Information retrieval and deduplication for tourism recommender Sightsplanner

Ago Luberg
Eliko Competence Center
Teaduspargi 6/2, 12618
Tallinn, Estonia
Tallinn University of Technology
Ehitajate tee 5, 19086
Tallinn, Estonia
ago.luberg@eliko.ee

Michael Granitzer
Prof. for Media Informatics
University of Passau
Passau, Germany
Michael.Granitzer@uni-passau.de

Honghan Wu
School of Computer and Software, Nanjing University of Information Science & Technology, China
Eliko Competence Centre
Teaduspargi 6/2, 12618
Tallinn, Estonia
honghan.wu@gmail.com

Priit Järv
Eliko Competence Center
Teaduspargi 6/2, 12618
Tallinn, Estonia
Tallinn University of Technology
Ehitajate tee 5, 19086
Tallinn, Estonia
priit@cc.ttu.ee

Tanel Tammet
Eliko Competence Center
Teaduspargi 6/2, 12618
Tallinn, Estonia
Tallinn University of Technology
Ehitajate tee 5, 19086
Tallinn, Estonia
tammet@staff.ttu.ee

ABSTRACT

This paper is about scraping web pages for tourism objects and resolving duplicates for a tourism recommender system Sightsplanner. Gathering information from different web portals, we end up having several versions of the same object in our database. It is very important that we can find out which objects are duplicates and merge those. Only unique objects are presented to the end user. The main focus of this paper is therefore on deduplication problem. We have implemented a duplication detection system and tuned the parameters manually to get up to 85% accuracy. In this paper we present a machine learning setup which we used to improve deduplication accuracy of tourism attractions by 13 percentage points to achieve 98% accuracy. All the steps in the process are presented along with problems we tackled.

Categories and Subject Descriptors

H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval

General Terms

Experimentation, Measurement

WIMS'12, June 13-15, 2012 Craiova, Romania

1. INTRODUCTION

Internet access has made tourism object information available for tourists. A tourist can find several interesting objects she wants to visit during her trip. With all this information, the tourist could have a problem that she cannot find out the best objects for her. We introduce a tourism recommender system named Sightsplanner[1], which helps tourists to find personalized suggestions depending on their interests. In this paper we focus on information retrieval from the Internet. Gathering information from different web portals produces duplicate objects. In this paper we also present how to deal with tourism object deduplication.

The contributions of the paper include:

- Empirical feature study for disambiguating non-frequent geo-objects scraped from different web portals.
- Evaluation of training set selection heuristics in order to tackle highly unbalanced training set in disambiguation problems.
- Optimization of distance functions with machine learning techniques for disambiguation geo-entities.

Our evaluation shows, that we can improve the disambiguation accuracy from a manually tuned similarity measure by 13 percent points to 98% accuracy. Hence, our work shows that machine learning based methods have to be favoured over manually tuning efforts in term of effectiveness and efficiency.

This paper is organised as follows. In section 2 we will give a short overview of the whole recommender system. In section 3 we describe data gathering techniques and give some

[1]Tallinn Sightsplanner, see *http://tallinn.sightsplanner.com/*

Figure 1: GUI first page

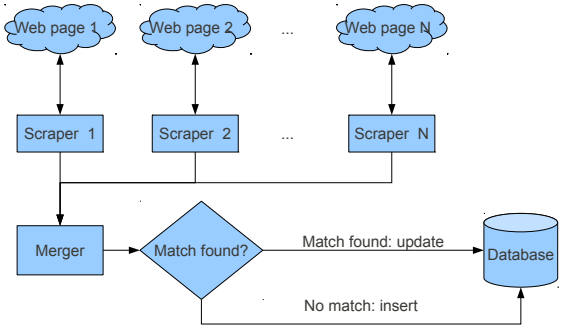

Figure 2: Data scraping

examples about the tourist object data. We will continue with deduplication overview in section 4. In section 5 we present our work on improving deduplication using machine learning. Finally, we give a short overview of related work in section 6, draw a conclusion and discuss future work in section 7.

2. RECOMMENDER SYSTEM

Sightsplanner is a semantic recommender and route composer system for tourists. A tourist can specify her location, time and duration of the visit and her preferences about different types of objects and events. Based on the created tourist profile, the recommender identifies interesting objects for the given user. For each found object a ranking score is found. The objects that the user probably likes have a higher score and vice versa. A planning mechanism organizes the objects and events into a trip timetable. In an interactive feedback cycle, the tourist has the option to modify the suggested trip. To calculate the final list of recommended objects, the following processes are involved: (a) Object verification process, (b) Matching process, (c) Planning process, (d) Result representation process, and (e) Feedback process.

The recommendation process starts when the user opens the web page and defines her interests by using a slider-based approach for adjusting individual preferences. In Figure 1 the first page of the user interface is shown. The tourist can indicate her interest in seven main topics moving the corresponding sliders. Each of them also has subtopics. In the figure the "Events" slider is "opened" and its subsliders are shown. The tourist has stated that she likes "Museums and arts" and "Architecture and city" very much. She also likes some "Eating out", "Shopping" and "Events". Especially she likes "Jazz" and "Rock concerts". Every topic in the user profile matches a type in tourism objects' properties. The user can also specify the start date and time of her visit, visit duration and preferred travelling method.

After the tourist has selected her interests, she starts the overall recommendation process. The user profile is sent to the planner, which is responsible of returning a personalised recommendation for the given profile. The planner uses all the relevant data from the memory database. All the objects in the requested city which also are opened during user's visit will be processed. Based on the object types, location, opening time and some other properties, different trip timetables are created for the user. The planner has to return the best trip it has found after a certain time limit - this is then presented to the user as a timetable and as a map.

The recommended plan can be modified by the user starting an interactive feedback cycle: she can remove some objects or change visit time for objects and re-run the recommendation based on the performed changes: after the modification, the planner takes the changes into account and a new recommendation is created for the user.

3. DATA ACQUISITION

In order to provide accurate recommendations, tourism objects like restaurants and their properties like opening times have to be scraped from web pages and harmonized with already existing data regularly. For Tallinn Sightsplanner, we are scraping six different sites all with different data structure. All those portals are regular web pages. For every data source we have manually described the page structure for our scraper, which then normalizes data into our custom ontology. Gathered information can be divided into more dynamic and more static objects. One-time events (concerts, performances etc.) are more rapidly changing, therefore we are scraping those daily to get changes and new events. Other type of objects are places of interest (POI) with mostly static information - nothing changes in months or even years. POI objects are updated typically once a month.

A brief overview of the architecture of our importer is presented in the Figure 2. The importer can connect to several web pages using manually created scraper algorithms. Each scraper downloads the content of the web pages, finds the necessary information (for example using XPath[2] to extract title, description etc.), normalizes data for our system and sends it to the Merger. We will describe the Merger functionality more in the section 4.

A tourism object, or simply an object, has several facts and zero or more child objects. Every fact is described with following fields: *property*, *value*, *language*, *datatype*, *score*, *source*, *timestamp*. *property* shows what this fact describes (e.g. "title", "address", "phone"). Fact's *value* can be stored in different *languages* and *datatypes* (e.g. number, string, date). A fact also has a confidence or probability score in range $[0; 1]$, where 1 indicates a certain fact and lower value

[2]XML Path Language, see *http://www.w3.org/TR/xpath/*

lowers the certainty of the fact (e.g. 0.7 means that the given fact is true with a probability 0.7). *source* indicates the datasource of the given fact, *timestamp* is date and time when the fact was written.

An object can have child objects. Every child object has the same fields as described previously. In our system we use child objects to describe opening times. One opening time consists of several facts, for example weekday (object is opened on Wednesdays), start time (from 10 AM) and end time (to 7 PM).

Definition 1. The set of all the tourism objects is $\mathbf{O} = \{O_1, O_2, \ldots, O_n\}$.

Possible property URIs are defined by our custom ontology. An object can have several facts with the same property (different titles in different languages etc.). Some of the most important properties for calculating recommendation are "#latitude", "#longitude" (location of the object), "#opening_time" (when is the object opened), "#type" (to match the object against user interests), "#popularity_tourist" (to favour more popular objects), "#visit_time" (the time suggested to spend on sight). Some other properties are important for presenting the object to the user. The most important ones are "#title" and "#description", which will be stored in different languages. An object can have a different title in English and in Estonian. Different values are presented to the user depending on their user interface language. Often web pages do not provide data in separate languages. We try to use automatic translation via Google Translate API [3] to fill in the missing values. We make use of the *score* field and have lower value for automatically translated texts. Usually scraped information from the source web page gets a score 1.0. For automatically translated title and description we will use lower score value (e.g. 0.5). Later, when we find the same object from another website, where missing information is available, we can prefer a title fact with higher score over the automatically translated one.

To recommend an object to the user, the object needs to have the same category that the user has marked as her interest. Often data sources do not define the type or the category of the object. Or categories they are using do not match with our set of categories. To overcome this problem, we are using keyword extraction from the description field in different languages to find keywords which could be mapped into a category known for us. For every language we have a list of keywords along with score values. The score value will be applied to the created category fact. For example, if the system finds a keyword "painting" in the text and in our keyword list the "painting" keyword is mapped to "#art" category with a score 0.6, then the object will get a fact, which says that the object has a category "#art" with a confidence score 0.6. The score for category facts can be also seen as a strength or a weight. More details about keyword extraction and data in general is presented in [4].

4. DEDUPLICATION

Scarping information from different sources yields in duplicate objects as most of the web pages do not provide global URI, which could be used to match the same resources from different sources. From the recommendation point of view, having duplicate objects in the proposed schedule reduces the quality of our system. Often sources do not have full information about an object. Different source may have some information about the object which was missing in the first source etc. So, merging duplicate objects may improve the quality of the merged object. Taken into account that deduplication will improve our recommendations and object information quality, we were motivated to work on the solution.

First, we will define some notations which we will be using later.

Definition 2. Object A_i is a duplicate of object A_j if they represent the same pyhsical object. Let d be a symmetric function which returns 1 if all its arguments are duplicates, 0 otherwise:

$$d(A_1, \ldots, A_n) = \begin{cases} 1 & \text{if all } A_i \text{ are duplicates} \\ 0 & \text{otherwise} \end{cases}$$

In addition to comparing different objects, in our notation object is a duplicate of itself: $d(A, A) = 1$ or $d(A) = 1$.

Definition 3. A group of duplicates, called a *cluster*, is a set C which consists of at least one object from all the objects $\mathbf{O}$ so that all the included objects are duplicates:

$$C = \{A_1, \ldots, A_n | d(A_1, \ldots, A_n) = 1\}$$

A cluster could also consist of only one object $C = \{A\}$, as $D(A, A) = 1$.

Definition 4. A maximal group of duplicates, called a *maximal cluster*, is a cluster which cannot accept any new objects so that all the objects would be duplicates of each other (there is no additional object which is a duplicate of the objects in the cluster).

In this paper, we are interested in *maximal clusters*. Therefore we use term *cluster* to denote *maximal clusters* if not noted differently.

Definition 5. A similarity between two objects A and B is defined by function S which is weighted average over similarity values:

$$S(A, B) = \frac{\sum_i w_i * sim_i(A, B)}{\sum_i w_i}$$

sim_i is a function which compares certain property or properties of two objects and returns a similarity score in range [0; 1]
w_i is a weight value for the similarity function sim_i.
The value of S will be in range [0; 1].

Definition 6. Two objects are duplicates by similarity function if the similarity function S between the objects A and B exceeds a threshold T:

$$S(A, B) \geq T \Rightarrow d(A, B) = 1$$

In the data acquisition section we gave an overview of the data importing process. We also mentioned the Merger component, which deals with deduplication. Every scraped object is sent to the Merger which compares the new object with the existing ones by computing the similarity value between the objects (using title similarity, location similarity

[3]Google Language API Family, see *http://code.google.com/apis/language/*

etc.). From certain similarity value threshold, the new object is considered to be a duplicate of the found existing object(s). In the case of match, the existing object in the database is updated with new scraped data. If there is no matching object in the database, the scraped object is added as a new object. More details about the merging and Merger implementation are presented in [5].

For our current system, we have manually tuned weights w_i for similarity functions sim_i and the threshold T value. Our best setup had equal weights for every similarity function except for the distance similarity, which had double importance. We used total of 6 different similarity functions. The threshold value which indicated the separation of duplicate and non-duplicate object pair was 0.9. Using this setup, we were able to get F-score 0.85. Our goal was to improve this metrics at least above 0.9. That is where we started using machine learning.

5. EXPERIMENTS

Instead of manually trying to adjust the weight parameters of similarity function in definition 5, we have used machine learning to find the best settings. In this section, we will present the setup of the whole process along with the results. The section is divided into subsections about learning problem, data used for experiments, feature selection, sample selection, learning setup and results.

5.1 Learning problem definition

Instead of trying to group objects directly, we have a different approach. We try to learn whether two objects are duplicates or not based on the property similarity functions. Every sample in our learning set is a set of similarity values calculated by comparing two objects. That is the reason, why we presented object pair counts in Table 1. We want to learn how to separate positive and negative pairs (e.g. whether a pair of objects represent the same physical object or not). If we have 4 objects A, B, C, D, we can have 6 unique samples: comparisons of pairs AB, AC, AD, BC, BD and CD.

Definition 7. Given the similarity function $S(A, B)$ (definition 5) we define $\mathbf{f_{A,B}}$ as a feature vector with f_i being the evaluation of sim_i on objects A, B.

$$\mathbf{f_{A,B}} = \langle f_1, f_2, \ldots, f_n \rangle$$

where

$$f_i = sim_i(A, B)$$

sim_i may be chosen from a set of similarity functions Sim.

A sample is given by $\mathbf{x_{A,B}} = \mathbf{f_{A,B}}$
A sample can be also called $\mathbf{x_i}$ if the compared objects are not known or not important.

A sample set
$X = \{\mathbf{x_{A_i,A_j}} | A_i, A_j \in \mathbf{O}\}$ or
$X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$, where m is the number of samples.

A classification label is given by

$$y_{A,B} = \begin{cases} 1 & \text{if } d(A, B) = 1 \\ 0 & \text{if } d(A, B) = 0 \end{cases}$$

A label set
$Y = \{y_{A_i,A_j} | (A_i, A_j \in \mathbf{O}), (\mathbf{x_{A_i,A_j}} \in X)\}$ or
$Y = \{y_1, y_2, \ldots, y_m\}$, where m is the number of samples. Label set Y has labels for the same object pairs which are present in sample set X in the same order.

A sample $\mathbf{x_{A,B}}$ is called a positive sample if $y_{A,B} = 1$, e.g. object A and B are duplicates. If $y_{A,B} = 0$, then the sample $\mathbf{x_{A,B}}$ is called negative sample.

A learning function is given by $\lambda = \langle L, X, Y \rangle$, where L defines a learning algorithm along with parameter values, X is a set of samples, Y is a set of labels for the samples X.
Learning function λ returns a function p which predicts with some accuracy whether given two objects A and B are duplicates or not.
The goal of λ is to find a function p which would yield in highest prediction accuracy.

Our learning problem is defined to learn, whether given two objects are duplicates or not. One sample (which can used for training and also for testing) is a set of similarity values between two objects. Every similarity value indicates a similarity of a certain property or properties between two objects. For example, let us use two features: title similarity and description similarity. Every sample in our training/testing dataset would have 2 values. If we have objects A and B, which are very similar, then the sample can have values $\mathbf{x_{A,B}} = 1.0, 0.9$. Another pair of objects A and C might not be similar and have values $\mathbf{x_{A,C}} = 0.3, 0.1$. Both those samples have also a label value: $y_{A,B} = 1$, $y_{A,C} = 0$. If the system would have only those two samples for training, it might learn that high feature values will yield in label 1 and vice versa. This is actually correct, because our features all have built this way that similar objects (near-duplicates or duplicates) have high values.

We use different sample and label sets for training and testing. After we have used some samples and labels for training, we want to test how well we have trained our model. For example, we have samples $\mathbf{x_{B,C}} = 0.5, 0.9$ and a label $y_{B,C} = 0$. After we have applied learning algorithm to our training set, we have got our deduplication function p. As we saw before, high feature values seem to indicate that two objects are duplicates. It may happen, that our function predicts $d(B, C) = 1$. The real label is $y(B, C) = 0$. We can see, that we made an error and may-be we did not learn the best possible deduplication function. Depending on the setup, we can just accept that we have made an error or we can try to train a better model. More about the learning setup will be presented in one of the next subsections.

5.2 Data for learning

Our initial Tallinn dataset had objects from different categories. Some data sources only provide eating places, some only events and so on. The dataset only has a small number of duplicate objects and maximum number of different data sources for one objects is three (there are three sources which provide eating places, so the same restaurant can be scraped from three sites). Therefore we have scraped a separate dataset for our learning setup. The dataset consists of Tallinn eating places from five different web portals. Some eating places are present in every data source, so after merging, the objects will be merged from five different sources. In addition to Tallinn dataset, we are using Riga tourism objects to test our learning model. The Riga dataset is not limited to eating places, there are also museums, galleries etc. Testing our trained model on Riga dataset gives us in-

formation whether our solution of duplicate detection can be applied for cross-city and cross-category datasets. Riga dataset is scraped from two different sources.

To be able to train and evaluate learning algorithms, we have manually annotated all the duplicate pairs. We have created a simple web interface which allows us to manually compare every objects with another object. Based on the information available, a user was able to state whether two objects are physically the same or not. Tallinn data consists of about 1800 scraped object, comparing every object with every other object would yield in 1.6M comparisons. To limit this number, we have added a duplicate candidate selection, where only a certain number of best matching objects are presented to the user. Objects, which are too far away from each other most likely are not duplicates. We also filter objects by type and data source. Data source filter means that we compare an object only with objects from different data sources. Our initial presumption was that a data source does not have duplicates within their data. It comes out, that actually there are duplicates within certain sources, but this is not a problem. After we have found that two compared objects are duplicates, then the connection is bidirectional (if object A is a duplicate of object B, then object B is a duplicate of object A). Every group of duplicate objects forms a complete graph (if object A is a duplicate of object C and object B is a duplicate of object C, then object A is also a duplicate of object B). Because of the completeness, it is usually enough to compare an object only to objects which do not have the same source. With filtering, every object had average about 20 possible candidates, which narrowed down the comparison space about 100 times.

Manual annotation still raised many questions amongst users who had to find duplicate pairs. Even if you are local and know most of the tourism objects, there are still cases, which cannot be solved with 100% confidence. Also, as we mentioned, we used candidate selection, which might have left some duplicates out. Objects, which were not in the candidate list, were not checked by the annotator. Altogether we believe that the error of manual deduplication can be about 3-5%. We will take this into account when we later evaluate our results.

Table 1: Statistics about the dataset for Tallinn and Riga

Property	Tallinn	Riga
Object count	1808	3839
Different sources	5	2
Non duplicates	478	3762
2-object groups (object count)	203 (406)	75 (150)
3-object groups (object count)	133 (399)	1 (3)
4-object groups (object count)	68 (272)	-
5-object groups (object count)	43 (215)	-
6-object groups (object count)	5 (30)	-
8-object groups (object count)	1 (8)	-
Positive (duplicate) pairs	1543	78
Negative (non-duplicate) pairs	1.6M	7M
Positive pair %	0.1 %	0.001 %

We have presented an overview of the dataset in Table 1. For Tallinn dataset, we have total of 1808 scraped objects from 5 data sources. 478 objects did not have any duplicate objects (or we could say they form up 478 clusters each consisting of only one object), 406 objects formed 203 groups with 2 duplicates in each cluster, etc. As can be seen, some objects have duplicate entries in the same data source. For example, in Tallinn dataset there is one cluster which is merged from 8 initial objects (duplicate object was present once in 2 data sources and twice in 3 data sources). Total number of unique duplicate pairs (graph undirected edges) is 1543. If object A is a duplicate of object B, then it is counted only once - pair object B is a duplicate of object A is not counted. All other possible unique pairs between the objects are non-duplicate pairs. For Tallinn data, there are about 1.6M non-duplicate pairs. We have also presented a percentage of duplicate pairs to non-duplicate pairs to indicate the balance of our dataset. As can be seen, for Riga the percentage is even worse.

5.3 Feature selection

We have implemented about 20 different functions for features. The most interesting for this paper are:

- Title comparison using Levenshtein distance[4];
- Custom title comparison with weighted words (common words weigh less and therefore do not change the outcome too much);
- Custom title comparison, which we will describe below;
- Custom title comparison with weighted words (common words weigh less and therefore do not change the outcome too much);
- Euclidean distance using originally scraped coordinates;
- Address string comparison;
- Euclidean distance using coordinates which were calculated from address strings.

Custom title comparison works as follows. The title is split into words. For every word a match from other title's words are found. Every match gives a positive score. All the words, which do not have a match in other title, will give negative score. Both scores are added together and normalized (to get a result within the range [0, 1]) and the result is the similarity between object titles. To find a match for a word, the words do not have to match exactly. A certain number of symbols can be different depending on the length of the word. If all the words from one title are found in the second title, then similarity is near to 1 (for example "papa pizza" vs "papa pizza with some extra words" will yield in almost total similarity, as additional words do not make the match worse). As we mentioned earlier, an object can have different titles in different languages. All the languages are compared and the best match is returned. The same logic is applied with weighted words. More common words have less impact on both positive (in case of match) and negative (in case of no match) score. If we compare titles like "Papa pizza" and "Mama pizza", then "pizza" is common word and having a common word in both titles does not make objects similar (positive match will be low). In the previous example, words "mama" and "papa" are probably

[4]Levenshtein distance, edit distance, see *http://en.wikipedia.org/wiki/Levenshtein_distance*

not so popular, which makes the negative match high. In this example, objects are not similar. But if we consider titles like "Papa pizza" and "Papa restaurant". "Papa" is not very popular name, so the positive match is high. "Pizza" and "restaurant" are usually very popular, which makes the negative score low. Based on the weighted title comparison, those two objects could represent the same physical object.

To give a short example of custom title comparison, let us consider two titles "Tallinn city hall" and "Tallinn city pharmacy". There are two words, which are present in both titles: "Tallinn" and "city". Both titles have one additional different word ("hall" and "pharmacy"). If we do not use weights, we find an average non-matching word count for both titles. In this example the average is 1. The similarity between two titles is calculated as follows: *matching_count* / (*matching_count* + *avg_non_matching_count*). In our example, it would be 2 / 3 = 0.66. If we consider titles like "papa pizza" and "papa pizza in shopping center", then matching word count is 2. We also have 3 non-matching words. All the non-matching words which are present due to title length difference (if one title has 2 words less than the other, then those 2 words usually are also non-matching) have lower penalty. In the current example, the calculation can be for example: 2 / 2.6 = 0.77. The lower penalty depends on the count of non-matching words and on the length of the titles. If the titles are long, then the penalty will get lower (if you have one non-matching word for 10 words title, then penalty will be near to 0).

In addition to mentioned custom title comparison, we also have so called custom title comparison with join. Regular custom title comparison splits the title into words and starts comparing. The comparison with join tries to combine different words into one and run the custom title comparison then. For example, if we have titles "McDonalds" and "Mc Donalds", then the regular custom title comparison would return 0 as there is no matching words, whereas after joining two words in the second name we would compare the same titles and the result will be high score. The join version of comparison basically finds all the combination of joining consecutive words on both titles and for every combination, the regular custom title comparison is run. This makes the join version several times slower.

To find out, which combination of features is the best, we have done training and testing with all the possible combinations up to 5 features. We will present results soon. First we have to discuss the sample selection.

5.4 Sample selection

As we presented in Table 1, the possible number of object pairs is for Tallinn data about 1.6M. The table also shows that only 0.1% of those positive samples (duplicate pairs). If we would take all the samples, then we would have several problems:

- Generating a feature value set for 1.6M pairs takes time;
- Learning with large number of data takes a lot of time;
- The balance between positive and negative samples is heavily skewed.

One of our goal is to use as few samples as possible. Therefore we did not try to use all the samples. Instead, we aimed for 10 000 samples for Tallinn data. If we used random sample selection, we would end up only about 15 positive training samples, which is obviously too few.

If we think about the possible samples in our dataset, then for one object, there are about 1800 samples available (comparison with every other object). Feature like objects distance will have similarity value 0 or very low when the distance is more than 500 meters for example. Most of the objects are farther than this. For distance features, maximum 100 objects would give similarity score about 0. The same is usually the problem with titles. For example, an object with title "McDonalds" do not have many matches amongst the whole dataset. If we would take all 1.6M negative samples, then many will end up having all the feature values near to zero. We do not need to include all those for our learning dataset. Instead, we are more interested on negative pairs, which are closer to duplicates. With positive samples, we do not filter anything out - we will use all 1543 positive samples for learning.

Our sample selection for Tallinn data currently has 10 000 samples, 1543 of those are positives, and we try to take mostly negative pairs which have high feature values (near-duplicates). Of course, we cannot leave out the low negative samples (all feature values near to zero), otherwise we might end up with classification which only recognizes mid-values as non-duplicates. About 1000 samples have close to zero feature values. For Riga dataset, we have taken more samples. We have just limited the number of samples with feature values close to zero to 10 000. Riga data is used only for testing and it has about 100 000 samples: 78 positive ones, about 10 000 low negative ones, the rest is mid-valued or near-duplicate negative samples.

5.5 Learning setup

We have used Python software scikit-learn[5] to assist our learning process. The software supports various number of different learning algorithms. For our problem, we have used SVM (Support Vectore Machine) classification and decision trees. For SVM, we used grid search, which tries several different parameters and returns the one with the best results. The grid search tries both linear and radial basis function (RBF) as a kernel. For decision trees, we are using extra trees which train several (30 in our case) independent models randomly and uses average over the models to predict. B learning algorithms are run with all the combination of all the features. This way we can find out the most important features. We also would like to minimize the calculation costs for prediction, therefore we try to minimize the number of features necessary for a model.

As mentioned earlier, we use Tallinn data for training and testing, Riga data is used only for evaluation. Tallinn dataset is divided into two equal sized parts where the ratio of positive and negative samples also remains the same. One part is for development and the other is for evaluation. On development part, we do training with 10-fold cross-validation. The model which yields in best results on cross-validation, will be used for evaluation both with remaining Tallinn data and with Riga data.

5.6 Learning results

We have constructed several different datasets (different number of samples and different selection of samples) which

[5]scikit-learn: machine learning in Python, see *http://scikit-learn.sourceforge.net/stable/*

Table 2: Learning results using one feature

Feature (code)	Algorithm	Tallinn			Riga		
		precision	recall	f-score	precision	recall	f-score
Address string comparison (ADD)	rbf	0.92	0.94	0.93	0.52	0.91	0.66
	extree	0.96	0.92	0.94	0.59	0.88	0.71
Title comparison without joining words (T1)	rbf	0.99	0.85	0.91	0.76	0.91	0.83
	extree	0.99	0.85	0.91	0.65	0.91	0.76
Title comparison with joining words (T2)	rbf	0.98	0.89	0.94	0.69	0.96	0.80
	extree	0.98	0.92	0.95	0.52	0.97	0.68
Title comparison with edit distance (ED)	rbf	0.96	0.85	0.90	0.09	0.94	0.16
	extree	0.97	0.86	0.91	0.14	0.91	0.24
Distance with original source coordinates (OD)	rbf	0.76	0.67	0.71	0.65	0.67	0.66
	extree	0.84	0.82	0.83	0.36	0.90	0.51
Distance with re-calculated coordinates (RD)	linear	0.93	0.95	0.94	0.55	0.96	0.70
	extree	0.96	0.95	0.95	0.71	0.88	0.79
Title comparison without joining words, with word weights (TW1)	linear	0.97	0.90	0.94	0.36	0.99	0.53
	extree	0.97	0.96	0.96	0.27	0.99	0.43
Title comparison with joining words, with word weights (TW2)	linear	0.97	0.94	0.95	0.28	1.00	0.43
	extree	0.98	0.97	0.98	0.28	0.99	0.44

Table 3: Learning results using two features

Feature codes	Algorithm	Tallinn			Riga		
		precision	recall	f-score	precision	recall	f-score
ADD + T1	rbf	0.99	0.95	0.97	0.93	0.99	0.96
	extree	0.99	0.97	0.98	0.73	0.99	0.84
T1 + RD	linear	0.98	0.97	0.98	0.93	1.00	0.96
	extree	1.00	0.97	0.98	0.93	0.99	0.96
RD + TW1	linear	0.97	0.99	0.98	0.71	1.00	0.83
	extree	1.00	0.98	0.99	0.88	1.00	0.93
RD + TW2	linear	0.99	0.97	0.98	0.80	1.00	0.89
	extree	1.00	0.99	0.99	0.89	1.00	0.94
T2 + RD	rbf	0.98	0.99	0.98	0.86	1.00	0.92
	extree	1.00	0.98	0.99	0.94	0.99	0.96

Table 4: Learning results using two features

Feature codes	Algorithm	Tallinn			Riga		
		precision	recall	f-score	precision	recall	f-score
ADD + T1 + RD	rbf	0.99	0.97	0.98	0.97	0.99	0.98
	extree	1.00	0.97	0.99	0.93	0.99	0.96
ADD + T2 + RD	linear	0.99	0.98	0.99	0.94	1.00	0.97
	extree	1.00	0.99	0.99	0.93	1.00	0.96
T1 + RD + TW1	linear	0.99	0.97	0.98	0.91	1.00	0.95
	extree	1.00	0.98	0.99	0.91	1.00	0.95
T1 + RD + TW2	linear	0.99	0.99	0.99	0.89	1.00	0.94
	extree	1.00	0.99	0.99	0.94	1.00	0.97
T1 + RD + T2	rbf	0.98	0.99	0.98	0.87	1.00	0.93
	extree	1.00	0.98	0.99	0.96	0.99	0.97
ED + RD + T2	linear	0.98	0.99	0.98	0.91	1.00	0.95
	extree	1.00	0.99	0.99	0.92	0.99	0.95

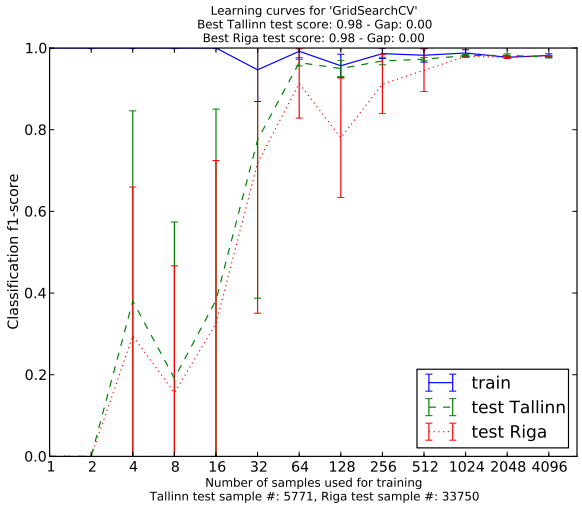

Figure 3: Learning curves for training, Tallinn test and Riga test data using grid search (SVM parameter optimization) and features ADD + T1 + RD

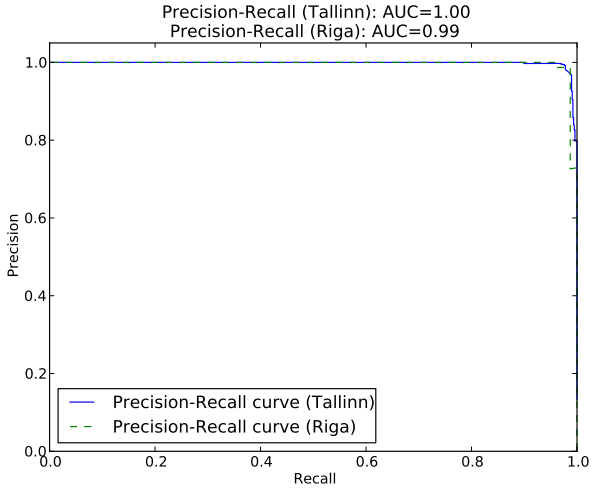

Figure 4: Precision-recall curve for features ADD + T1 + RD trained with SVM

were trained with different learning algorithms. The total number of test run is over 400. Here we present results for some of those tests. The tables described in this section all have the same structure. Each table has different number of features used for the learning problem. The first column indicates, which features are used for training and testing. Every feature set is evaluated with 2 different learning algorithms. The first one is Support Vector Machine (SVM) and the other is extra trees (extended version of decision trees). For SVM, we have shown the used kernel (linear or rbf - radial basis function). All this data is trained with the same amount of randomly chosen samples from Tallinn data. The ratio of positive and negative samples remains the same for training and testing data. All the tables present results which are trained with 50% of Tallinn data (5771 samples, 771 of those are positive). The other part of Tallinn data will be used for evaluation. In addition, we have evaluated every trained model with Riga data (33750 samples, 78 of those are positives).

Metrics used for evaluation are:

$$precision = \frac{tp}{tp + fp} \tag{1}$$

$$recall = \frac{tp}{tp + fn} \tag{2}$$

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3}$$

,

where tp is the number of *true positive* predictions (how many predicted positive samples are actually positives), fn is the number of *false negative* predictions (predicted negative, but actually are positive), fp is the number of *false positive* predictions (predicted positive, actually are negative). In our problem, we concider both precision and recall equally important, therefore we use F_1 score[6] as the main evaluation of our model.

The Table 2 shows the results for 8 best features used alone. For training and testing, only values of one feature (for example title similarity) was used. Note that every feature has a short code after the name. Those codes are used later in other tables. From the results we can see that using only one feature, we can have F-score near 0.8. We can also see that edit distance or Levenshtein distance (feature ED) alone does not separate duplicates from non-duplicates very well.

The Table 3 present results for some combination of two features. The F-score here is already very promising. We can see here that title similarity combined with location/address similarity gives good results. For example ADD + T1 (address string similarity and custom title similarity) and T1 + RD (custom title similarity and re-calculated distance similarity) both give F-scores about 0.95. We did test also with the combinations of three features which are presented in Table 4. All presented results have F-score near or over 0.95. The first feature set ADD + T1 + RD (address string similarity, title custom similarity and recalculated distance similarity) has the highest Riga test result. Also, as we stated earlier, we believe that the manual annotation of duplicate objects has also an error of 3-5%, then 0.98 F-score

[6] F_1 score treats both precision and recall equally important

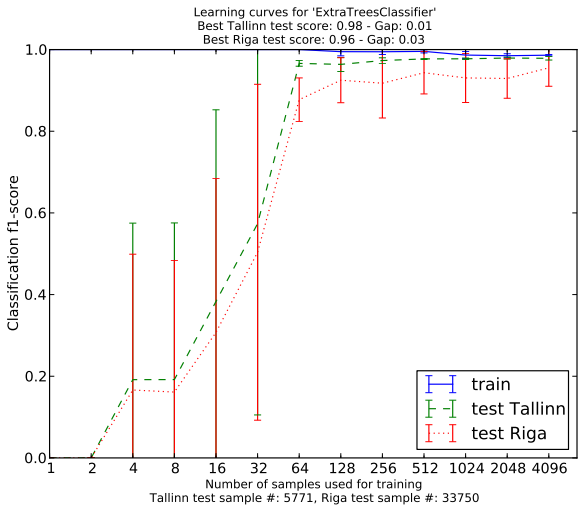

Figure 5: Learning curves for training, Tallinn test and Riga test data using extree (decision trees) and features ADD + T1 + RD

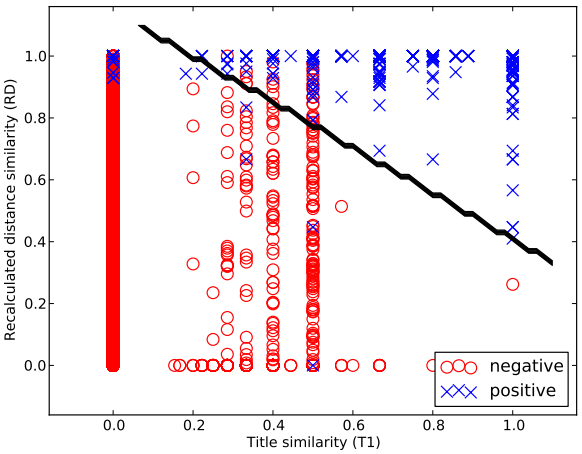

Figure 6: Tallinn dataset with features T1 + RD

is very good result. We have completed tests also for 4 and 5 feature combinations, none of those performed better than 3 feature combination ADD + T1 + RD.

In addition, we evaluated the learning capability of our best model. In the Figure 3 we have drawn learning curves (which indicate F-scores) for training and testing datasets for different training dataset size using grid search (SVM). To give a better overview of the changes to the curves, we have used logarithmic scale. X-axis shows the training dataset size. Our results in the tables were trained with about 5000 samples. As can be seen from the learning curve, we get good predictions already starting from 64 training samples. The learning curve also shows that our learning problem does not suffer from overfitting (our training accuracy is not too high compared to testing accuracy, the gap is close to 0 starting from 1000 training samples).

If we compare grid search learning curve with decision trees' curve in Figure 5, we can see, that decision tree takes more training samples to get better result on Riga (additional test) data. In addition to learning curve, we have plotted precision and recall curve in Figure 4 for SVM model with features ADD + T1 + RD.

In the end, we will give an example plot of our samples with 2 features in Figure 6. We have used SVM to plot features T1 + RD (custom title similarity and recalculated distance similarity) which gives an idea, how the features are located. The black line is class trained separator: right upper corner is for positive (duplicates) predictions, lower left corner is for negative (non-duplicates) predictions.

6. RELATED WORK

Deduplication is very popular topic, especially when we have access to more information. Various data sources across the Internet are presenting information about the same objects. Semantic web tries to ease the problem to offer global URIs or references to widely known information sites (for example DBpedia[7]). Here we will look into some papers, which have been inspired our work and will continue to do that.

One similar system to ours is presented in [7]. They also have geographical objects and they need to detect duplicates. They have introduced 4 similarity functions or features: two title comparisons, address similarity and category similarity. Taking category into account is something we have to look into. The main problem is that every data source has its own categories. We do keyword extraction to find some types for every object, but often we will get misplaced objects in some category. Another problem is that currently we have build category hierarchy for user interface. Often this does not work well in terms of similarity. We should introduce a separate category model, which could be used for finding duplicate objects.

Aforementioned article has surprisingly low accuracy after the training. May-be the data we are using has more quality and therefore learning and predicting work even with small dataset sizes. The authors do not go into details about the learning part. Also, it is somewhat unclear, why they did not use more samples for their training and testing set.

Bilenko and Mooney have written about deduplication in databases [2] and evaluating duplicate detection [3]. The approach they are using is similar to what we have described in this paper. One interesting work from them is learning string distance metrics. While in our case, edit distance did not give good enough results, we manually created a custom title comparison function. Instead, we could also learn, which metrics could be used and combine different standard methodologies with weights in order to find the best one for tourism object titles. As most of our objects do not have descriptions, we cannot make use of using larger text to compare objects. This would be interesting and hopefully we will start getting more descriptions and reviews about the objects, which gives us opportunity to do more text similarity calculations. One very active field of study is research paper deduplication. The text of the paper is very important to detect whether two articles are the same or not.

In [3], they have mentioned precision and recall curve as one evaluation metrics for deduplication problem. We have also drawn those curves to study features and dataset. In

[7]DBpedia, see *http://dbpedia.org/*

the result section we presented the curve for our best model. With current data and feature setup, we were not able to use this curve much, because our problem was too easy to solve. We will be applying our deduplication methodology on another domain, where we certainly will have lower accuracy. That is where the the precision and recall curves can help to solve problems.

In [6] the author represents learning approach for duplicate detection for geospatial objects. The approach is again very similar to what we have presented here. They argue in their paper that traditional textual similarity functions do not work well with place names, because their stylistic variability is too great. Our custom title similarity function is implemented exactly because of this. They also mention that common names (like *street*) could be ignored when comparing address strings. We had one feature, which tried to do that (more specifically, words that occur more often, have less impact). But non weighted word similarity worked better. We are in the middle of analysing the results, where we can see, in which cases weighted words would outperform non-weighted words, if at all. The results the author presents in the paper are very promising. The paper has a good overview of text comparison metrics, which we could also try on our system to see, whether we can get better results for title comparison.

Martins describes the problem of different geocoding, which we also tackled in our research. We try to get geo coordinates for every object in our database to ease to route planning. Also, the same coordinates can be used to find duplicates. If we scrape information from different web sites, we end up with noisy data. Sometimes the same object was located 500-1000 meters away. We used normalization of geo coordinates using object addresses, which improved the quality. In the result table with one feature, you can compare features OD (originally scraped coordinates) and RD (recalculated based on address). After recalculation, if the address was the same, we actually got the same coordinates for the objects. Before, there was always some difference. For some objects, it was 5-10 meters, for other 100 meters etc.

An article about Swoosh [1] presents good results in entity resolution problem in terms of accuracy and speed. They introduce an algorithm which can reduce the number of comparisons between the objects. While our system basically compares every object to every other object, their algorithm can perform magnitude of less comparisons. To determine the similarity between two objects, they use features and coefficients/thresholds. Manual tuning of those coefficients can give good results, but as we learned in our system, we were able to improve the accuracy when we learned from the data itself. Mentioned algorithm Swoosh is still very promising and we are planning to do some more experiments where we integrate Swoosh with machine learning. The idea would be to learn which features and which coefficients to use, while Swoosh can help with finding candidate objects and merging.

7. CONLUSIONS

In our paper we have given an overview of a tourism recommender system Sightsplanner. We described the information gathering process and presented our manually tuned deduplication algorithm. From there we were motivated to improve our accuracy of duplicate detection. We have used machine learning techniques to find ways to increase our accuracy. In this paper we describe, how we have set up the learning system. We have manually annotated 1500 duplicate objects to evaluate our learning. We introduced the technique, how we compare objects, where every sample in our dataset is a comparison of two objects. Using pairs like that, we are able to find groups of three or more duplicate objects.

We present some of our results in the previous section. Along with prediction accuracy, we also analysed learning curves. As can be seen from one of our learning curves, we could use several times less training data to achieve approximately the same f-score. Depending on the needs of the system. To guarantee f-score 0.9 or higher, we could use may-be 10 times less training data. This gives an opportunity to start with training your model with rather small dataset. The learning curve also showed that with adding more data, the f-score may even improve (if we start for example 256 training samples).

Another finding of our work is that our approach learning by comparing tourism object pairs can be applied cross-category, cross-language and cross-city datasets. For training, we used Tallinn eating places. For additional evaluation we tested trained models on Riga data, where in addition to eating places there are museums, galleries etc. The results show that using two or more features, we are able to get very high results with Riga data.

The main goal of our learning setup was to improve the manual deduplication accuracy. Our manually tuned algorithm was able to detect duplicates with F-score 0.85. With machine learning, we have increased the f-score to **0.98**.

In the future, we will try to apply the same methodology used in tourism domain to some other domain, probably for research paper deduplication. We have to work on our textual similarity functions. We also plan to run our recommender system for larger cities, where we can gather more data to further evaluate our deduplication system. Along with more data, we will investigate new features (we already mentioned category similarity).

8. ACKNOWLEDGMENTS

This research has been supported by European Regional Development Fund. This work has been funded by the European Commission as part of the FP7 Marie Curie IAPP project TEAM (grant no. 251514).

9. REFERENCES

[1] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal*, March 2008. VLDB Journal (Online First) link: http://dx.doi.org/10.1007/s00778-008-0098-x.

[2] M. Bilenko. Learning to combine trained distance metrics for duplicate detection in databases. *Submitted to CIKM-2002*, (February):1–19, 2002.

[3] M. Bilenko. On evaluation and training-set construction for duplicate detection. *of the KDD-2003 Workshop on*, pages 7–12, 2003.

[4] A. Luberg, P. Järv, K. Schoefegger, and T. Tammet. Context-aware and multilingual information extraction for a tourist recommender system. In *Proceedings of the 11th International Conference on Knowledge*

Management and Knowledge Technologies - i-KNOW '11, number c, pages 1–8, New York, New York, USA, 2011. ACM Press.

[5] A. Luberg, P. Järv, and T. Tammet. Information Extraction for a Tourist Recommender System. In *Information and Communication Technologies in Tourism 2012*. 2012.

[6] B. Martins. A Supervised Machine Learning Approach for Duplicate Detection over Gazetteer Records. In C. Claramunt, S. Levashkin, and M. Bertolotto, editors, *GeoSpatial Semantics*, volume 6631 of *Lecture Notes in Computer Science*, pages 34–51. Springer Berlin / Heidelberg, 2011.

[7] Y. Zheng, X. Fen, X. Xie, S. Peng, and J. Fu. Detecting nearly duplicated records in location datasets. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 137–143, New York, NY, USA, 2010. ACM.