner
HANDLEDARE Pierre Nugues (LTH), Frank Camara (AFRY)
EXAMINATOR Jacek Malec (LTH)

Undersökning av förklarbar AI för sammanslagning av POI-data

POPULÄRVETENSKAPLIG SAMMANFATTNING **Emil Aminy, Marcus Lissner**

En undersökning av hur effektivt förklarbar AI är för binär sammanslagning av POI-data samt hur användbar dess förklarbarhet är för denna uppgift.

Det finns idag många företag och tjänster som tillgodoser diverse kartdata. Det kan till exempel handla om vägar, vegetation, eller platser av intresse (POI). Det finns dock ingen garanti att data från en enstaka leverantör är helt korrekt, fel kan alltid uppstå. En metod för att korrigera sådana fel är sammanslagning; kombinering av data från flera källor för att mitigera eventuella fel.

Att uföra en sådan process på ett vis där slutprodukten är mer korrekt än alla dess delar är dock inte helt trivial. Någon form av beslutsprocess måste existera som kan avgöra vilken data från vilken källa kan anses vara av bättre kvalité. Ett väl lämpat verktyg för denna process är maskininlärning. Men, om detta moment skulle genomföras med avancerade maskininlärningsalgoritmer, såsom neurala nätverk, kan algortimens resonemang kring datasammanslagningen bli väldigt svårtolkad för en människa.

Att förstå sig på varför en algoritm gör som den gör kan vara användbart för vidareutvecklingen av algoritmen. Dessutom kan insikten ge människor förtroende i att algoritmen utför sitt jobb korrekt. Det är därför som vi i detta examensarbete har tittat på hur förklarbar AI (XAI) kan användas för att råda bot på detta insynsproblem. Vårt fokus ligger specifikt på XAIs användbarhet vid sammanslagningen av POI-data. Vi vill ta reda på hur pass jämförbar en XAI-lösning för just denna uppgift är, både i förhållande till en mindre förklarbar lösning men även jämfört med en människa. Vi vill även undersöka hur användbar och effektiv förklarbarheten är.

Vårt experiment gick ut på att utföra binär sammanslagning av POI-data med hjälp av ett beslutsträd och jämföra dess resultat med ett neuralt nätverk och mänsklig prestanda. POI-data för detta experiment insamlades från fem källor: Eniro, Foursquare, Hitta, OpenStreetMap, och Tripadvisor. Data begränsades endast till POI:ar inom Sverige som innehöll information om hotell- och restaurangverksamheter.

Resultaten pekar mot att beslutsträd är jämförbara med både neurala nätverk och människor för denna uppgift. Deras förklarbarhet är dessutom väldigt användbar vid utvecklingsstadiet då insynen tillåter snabb evaluering och förbättring av skapade modeller.