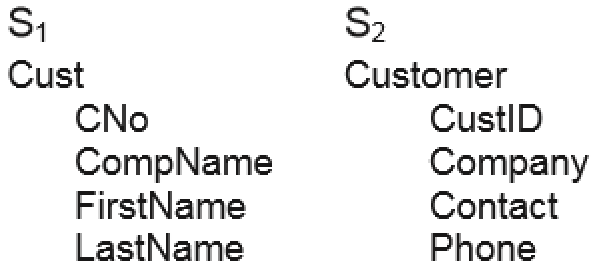
Introduction to Schema Matching

The fundamental problem is schema matching, which takes two (or more) database schemas to produce a mapping between elements (or attributes) of the two (or more) schemas that correspond semantically to each other. The objective is to merge the schemas into a single global schema. This problem arises in building a global database that comprises several distinct but related databases. One application scenario in a company is that each department has its database about customers and products that are related to the operations of the department. Each database is typically designed independently and possibly by different people to optimize database operations required by the functions of the department. This results in different database schemas in different departments. However, to consolidate the data about customers or company operations across the organization in order to have a more complete understanding of its customers and to better serve them, integration of databases is needed. The integration problem is clearly also important on the Web as we discussed above, where the task is to integrate data from multiple sites.

Schema matching is challenging for many reasons. First of all, schemas of identical concepts may have structural and naming differences. Schemas may model similar but not identical contents, and may use different data models. They may also use similar words for different meanings.

Although it may be possible for some specific applications, in general, it is not possible to fully automate all matches between two schemas because some semantic information that determines the matches between two schemas may not be formally specified or even documented. Thus, any automatic algorithm can only generate candidate matches that the user needs to verify, i.e., accept, reject or change. Furthermore, the user should also be allowed to specify matches for elements that the system is not able to find satisfactory match candidates. Let us see a simple example.

Example 1: Consider two schemas, S1 and S2, representing two customer relations, Cust and Customer.



S_1	S_2
Cust	Customer
CNo	CustID
CompName	Company
FirstName	Contact
LastName	Phone

We can represent the mapping with a similarity relation, , over the power sets of S1 and S2, where each pair in represents one element of the mapping. For our example schemas, we may obtain

Cust.CNo $\cong$ Customer.CustID
Cust.CompName $\cong$ Customer.Company
{Cust.FirstName, Cust.LastName} $\cong$ Customer.Contact

There are various types of matching based on the input information.

1. Schema-level only matching: In this type of matching, only the schema information (e.g. names and data types) is considered. No data instance is available.

2. Domain and instance-level only matching: In this type of match, only instance data and possibly the domain of each attribute are provided. No schema is available. Such cases occur quite frequently on the Web, where we need to match corresponding columns of the hidden schemas.

3. Integrated matching of schema, domain and instance data: In this type of match, both schemas and instance data (possibly domain information) are available. The match algorithm can exploit clues from all of them to perform matching.

Pre-Processing for Schema Matching

For pre-processing, issues such as concatenated words, abbreviations, and acronyms are dealt with. That is, they need to be normalized before being used in matching

Prep 1 (Tokenization): This process breaks an item, which can be a schema element (attribute) or attribute value, into atomic words. Such items are usually concatenated words. Delimiters (such as "-", "_", etc.) and case changes of letters are used to suggest the breakdown. For example, we can break "fromCity" into "from City", and "first-name" into "first name". A domain dictionary of words is typically maintained to help the breakdown. Note that if "from", "city", "first" and "name" are not in the dictionary, they will be added to the dictionary. Existing dictionary words are also utilized to suggest the breakdown. For example, "deptcity" will be split into "dept" and "city" if "city" is a word. The dictionary may be constructed automatically, which consists of all the individual words appeared in the given input used in matching, e.g., schemas, instance data and domains. The dictionary is updated as the processing progresses. However, the tokenization step has to be done with care. For example, we have "Baths" and "Bathrooms" if we split "Bath" with "Room" it could be a mistake because "Rooms" could have a very different meaning (the number of rooms in the house). To be sure, we need to ensure that "Bathroom" is not an English word, for which an online English dictionary may be employed.

Prep 2 (Expansion): It expands abbreviations and acronyms to their full words, e.g., from "dept" to "departure". The expansion is usually done based on the auxiliary information provided by the user or collected from other sources. Constraints may be imposed to ensure that the expansion is likely to be correct. For example, we may require that the word to be expanded is not in the English dictionary, with at least three letters, and having the same first letter as the expanding word. For example, "CompName" is first converted to (Comp, Name) in tokenization, and then "Comp" is expanded to "Company".

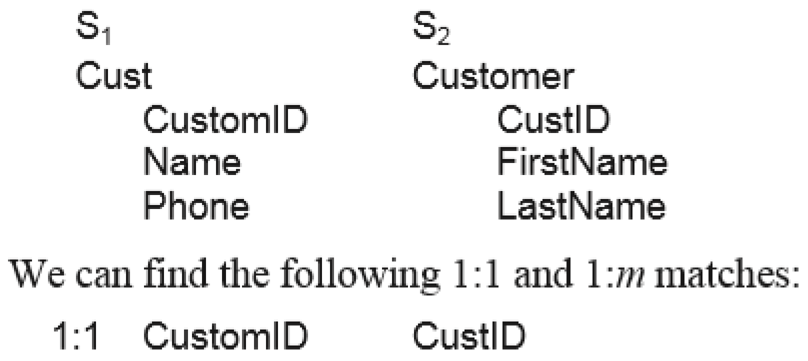
Prep 3 (Stopword removal and stemming): These are information retrieval pre-processing methods (see Chap. 6). They can be performed to attribute names and domain values. A domain specific stopword list may also be constructed manually. This step is useful especially in linguistic based matching methods discussed below.

Prep 4 (Standardization of words): Irregular words are standardized to a single form (e.g., “colour” --“color”, “Children” -- “Child”)

Schema-Level Matching

A schema level matching algorithm relies on information about schema elements, such as name, description, data type and relationship types (such as part-of, is-a, etc.), constraints and schema structures. Before introducing some matching methods using such information, let us introduce the notion of match cardinality, which describes the number of elements in one schema that match the number of elements in the other schema. In general, given two schemas, S1 and S2, within a single match in the match relation one or more elements of S1 can match one or more elements of S2. We thus have 1:1, 1:m, m:1 and m:n matches. 1:1 match means that one element of S1 corresponds to one element of S2, and 1:m means that one element of S1 corresponds to a set of m (m > 1) elements of S2.

Consider the following schemas:



S_1	S_2
Cust	Customer
CustomID	CustID
Name	FirstName
Phone	LastName

We can find the following 1:1 and 1:*m* matches:

1:1	CustomID	CustID
1:*m*	Name	FirstName, LastName

m:1 match is similar to 1:m match; m:n match is considerably more complex. An example of an m:n match is to match Cartesian coordinates with polar coordinates. There is little work on such complex matches. Most existing approaches are for 1:1 and 1:m matches.

We now describe some general matching approaches that employ various types of information available in schemas. There are two main types of information in schemas, natural language words and constraints. Thus, there are two main types of approaches to matching.

Linguistic Approaches

They are used to derive match candidates based on the names, comments or descriptions of schema elements

N2 – Synonyms: The names of two elements from different schemas are **synonyms**, e.g., Customer ≅ Client. This requires the use of thesaurus and/or dictionaries such as WordNet. In many cases, domain dependent or enterprise specific thesaurus and dictionaries are required.

N3 – Equality of hypernyms: *A* is a **hypernym** of *B* if *B* is **a kind of** *A*. If *X* and *Y* have the same hypernym, they are likely to match. For example, "Car" *is-a* "vehicle" and "automobile" *is-a* "vehicle". Thus, we have Car ≅ vehicle, automobile ≅ vehicle, and Car ≅ automobile.

N4 – Common substrings: Edit distance and similar pronunciation may be used. For example, CustomerID ≅ CustID, and ShipTo ≅ Ship2.

N5 – Cosine similarity: Some names are natural language words or phrases (after pre-processing). Then, text similarity measures are useful. **Cosine similarity** is a popular similarity measure used in information retrieval (see Chap. 6). This method is also very useful for Web query interface integration since the labels of the schema elements are natural language words or phrases (see the query interfaces in Fig. 10.1)

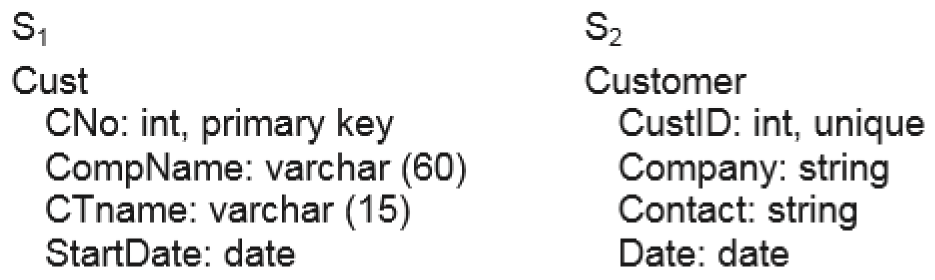
N6 – User provided name matches: The user may provide a domain dependent match dictionary (or table), a thesaurus, and/or an ontology.

Constraint Based Approaches

Constraints such as data types, value ranges, uniqueness, relationship types and cardinalities, etc., can be exploited in determining candidate matches:

C1: An equivalence or compatibility table for data types and keys that specifies compatibility constraints for two schema elements to match can be provided, e.g., string ≅ varchar, and (primary key) ≅ unique.

Consider the following two schemas:



S_1	S_2
Cust	Customer
CNo: int, primary key	CustID: int, unique
CompName: varchar (60)	Company: string
CTname: varchar (15)	Contact: string
StartDate: date	Date: date

Constraints can suggest that “CNo” matches “CustID”, and “StartDate” may match “Date”. “CompName” in S1 may match “Company” in S2 or “Contact” in S2. Likewise, “CTname” in S1 may match “Company” or “Contact” in S2. In both cases, the types match. Although in these two cases, we are unable to find a unique match, the approach helps limit the number of match candidates and may be combined with other matchers (e.g., name and instance matchers). For structured schemas, hierarchical relationships such as is-a and part-of relationships may be utilized to help match.

Domain and Instance-Level Matching

In this type of matching, value characteristics are exploited to match schema elements For example, the two attribute names may match according to the linguistic similarity, but they may have different domain value characteristics. Then, they may not be the same but homonyms. For example, Location in a real estate sell may mean the address, but could also mean some specific locations, e.g., lakefront property, hillside property, etc. In many applications, data instances are available, which is often the case in the Web database context. In some applications, although the instance information is not available, the domain information of each attribute may be obtained. This is the case for Web query interfaces. Some attributes in the query interface contain a list of possible values (the domain) for the user to choose from. No type information is explicitly given, but it can often be inferred. We note that the set of value instances of an attribute can be treated in the similar way as a domain. Thus, we will only deal with domains below.

Let us look at two types of domains or types of values: simple domains and composite domains. The domain similarity of two attributes, A and B, is the similarity of their domains: dom(A) and dom(B). Definition (Simple Domain): A simple domain is a domain in which each value has only a single component, i.e., the value cannot be decomposed.

A simple domain can be of any type, e.g., year, time, money, area, month, integer, real, string, etc.

Data Type: If there is no type specification at the schema level, we identify the data type from the domain values. Even if there is a type specification at the schema level for each attribute, we can still refine the type to find more characteristic patterns. For example, the ISBN number of a book may be specified as a string type in a given schema. However, due to its fixed format, it is easy to generate a characteristic pattern from a set of ISBN numbers, e.g., a regular expression. Other examples include phone numbers, post codes, money, etc. Such specialized patterns are more useful in matching compatible attribute types.

Semi-automatic approach: This is done via pattern matching. The pattern for each type may be expressed as a regular expression, which is defined by a human expert. For example, the regular expression for the time type can be defined as "[0⍰9]{2}:[0⍰9]{2}" or "dd:dd" (d for digit from 0-9) which recognizes time of the form "03:15". One can use such regular expressions to recognize integer, real, string, month, weekday, date, time, datetime (combination of date and time), etc. To identify the data type, we can simply apply all the regular expression patterns to determine the type. In some cases, the values themselves may contain some information on the type. For example, values that contain "$" or "US$" indicate the monetary type. For all values that we cannot infer their types, we can assume their domains are of string type with an infinite cardinality.

Automated approach: Machine learning techniques, e.g., grammar induction, may be used to learn the underlying grammar/pattern of the values of an attribute, and then use the grammar to match attribute values of the other schemas. This method is particularly useful for value of fixed format, e.g., zip codes, phone numbers, zip codes, ISBNs, date entries, or money-related entries, if their regular expressions are not specified by the user.

The following methods may be used in matching:

DI 1 – Data types are used as constraints. The method C1 above is applicable here. If the data/domain types of two attributes are not compatible, they should not be matched. We can use a table specifying the degree of compatibility between a set of predefined generic data types, to which data types of schema elements are mapped in order to determine their similarity.

DI 2 – For numerical data, value ranges, averages and variances can be computed to access the level of similarity.

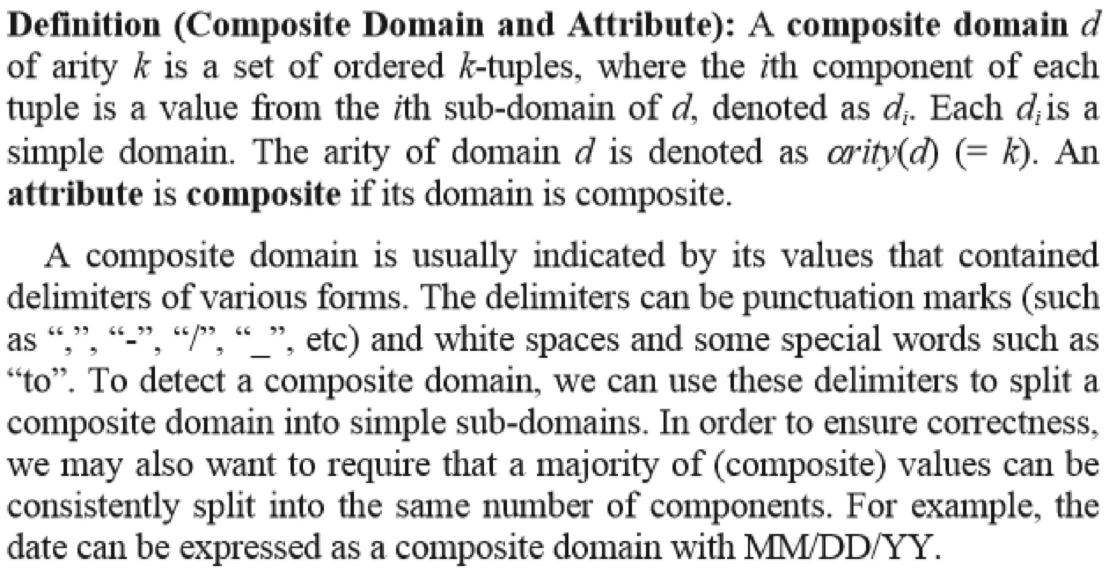
DI 3 – For categorical data, we can extract and compare the set of values in the two domains to check whether the two attributes from different schemas share some common values. For example, if an attribute from S1 contains many "Microsoft" entries and an attribute in S2 also contains some "Microsoft"'s, then we can propose them as a match candidate.

DI 4 – For alphanumeric data, string-lengths and alphabetic/non-alphabetic ratios are also helpful.

DI 5 – For textual data, information retrieval methods such as the cosine measure may be used to compare the similarity of all data values in the two attributes.

DI 6 – Schema element name as value is another match indicator, which characterizes the cases where matches relate some data instances of a schema with a set of elements (attributes) in another schema. For example, in the airfare domain one schema uses

“Economy” and “Business” as instances (values) of the attribute “Ticket Class”, while in another interface, “Economy” and “Business” are attributes with the Boolean domain (i.e., “Yes” and “No”). This kind of match can be detected if the words used in one schema as attribute names are among the values of attributes in another schema.



Definition (Composite Domain and Attribute): A **composite domain** d of arity k is a set of ordered k-tuples, where the ith component of each tuple is a value from the ith sub-domain of d, denoted as d_i. Each d_i is a simple domain. The arity of domain d is denoted as $arity(d)$ $(= k)$. An **attribute** is **composite** if its domain is composite.

A composite domain is usually indicated by its values that contained delimiters of various forms. The delimiters can be punctuation marks (such as “,”, “-”, “/”, “_”, etc) and white spaces and some special words such as “to”. To detect a composite domain, we can use these delimiters to split a composite domain into simple sub-domains. In order to ensure correctness, we may also want to require that a majority of (composite) values can be consistently split into the same number of components. For example, the date can be expressed as a composite domain with MM/DD/YY.

DI 7 – The similarity of a simple domain and a composite domain is determined by comparing the simple domain with each sub-domain of the composite domain. The similarity of composite domains is established by comparing their component sub-domains. We note that splitting a composite domain can be quite difficult in the Web context. For example, without sufficient auxiliary information (e.g., information from other sites) it is not easy to split the following: “Dell desktop PC 1.5GHz 1GB RAM 30GB disk space”.

The Problem of Opinion Mining

In this first section, we define an abstraction of the opinion mining problem. It enables us to see a structure from the complex and intimidating unstructured text. Moreover, for most opinion-based applications, it is essential to analyze a collection of opinions rather than only one because one opinion represents only the view of a single person, which is usually not sufficient for action. This indicates that some form of summary of opinions is needed.

-Opinion mining, also known as sentiment analysis, is the process of extracting subjective information from text data, such as reviews, tweets, and social media posts. The main problem with opinion mining is the subjectivity of language, which makes it difficult to accurately classify text data as positive, negative, or neutral.

-There are several challenges associated with opinion mining. First, people express opinions in many different ways, using idiomatic expressions, sarcasm, irony, and other linguistic devices that can be difficult to interpret. Second, context plays a crucial role in determining the sentiment of a text. For example, the same sentence can be positive in one context and negative in another. Third, there is a high degree of variability in people's opinions, making it difficult to establish a standard for what constitutes positive, negative, or neutral sentiment.

-Despite these advances, opinion mining remains a challenging problem, as language is constantly evolving and new linguistic devices are being developed all the time. As such, researchers in this field must continually refine their methods and models to keep pace with the ever-changing landscape of language and opinion.

- The detection of spam and fake reviews, mainly through the identification of duplicates, the comparison of qualitative with summary reviews, the detection of outliers, and the reputation of the reviewer

- The limits of collaborative filtering, which tends to identify most popular concepts and to overlook most innovative / out of the box thinking - the risk of a filter bubble, where automated content analysis combined with behavioural analysis leads to a very effective but ultimately deviating selection of relevant opinions and content, so that the user is not aware of content which is somehow different from his expectations

- The asymmetry in availability of opinion mining software, which can currently be afforded only by organisations and government, but not by citizens. In other words, government have the means today to monitor public opinion in ways that are not available to the average citizens. While content production and publication has democratized, content analysis has not.

- The integration of opinion with behaviour and implicit data, in order to validate and provide further analysis into the data beyond opinion expressed

- The continuous need for better usability and user-friendliness of the tools, which are currently usable mainly by data analysts

Problem Definitions We use the following review segment on iPhone to introduce the problem (an id number is associated with each sentence for easy reference):

"(1) I bought an iPhone a few days ago. (2) It was such a nice phone. (3) The touch screen was really cool. (4) The voice quality was clear too. (5) However, my mother was mad with me as I did not tell her before I bought it. (6) She also thought the phone was too expensive, and wanted me to return it to the shop. … "

The question is: what we want to mine or extract from this review? The first thing that we notice is that there are several opinions in this review.

Sentences (2), (3), and (4) express some positive opinions, while sentences (5) and (6) express negative opinions or emotions. Then we also notice that the opinions all have some targets. The target of the opinion in sentence (2) is the iPhone as a whole, and the targets of the opinions in sentences (3) and (4) are "touch screen" and "voice quality" of the iPhone, respectively. The target of the opinion in sentence (6) is the price of the iPhone, but the target of the opinion/emotion in sentence (5) is "me", not iPhone. Finally, we may also notice the holders of opinions. The holder of the opinions in sentences (2), (3), and (4) is the author of the review ("I"), but in sentences (5) and (6) it is "my mother." With this example in mind, we now formally define the opinion mining problem. We start with the opinion target.

Document sentiment classification

Document sentiment classification is the task of analyzing a piece of text, such as a document or a tweet, and determining the overall sentiment expressed within it. This is typically done by assigning a label to the text that reflects its sentiment, such as positive, negative, or neutral.

Classification Based on Supervised Learning

Sentiment classification obviously can be formulated as a supervised learning problem with three classes, positive, negative, and neutral. Training and testing data used in the existing research are mostly product reviews, which is not surprising due to the above assumption. Since each review already has a reviewer-assigned rating (e.g., 1–5 stars), training and testing data are readily available. For example, a review with 4 or 5 stars is considered a positive review, a review with 1 or 2 stars is considered a negative review and a review with 3 stars is considered a neutral review.

Sentiment classification is similar to but also somewhat different from classic topic-based text classification, which classifies documents into predefined topic classes, e.g., politics, sciences, sports, etc. In topic-based classification, topic-related words are important. However, in sentiment classification, topic-related words are unimportant. Instead, opinion words (also called sentiment words) that indicate positive or negative opinions are important, e.g., great, excellent, amazing, horrible, bad, worst, etc

Terms and their frequency. These features are individual words or word ngrams and their frequency counts (they are also commonly used in traditional topic-based text classification). In some cases, word positions may also be considered. The TF-IDF weighting scheme from information retrieval may be applied too. These features have been shown quite effective in sentiment classification.

Part of speech. It was found in many researches that adjectives are important indicators of opinions. Thus, adjectives have been treated as special features.

Opinion words and phrases. Opinion words are words that are commonly used to express positive or negative sentiments. For example, beautiful, wonderful, good, and amazing are positive opinion words, and bad, poor, and terrible are negative opinion words.

Rules of opinions. Although opinion words and phrases are important, there are also many other expressions that contain no opinion words or phrases but indicate opinions or sentiments.

Syntactic dependency. Words dependency-based features generated from parsing or dependency trees are also tried by several researchers.

Classification Based on Unsupervised Learning

It is not hard to imagine that opinion words and phrases are the dominating indicators for sentiment classification. Thus, using unsupervised learning based on such words and phrases would be quite natural.It performs classification based on some fixed syntactic phrases that are likely to be used to express opinions. The algorithm makes use of a natural language processing technique called part-of-speech (POS) tagging. The part-of-speech of a word is a linguistic category that is defined by its syntactic or morphological behaviour. Common POS categories in English grammar are: noun, verb, adjective, adverb, pronoun, preposition, conjunction, and interjection. Then, there are many categories which arise from different forms of these categories. For example, a verb can be a verb in its base form, in its past tense, etc

Step 1: It extracts phrases containing adjectives or adverbs as adjectives and adverbs are good indicators of opinions. However, although an isolated adjective may indicate opinion, there may be insufficient context to determine its opinion orientation

For example, the adjective “unpredictable” may have a negative orientation in an automotive review, in a phrase such as “unpredictable steering,” but it could have a positive orientation in a movie review, in a phrase such as “unpredictable plot.” Therefore, the algorithm extracts two consecutive words, where one member of the pair is an adjective or adverb, and the other is a context word.

Step 2: It estimates the semantic orientation of the extracted phrases using the pointwise mutual information (PMI) measure given in Equation (1):

$$PMI(term_1, term_2) = \log_2\left(\frac{\Pr(term_1 \wedge term_2)}{\Pr(term_1)\Pr(term_2)}\right). \quad (1)$$

Here, Pr(term1 $\wedge$ term2) is the co-occurrence probability of term1 and term2, and Pr(term1)Pr(term2) gives the probability that the two terms co-occur if they are statistically independent. The ratio between Pr(term1 $\wedge$ term2) and Pr(term1)Pr(term2) is thus a measure of the degree of statistical dependence between them. The log of this ratio is the amount of information that we acquire about the presence of one of the words when we observe the other.

The semantic/opinion orientation (SO) of a phrase is computed based on its association with the positive reference word “excellent” and its association with the negative reference word “poor”:

SO(phrase) = PMI(phrase, “excellent”) – PMI(phrase, “poor”).

(2) The probabilities are calculated by issuing queries to a search engine and collecting the number of hits. For each search query, a search engine usually gives the number of relevant documents to the query, which is the number of hits. Thus, by searching the two terms together and separately, we can estimate the probabilities in Equation (1). Turney, the author of, used the AltaVista search engine because it has a NEAR operator, which constrains the search to documents that contain the words within ten words of one another in either order. Let hits(query) be the number of hits returned. Equation (2) can be rewritten as:

$$SO(phrase) = \log_2\left(\frac{hits(\text{phrase } NEAR \text{ "excellent"})hits(\text{"poor"})}{hits(\text{phrase } NEAR \text{ "poor"})hits(\text{"excellent"})}\right). \qquad (3)$$

To avoid division by zero, 0.01 is added to the hits.

Step 3: Given a review, the algorithm computes the average SO of all phrases in the review and classifies the review as recommended if the average SO is positive, not recommended otherwise. Final classification accuracies on reviews from various domains range from 84% for automobile reviews to 66% for movie reviews.

Explain how DOM trees are used in structured data extraction.

Building DOM Trees

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content.

To build a DOM tree, we start with the HTML code, which is parsed by the web browser into a tree structure. Each element in the tree is represented by a node. There are different types of nodes, such as element nodes, attribute nodes, and text nodes.

Here's an example of HTML code and its corresponding DOM tree:

```
<html>
<head>
    <title>My Website</title>
</head>
<body>
    <header>
        <h1>Welcome to My Website</h1>
        <nav>
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
        </nav>
    </header>
    <main>
        <article>
            <h2>My First Article</h2>
            <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed ac lectus
eget nunc dictum lobortis sed sit amet neque. Nullam luctus est eu libero venenatis pharetra.
</p>
        </article>
        <article>
            <h2>My Second Article</h2>
            <p>Etiam convallis tellus id ipsum congue, ut bibendum quam molestie.
Maecenas faucibus ipsum sed tellus imperdiet, in cursus ex bibendum. </p>
```

```
        </article>
    </main>
    <footer>
        <p>&copy; 2023 My Website. All rights reserved.</p>
    </footer>
</body>
</html>
```

```
- Document
    - Doctype: html
    - Element: html
        - Element: head
            - Element: title
                - Text: "My Website"
        - Element: body
            - Element: header
                - Element: h1
                    - Text: "Welcome to My Website"
                - Element: nav
                    - Element: ul
                        - Element: li
                            - Element: a
                                - Text: "Home"
                        - Element: li
                            - Element: a
                                - Text: "About"
                        - Element: li
                            - Element: a
                                - Text: "Contact"
            - Element: main
                - Element: article
                    - Element: h2
```

```
                        - Text: "My First Article"
                    - Element: p
                        - Text: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed ac lectus eget nunc dictum lobortis sed sit amet neque. Nullam luctus est eu libero venenatis pharetra. "
                - Element: article
                    - Element: h2
                        - Text: "My Second Article"
                    - Element: p
                        - Text: "Etiam convallis tellus id ipsum congue, ut bibendum quam molestie. Maecenas faucibus ipsum sed tellus imperdiet, in cursus ex bibendum. "
            - Element: footer
                - Element: p
                    - Text: "&copy; 2023 My Website. All rights reserved."
```