Entity extraction for comparison sentences from reviews over the restaurants domain.

By: Sabina Shleizer 307523167

Advisor: Dr. Osnat Mokryn

Contents:

1. Introduction

In recent years, technological advances have led to an increase of consumer-review websites, such as Yelp[1], where service-quality experiences can be shared. These reviews provide the consumers with information about previous consumers' experience. The consumer can then get information about a product or a service quality observed after its consumption. With a click of a button one can now gather information from multiple consumers regarding a variety of products, from restaurants to movies to physicians. However, consumer-review websites do not provide consumers with sufficient tools for evaluating opinions from reviews, so that many of them find it inconvenient to browse through a large number of reviews (Dai, Jin, Lee, & Luca, 2012).

Many customers are facing a situation in which they have to decide upon one restaurant over the other. In order to make better decisions, they can attempt to compare several reviews of restaurants. However, it is clear that as long as customers have an access to a huge amount of reviews, reading each restaurant's reviews, for the purpose of getting a better point of view, is not an ideal solution. In addition, investigating large amounts of data is a time-consuming job, which may cause many helpful or otherwise important reviews to be left unnoticed due to lack of time. Therefore, a system that can automatically provide a summary of comparisons between two (or more) restaurants, based on their reviews, would be very useful in many areas such as marketing (Yang & Ko, 2011).

Comparison is also one of the most convincing ways of evaluation (Xu et al., 2011). Extracting comparative sentences from a text can be useful for business decisions. For example, if the owner of a new restaurant would like to know its consumers' opinions on the restaurant, and how they are compared with other restaurants' reviews within the area or the genre. Much of that information is now readily available on the Web in the form of customer reviews, forum discussions, blogs, etc. Extracting such information can significantly help businesses in their marketing and product benchmarking efforts.

[1] http://www.yelp.com

Much research in the information extraction field has been done in the recent years (Davidov & Rappoport, 2006; Davidov, Tsur, & Rappoport, 2010; Etzioni et al., 2005; Tsur, Rappoport, & Davidov, 2010). This research has a huge influence on business processes. Automatic identification of consumers' opinions from reviews can help identifying strengths and weaknesses of a business, as well as taking marketing decisions. For example, Tsur et al., (2010) examine the issue of sarcasm within online reviews. In their research, they suggest an algorithm that can recognize sarcastic sentences in online product reviews automatically.

In this paper, we focus on obtaining and extracting comparisons from social media sites in the restaurants domain. Extracting this information is not only important for the business owners, but also for potential customers, as it enables customers to make better decisions (Jindal & Liu, 2006). Specifically, this paper deals with finding and extracting comparative sentences between items from online reviews. The restaurant domain was chosen for this paper as it suggests an interesting challenge in particular as restaurant names vary in a free manner style. Restaurants might be named by their location, after the owner, or even after a favorite dish. Unlike the product domain, in which a product's name is a unique identifier when referenced in a review, the restaurant domain introduces an additional challenge in this respect. Consider, for example, a restaurant named "Tika Masala". Clearly, when diners at other Indian restaurants refer to "Tika Masala", they might refer a favorite dish rather than another restaurant. Similarly is the case with restaurants named "So", or "Lemon", which are commonly used words in different contexts.

In linguistics, comparatives are based on specialized morphemes *more/most, -er/-est, less/least and as*, for the purpose of establishing orderings of superiority, inferiority and equality. Alternatively, *than, as*, are used for making a 'standard' against which an entity is compared. (Jindal, Liu, & Bing, 2006). To extract comparative sentences from restaurant review we present here two methods that were tailored for the restaurant domain The first method is based on SASI algorithm for semi-supervised recognition of sarcastic sentences in product reviews (Tsur et al., 2010). We create a two-phase algorithm. The first phase consists of a semi-supervised pattern acquisition module, based on SASI, for identifying patterns of sentences that compare two or more restaurants. The second phase involves employing a module for comparative patterns classification on the sentences found on the first phase The second method, which outperforms

the first, is based on an algorithm for identifying comparative sentences in text documents (Jindal & Liu, 2006). In this algorithm we replace all the words in the sentence with the part-of-speech tag and then, by applying class-sequential rules and by using various classification methods, we extract comparative sentences.

2. Background and related work

2.1 Identification of comparative sentences

2.1.1 Mining comparative sentences:

Although the problem of identifying a sentiment or an opinion is well studied, only a few works focused on the identification of comparative sentences, discussed hereafter. Jindal & Liu (2006) were the first to define the text-mining problem of comparative sentence mining (CSM). According to them, a comparative sentence expresses an ordering relation between two sets of entities, with respect to some common features. For example, the comparative sentence *"Canon's optics are better than those of Sony and Nikon"* expresses the comparative relation: (better, {optics}, {Canon}, {Sony, Nikon}).

Given a set of evaluative texts on the Web, e.g., reviews, forum postings, and news articles, the task of comparative sentence mining is to:
1. Identify comparative sentences from texts (explained below).
2. Extract comparative relations from the identified comparative sentences, so that a new type of rules ("LSR label sequential rules") is proposed for extraction. These rules are based on the following assumptions: a) there is only one relation in a sentence. In practice, they show that in their dataset this assumption is violated only in a very small number of cases; b) Entities and features are nouns (include nouns, plural nouns and proper nouns) and pronouns. This assumption showed to cover the majority of the cases in their dataset.

Jindal & Liu propose an approach based on pattern discovery and supervised learning to identify comparative sentences. They created an algorithm that finds comparative sentences in 4 stages:

1) Using a keyword strategy to find the comparing sentences.
2) Utilizing POS tags in the sentence in order to convert the sentence into a pattern.
3) Using sequential pattern-mining (SPM) algorithm to find all sequential patterns that satisfy a selected minimum support and confidence.
4) Applying the NB and SVM classifier to automatically classify each sentence into one of the two classes: "comparative" and "non-comparative" (based on the filtered data).

On their work, Jindal et al., aimed to find keywords that cover almost all comparative sentences, with a very high recall. First, they manually found a list of 30 words by going through a subset of comparative sentences. Then, they used WordNet to find their synonyms. After a manual pruning- process, a final list of 69 words was produced. However, since the non-gradable comparative sentences did not necessarily use the specific keywords, they included 9 more words and phrases such as "but", "whereas", "on the other hand", etc., which are sometimes used in non-gradable comparisons. The words with POS tags of JJR, RBR, JJS and RBS were also good indicators so they included words with this POS tags also. Altogether, they had 83 keywords and key phrases which contained {JJR, RBR, JJS, RBS} ∪ {words such as favor, prefer, win, beat, but, etc.} ∪ {phrases such as number one, up against, etc.}. Therefore, the researchers could find most of the comparative sentences and their new problem was to remove the non-comparative sentences from the comparative.

In the second stage of their algorithm, Jindal & Liu utilized POS tags in order to find patterns in the dataset sentences. They assumed that Even though the contents of some sentences may vary, their underlying language patterns can be the same. Using the raw words, such patterns were not found. However, by Replacing each word with its POS tag (Part-of-speech tag), the pattern became apparent. Thus, POS tags capture content-independent language patterns, which are useful to this research, according to the following process:

1) Select the LSR rule with the highest confidence. Replace the matched elements in the sentences that satisfy the rule with the labels ({ci}) in the rule.

2) Recalculate the confidence of each remaining rule based on the modified data from step 1.

3) Repeat step 1 and 2 until no rule is left with confidence higher than the minimum confidence value (they used 90%).

The rules are then applied to match each comparative sentence in the test data to extract the components of the relation.

To extract comparison sentences using key words, Jindal & Liu constructed a database in the following way:

1. For each sentence that contains at least one keyword or key phrase, they use the words that are within the radius of 3 of each keyword in the sentence as a sequence in their data. Their

experiments demonstrated that the radius of 3 was optimum whereas the Radius of 4 or more gave many spurious patterns that over-fit the data. Using too few words was found to give insufficient information.

2. Each word is then replaced with its POS tag.

3. A class is attached to each sequence, according to whether the sentence is comparative or non-comparative,

As the database was constructed, Jindal & Liu used the sequential pattern-mining (SPM) algorithm to find all sequential patterns that satisfy a selected minimum support (frequency). A sequential pattern is simply a sub-sequence that appears more frequently in the input sequences than the minimum support threshold. Unlike classic sequential pattern-mining, which is unsupervised, they mine sequential rules with fixed classes. A class-sequential rule (CSR) is a rule with a sequential pattern on the left and a class label on the right. They generated class-sequential rules which meet the minimum confidence threshold of 60%. For the minimum support threshold, they proposed the multiple minimum supports model, in which each word set a minimum support based on the frequency that appears in the training data. This model enables to find those rare patterns without generating too many over-fitting rules that harm the classification process. To achieve the multiple minimum support effect, Jindal & Liu established the parameter τ and set it to 0.1. The minimum support was changed according to the actual frequency of the items in the data (for frequent items the minimum support was high and for rare items the minimum support was low). The mining algorithm is presented in the following pseudo-code:

1. Compute the frequencies of all the items in the training data
2. for each group of items W with the same frequency do
    a. minsup = frequency(W) * τ;
    b. *CSR(trainingData, W, minsup, minconf);
3. end_for

* The function CSR generates all the rules related to the items in W.

Once the rules were constructed, the researches added some manually compiled rules that were more complex and hard to be generated by current pattern-mining techniques. For each

sentence, they found all the rules that were satisfied by the sentence and ran an NB classification on them. The algorithm indicated whether the sentence was comparative or non-comparative, according to the Naive Bayesian classifier result. In conclusion, Jindal & Liu identified the following two challenges:

1. Not all sentences considered as comparisons and having the POS tags JJR, RBR, JJS and RBS, are indeed comparisons.
2. Some sentences are comparisons but do not use any indicative word.

While their method was suitable for the product domain, our results, as detailed in Section 4 show that their method failed to find the majority of comparative sentences in the restaurants domain.

Yang & Ko (2009) proposed a method for automatically identifying Korean comparative sentences from text documents. First, they built an algorithm that extracts comparison sentences from a text document in three stages:

1) They defined a set of comparative keywords.
2) Using the keyword strategy, they extracted a set of comparative-sentence candidates.
3) Using MEM and NB classification methods, they eliminated non-comparative sentences from the candidates.

Their next step was to construct a set of 177 manually extracted comparative keywords. They defined comparative keyword (CK) as a word or a phrase or a long-distance-words sequence.

On the second stage, Yang & Ko divided all the sentences into four categories as follows:

S1 - comparative sentences containing a keyword.
S2 - comparative sentences not containing a keyword.
S3 - non-comparative sentences containing a keyword.
S4 - non-comparative sentences not containing a keyword.

Their goal was to find an effective method to extract S1 and S2 from all the other sentences. However, extracting comparative sentences is not a simple nor an easy problem. It needs more complicated and challenging processes than only searching out some keywords for extracting comparative sentences.

The main problem Yang & Ko identified on their research was that many comparative sentences do not contain comparative words, while non-comparative sentences can contain comparative words. However, as long as a comparative keyword is included in each sentence, the sentence is considered likely to be a comparative sentence. Keyword searching process can therefore detect most of comparative-sentence candidates (S1, S2 and S3) from original text documents. That is, the recall is high but the precision is low, so there is a need to eliminate the incorrect sentences (S3) from the candidate sentences. According to that, Yang & Ko's suggestion was to divide the set of all comparative-sentence candidates into two subsets according to the precision of each keyword, using 90% of the precision as a threshold value. The candidates in the first group, which contained popular keywords, had the average precision of 97.44% and did not require any additional process. The second group needed further processing, due to a low precision of 29.34%.

On the last stage of the algorithm, Yang & Ko used Maximum Entropy Method (MEM) and Naïve Bayes (NB) classifiers to eliminate non-comparative sentences. For feature extraction from each comparative sentence candidate, they used continuous word sequences within the radius of 3 of each keyword in the sentence. After determining the radius, they replaced each word with its POS tag. They added comparative or non-comparative class-tag to each sentence and applied the MEM and NB on the created vectors, which enabled them to decide if comparative sentence candidates are a comparative sentences.

Yang & Ko (2011) further expanded the research on Korean comparative sentences. They focused mainly on two tasks:

1) Classifying comparative sentences into one non-comparative class and seven comparative classes.
2) Mining comparative entities.

On the first stage of the algorithm, they classified the sentences into two groups: comparative and non-comparative sentences, using the method described above (Yang & Ko, 2009).
The next step of their algorithm was to classify the comparative sentences into different comparative types. They defined seven comparative types and employed transformation-based learning (TBL) for comparative sentence classification. TBL is a method for adding

classification to each token developed by Brill in 1995 (Bril 1995). The last step on their work was to extract three kinds of comparative elements: SE (subject entity), OE (object entity) and PR (comparative predicate).

Kennedy (2004) proposed a linguistic approach for identifying comparative sentences. He tried to categorize different types of comparative sentences on the basis of syntax and semantics. Syntax and semantics are terms used in relation to characteristics of language. Syntax is concerned with the structure of language and it is a matter of the logical or grammatical form of sentences, rather than what they refer to or mean. Semantics is concerned with the meaning of words and sentences. In many languages, comparatives are based on specialized morphology and syntax. English exemplifies this type of system: it uses the morphemes "more/-er", "less"and "as", specifically for the purpose of establishing orderings of superiority, inferiority and equality, respectively, and the morphemes "than" and "as" to make the ‘standard’ against which an object is compared. In contrast to Kennedy's work, this paper studied the comparative sentences from the technical side.

Park & Blake (2012) aimed to identify comparative sentences automatically from full text scientific articles. In their work, they introduced 35 features that capture both semantic and syntactic characteristics of a sentence. They used those features with three different classifiers: Naïve Bayes, Support Vector Machines, and Bayesian Networks in order to predict comparison sentences. The experiments were conducted on 122 full-text toxicology articles containing 14,157 sentences, of which 1,735 (12.25%) were comparisons. The experiments shown an F1 score of 71%, 69% and 74% on development set and 76%, 65% and 74% on a validation set for NB, SVM and BN, respectively. The results Park & Blake received where not as good as the result we found, and the results presented in Jindal & Liu (2006). Therefore, we do not preside with nor compare to their method.

2.1.2 Mining the orientation of comparative sentences:

Ganapathibhotla & Liu (2008) focus on mining opinions from comparative sentences, i.e., to determine which entities in a comparison are preferred. Their observation is based upon the notion that in comparative sentences there is usually a context-dependent comparative word, which identifies the opinion's orientation (positive or negative). The entities being compared often appear on the two sides of the comparative word and the basic idea is to convert the opinion's adjectives/adverbs to their comparative forms. After the conversion, these words are manually categorized into increasing and decreasing comparatives. Sentences with opinionated words (e.g., “better”, and “worse”) are usually easy to handle. However, many comparative words are not opinionated. Thus, they used reviews from review sites, which have separate Pros and Cons (identified positive and negative opinions that were separated by reviewers while writing). They identified comparative and entity features words in Pros and Cons sentences, and converted the comparatives to their base forms. Then they applied a set of evaluation rules to determine which of the entities is preferred. For example: If the comparative or superlative comparative word has a positive orientation (e.g. “better”), Entity S1 (which appears before comparative word in the sentence) is temporarily assigned as the preferred entity. Otherwise, Entity S2 is assigned as the preferred entity.

Visualizations of product aspects were also considered in the context of mining the orientation of comparative sentences. Xu, Liao, Li, & Song (2011) used the corpus of Amazon customer reviews in order to suggest a graphical model to extract and visualize comparative relations between products from customer reviews. The interdependencies among relations were taken into consideration to help enterprises discover potential risks and further design new products and marketing strategies. In contrast to this work, Xu et al., (2011) tried to build a map of relations between all the products and did not focus on extracting comparative sentences specifically. Additionally, their work, as well as the others mentioned above, focused on product comparison and not on restaurant comparison.

Gamon et al., (2005) presents a prototype system code named "Pulse" for mining topics and sentiment orientation jointly from free text customer feedback. Their work combines the two dimensions of a free-form customer opinions classification: topic and sentiment (the content, opinions characterization), and presents the results in an intuitive visualization. Both sentiment

detection and topic detection in Pulse are performed at the sentence-level rather than at the document level (sentence-level granularity of analysis allows the discovery of new information even in those scenarios where an overall product rating is already provided at the document level).

Pulse system was applied to a sample of a car reviews database, which contains 406,818 customer car reviews written over a four year period, with no editing beyond simple filtering for profanity. Pulse first extracts a taxonomy of major categories (makes) and minor categories (models) of cars by simply querying the car reviews database. Sentences are extracted from the reviews of each make and model and are processed according to the two dimensions of information that are shown in the final visualization stage: sentiment and topic. To train the domain-specific sentiment classifier (a Naive Bayes classifier), Pulse requires that only a small amount of data be annotated. A small random selection of sentences is labeled by hand as expressing positive, other, or negative sentiment. This small labeled set of data is used with the entirety of the unlabeled data to bootstrap a classifier.

The Sentence-clustering algorithm (a modified version of Nigam et al.'s algorithm) forms clusters from the set of sentences that correspond to a leaf-node in the taxonomy (i.e. a specific model of car). The prototype implements a simple keyword-based soft clustering algorithm with TF-iDF weighting and phrase identification. The clusters are labeled with the most prominent key terms. Once the sentences for a make and model of a car are assigned to clusters and receive a sentiment score from the sentiment classifier, the visualization component displays the clusters and the keyword labels that are produced for the sentences associated with that car. Their work is different from our work as it focuses on mining opinions and not on comparative sentences.

In summary, this chapter discussed various methods for the identification of comparative sentences. It clarified that studies considering comparative sentences from reviews over the restaurants domain are yet to find. Also, when viewing the task of extracting comparisons between the reviewed items only few related papers studied non-supervised extraction of comparative sentences.

2.2 Tools

2.2.1 Text classification:

Given a set of classes, we seek to determine which classes a given document belongs to. The input of this problem is a description of document $x \in X$, where X is the document space; and a fixed set of classes $C = \{c1, c2, \dots, cJ\}$. Using a learning method or a learning algorithm, we then wish to learn a classifier or classification function γ that maps documents to classes: $\gamma: X \rightarrow C$.

The set of rules, or the decision criterion of the text classifier, is learned automatically from a training data. The training-set requires a number of good examples for each class. This type of learning is called "supervised learning", since a supervisor serves as a teacher directing the learning process. The need for a manual classification is not eliminated because the training documents come from a person who has labeled them (labeling refers to the process of annotating each document with its class). We denote the supervised learning method by G and write $G(X)=\gamma$. The learning method G takes the training-set X as input and returns the learned classification function γ. Once we have learned γ, we can apply it to the test set (or test data), for example, a new unlabeled document. This type of learning is analogous to a human learning from past experiences to gain new knowledge in order to improve our ability to perform real-world tasks (Blanken, de Vries, Blok, & Feng, 2007; Id, 2009).

2.2.2 Naive Bayes (NB):

The task of classification can be regarded as estimating the class probabilities given a test example d. We then see which class cj is more probable. The class with the highest probability is assigned to example d. Let A1, A2, …, A|A| be the set of attributes with discrete values in the data set D. Let C be the class attribute with |C| values, c1, c2, …, c|C|. Given a test example d with observed attribute values, the prediction is the class cj such that Pr(C=cj | A1=a1, ..., A|A|=a|A|) is maximal. An assumption is that all attributes are conditionally independent given the class C = cj.

$$\Pr(C = c_j) = \frac{\text{number of examples of class } c_j}{\text{total number of examples in the data set}}$$

$$\Pr(A_i = a_i | C = c_j) = \frac{\text{number of examples with } A_i = a_i \text{ class } c_j}{\text{total number of examples of class } c_j}$$

$$\text{Assign d to calss c} = \arg\max_{c_j} \Pr(C = c_j) \prod_{i=1}^{|A|} \Pr(A_i = a_i | C = c_j)$$

Most assumptions made by NB learning are violated in practice. For example, words in a document are clearly not independent of each other. Despite such violations, researchers have shown that NB learning produces very accurate models. NB can be applied to many different learning problems and is unlikely to produce classifiers that fail catastrophically, but it has an error rate of 20%. NB learning is also very efficient for it scans the training data only once to estimate all the probabilities required for classification. NB is a good classifier if one has fairly little data and intends to train a supervised classifier (Blanken et al., 2007; Id, 2009).

2.2.3 Rocchio:

Our task in the vector space classification is to find good boundaries to divide the vector space into regions. A "good" classification means high classification accuracy on a test data that was not seen during training. Centroid of a class is computed as a vector average or a center of mass from its class members. The boundary between two classes in Rocchio classification is a set of points with an equal distance from the two centroids. This set of points forms a line. Rocchio classification is simple and efficient, but inaccurate if classes are not approximately spheres with similar radii. It ignores details that are related to the distribution of points in a class and only uses distance from the centroid for classification. The Rocchio classification also fails in the case of nonlinear classification problem when, for example, one class is contained in another class (Blanken et al., 2007; Id, 2009).

Centroid calculation: $\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d)$

D_c is the set of documents in D whose class is c and $\vec{v}(d)$ is a normalized vector of D.

The line that determines the boundaries between 2 classes: $\vec{w}^{T}\vec{x} = b$

$\vec{w}^{T}$ is the M-dimensional normal vector and b is a constant.

The classification rule in Rocchio is to classify a point in accordance with the region it falls into.

Equivalently, we determine the centroid $\vec{\mu}(c)$ that the point is closest to and then assign it to c.

$$\text{Assign d to calss } c = \arg\max_{c'} \cos(\vec{\mu}(c'), \vec{v}(d))$$

2.2.4 K nearest neighbors (kNN):

kNN classification determines the decision boundary locally and assigns each document to the majority class of its k closest neighbors, where k is a parameter. kNN is a "lazy" learning method in a sense that no model is learned from the training data. Learning occurs only when a test example needs is classified. kNN requires no explicit training and may use the unprocessed training-set directly in classification. It is less efficient than other classification methods in classifying documents. If the training set is large, then kNN can handle non-spherical and other complex classes better than Rocchio. The parameter k in kNN is often chosen based on experience or knowledge about the classification problem at hand. An alternative way of setting the parameter is to select the k that gives best results on a held-out portion of the training set. Other approach is to set K proportionally to the test set size as $\sqrt[2]{(\text{test set size})}$.

$$\text{score}(c, d) = \sum_{d' \in S_k(d)} I_c(d') \cos(\vec{v}(d'), \vec{v}(d))$$

$S_k(d)$ is the set of d's k nearest neighbors

$I_c(d') = 1$ if d' is in class c and 0 otherwise

Weighing by similarities is often more accurate than simple voting. For example, if two classes have the same number of neighbors in the top k, then the class with the more similar neighbors wins.

Despite its simplicity, researchers have shown that the classification accuracy of kNN can be quite strong, and in many cases as accurate as those elaborated methods. The kNN performs well when we cope with a large amount of training data. kNN is also very flexible as it can work with any arbitrarily shaped decision boundaries. However, kNN is slow at classification time. Due to the fact that there is no model building, each test instance is compared with every training example at the classification time, which can be quite time-consuming, especially when the training-set d and the test-set are large (Blanken et al., 2007; Id, 2009).

2.2.5 Support vector machines (SVM):

SVM is another type of learning system, which has many desirable qualities that make it one of the most popular algorithms. Not only it has a solid theoretical foundation, but it also performs classification more accurately than most other algorithms. An SVM is a kind of large-margin classifier. It is a vector space-based machine learning method where the goal is to find a decision boundary between two classes that are maximally far from any point in the training data. The SVM requires numeric data and builds only two-class classifiers, which are separable training data sets with lots of possible linear separators. The SVM defines the criterion to be looking for a decision surface that is maximally far away from any data point. Only a small part of the points (the point around the decision surface) play part in determining the decision surface that is chosen. Maximizing the margin is good because the points near the decision surface represent very uncertain classification decisions and there is almost a 50% chance of the classifier deciding either way. A classifier with a large margin makes no low certainty classification decisions. In Addition, if we have to place a fat separator between classes, one has fewer choices of where it can be put, what increases the ability to test the data correctly.

2.2.6 Class Sequential Rules (CSR):

Association rules are an important class of regularities in data. Mining of association rules is a fundamental data-mining task. It may be the most important model invented and vastly studied

by the database and data-mining community. Given a set of input sequences, the task is to find all sequential patterns that satisfy a user-specified minimum support (or frequency) constraint. Association rule mining does not consider the order of transactions. However, in many applications such orderings are significant. For these applications, association rules will not be appropriate and sequential patterns are needed. We define the problem of mining sequential patterns and introduce the main concepts involved.

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. A sequence is an ordered list of item sets. An item-set X is a non-empty set of items where $X \subseteq I$. We denote a sequence s by $\langle a_1 a_2 \ldots a_r \rangle$, where a_i is an item set, also called an element of s, and a_i is denoted by $\{x_1, x_2, \ldots, x_k\}$, where $x_j \in I$ is an item. An item can occur only once in an element of a sequence, but can also occur many times in different elements. The size of a sequence is the number of elements in the sequence. The length of a sequence is the number of items in the sequence. A sequence of length k is called a k-sequence. If an item occurs multiple times in different elements of a sequence, each occurrence contributes to the value of k. A sequence $s_1 = \langle a_1 a_2 \ldots a_r \rangle$ is a subsequence of another sequence $s_2 = \langle b_1 b_2 \ldots b_m \rangle$, if there exist integers $1 \leq j_1 < j_2 < \ldots < j_{r-1} \leq j_r$ such that $a_1 \subseteq b_{j_1}$, $a_2 \subseteq b_{j_2}$, …, $a_r \subseteq b_{j_r}$. We also say that s2 contains s1.

Example: We have I = {1, 2, 3, 4, 5, 6, 7}. The sequence ⟨{3}{4, 5}⟩ is contained in ⟨{6}{3, 7}{4, 5, 6}⟩ because {3} ⊆ {3, 7} and {4, 5} ⊆ {4, 5, 8}. However, {3}{8} is not contained in {3, 8} or vice versa. The size of the sequence {3}{4, 5}{8} is 3, and the length of the sequence is 4.

The problem definitions: Given a set S of input data sequences, the problem of mining sequential patterns is to find all sequences that have a user-specified minimum support. The support for a sequence is the fraction of total data sequences in S that contains this sequence.

Example:

The sequence database:

1	⟨{30}{90}⟩
2	⟨{10,20}{30}{10,40,60,70}⟩
3	⟨{30,50,70,80}⟩
4	⟨{30}{30,40,70,80}{90}⟩
5	⟨{90}⟩

The output of sequential patterns with the minimum support of 25%:

1-sequences	⟨{30}⟩, ⟨{40}⟩, ⟨{70}⟩, ⟨{80}⟩, ⟨{90}⟩
2-sequences	⟨{30}{40}⟩, ⟨{30}{70}⟩, ⟨{30}{90}⟩, ⟨{30, 70}⟩, ⟨{30, 80}⟩, ⟨{40, 70}⟩, ⟨{70, 80}⟩
3-sequences	⟨{30} {40, 70}⟩, ⟨{30, 40, 80}⟩

A sequential pattern then is simply a sub-sequence that appears more frequently in the input sequences than the minimum support threshold. Many algorithms exist for mining such patterns in data-mining. A class sequential rule (CSR) is a rule with a sequential pattern on the left and a class label on the right.

Class sequential rules (CSR) are also analogous to class association rules (CAR). Let S be a set of data sequences. Each sequence is also labeled with a class y. Let I be the set of all items in S, and Y be the set of all class labels, $I \cap Y = \emptyset$. Thus, the input data D for mining is represented with $\{(s_1, y_1), (s_2, y_2), \ldots, (s_n, y_n)\}$, where s_i is a sequence in S and $y_i \in Y$ is its class.

CSR is of the form $X \rightarrow y$, where X is a sequence, and $y \in Y$. A data instance (s_i, y_i) is said to cover a CSR, $X \rightarrow y$, if X is a subsequence of s_i. A data instance (s_i, y_i) is said to satisfy a CSR if X is a subsequence of s_i and $y_i = y$. The support (sup) of the rule is the fraction of total instances in D that satisfies the rule. The confidence (conf) of the rule is the proportion of instances in D that covers the rule and also satisfies the rule.

Example:

1	⟨{1}{3}{5}{7, 8, 9}⟩	c_1
2	⟨{1}{3}{6}{7, 8}⟩	c_1
3	⟨{1, 6}{9}⟩	c_2
4	⟨{3}{5, 6}⟩	c_2
5	⟨{1}{3}{4}{7, 8}⟩	c_2

Using the minimum support of 20% and a minimum confidence of 40%, one of the discovered CSRs is:

⟨{1}{3}{7, 8}⟩ → c1 [support = 2/5 and confidence = 2/3]

Data sequences 1 and 2 satisfy the rule, and data sequences 1, 2 and 5 cover the rule.

The GSP algorithm can be modified to produce algorithms for mining all class-sequential rules. The algorithm displayed below is a modified GSP algorithm for mining class-sequential rules. Given a set S of input data sequences where each data sequence is labeled with a class Y, the modified GSP algorithm finds all sequences that have a user-specified minimum support and user minimum confidence (Blanken et al., 2007).

2.2.7 Part-Of-Speech (POS):

Part-of-speech tagging is one of the most important text analysis tasks used to classify words into their part-of-speech and label them according to the tag set, which is a collection of tags used for the POS tagging. The POS tagging is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition, as well as its context.

Example:
Text: They refuse to permit us to obtain the refuse permit

POS: ('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PRP'),
('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse', 'NN'), ('permit', 'NN')]

In this paper we use NLTK 3 Taggers, which contain classes and interfaces for part-of-speech tagging. A tag is a case-sensitive string that specifies some property of a token, such as its part of

speech. This package defines several taggers, which take a token list, assign a tag to each token, and return the resulting list of tagged tokens. Most of the taggers are built automatically based on the "Penn Treebank tag-set".

The Penn Treebank tag-set is based on the Brown Corpus, along with the deduction of the number of POS considerably. A key-factor in reducing the tag-set was to eliminate redundancy by taking into account both lexical and syntactic information. Another difference between these two tag-sets is that in the Brown Corpus words tend to be tagged independently of their syntactic function. By contrast, the Penn Treebank encodes a word's syntactic function in its POS tag whenever possible. A final difference between the Penn Treebank tag-set, and all other tag-sets, is that it allows words to be associated with more than one POS tag. The Penn Treebank tag-set contains 36 POS tags and 12 other tags (for punctuation and currency symbols).

1. CC Coordinating conjunction	19. PP$ Possessive pronoun
2. CD Cardinal number	20. RB Adverb
3. DT Determiner	21. RBR Adverb, comparative
4. EX Existential there	22. RBS Adverb, superlative
5. FW Foreign word	23. RP Particle
6. IN Preposition/subordinating conjunction	24. SYM Symbol
7. JJ Adjective	25. TO to
8. JJR Adjective, comparative	26. UH Interjection
9. JJS Adjective, superlative	27. VB Verb, base form
10. LS List item marker	28. VBD Verb, past tense
11. MD Modal	29. VBG Verb, gerund/present participle
12. NN Noun, singular or mass	30. VBN Verb, past participle
13. NNS Noun, plural	31. VBP Verb, non-3rd ps. sing. Present
14. NNP Proper noun, singular	32. VBZ Verb, 3rd ps. sing. Present
15. NNPS Proper noun, plural	33. WDT wh-determiner
16. PDT Predeterminer	34. WP wh-pronoun
17. POS Possessive ending	35. WP$ Possessive wh-pronoun
18. PRP Personal pronoun	36. WRB wh-adverb

The tagged version of the Penn Treebank corpus is produced in two stages, using a combination of automatic POS assignment and manual correction. The automatic POS assignment is provided by a cascade of stochastic and rule-driven taggers developed on the basis of early experience. The result of the first, automated stage of POS tagging is given to annotators to correct (Marcus, Santorini, & Marcinkiewicz, 1993; Santorini, 1990).

2.2.8 Semi-supervised Algorithm for Sarcasm Identification (SASI):

The aim of the SASI algorithm is to recognize sarcastic sentences in product reviews.
The SASI algorithm employs two modules:

1. Semi-supervised pattern acquisition for identifying sarcastic patterns that serve as features for a classifier.
2. A classification algorithm that classifies each sentence to a sarcastic class.

The algorithm was experimented on a data set of about 66000 Amazon reviews for various books and products and obtained a precision of 77% and a recall of 83.1% for identifying sarcastic sentences (Tsur et al., 2010). We further detail SASI here, as in this paper we devise an algorithm for extracting comparison sentences that are built upon it.
First, the researches collected a small set of existing sentences and gave them scores between 1-5, where 1 is a sarcastic sentence and 5 is not.
From each sentence a feature-vector was created, while two features were taken under consideration:

1. Pattern base
2. Syntactic

(These vectors will be used later in the classification stage).

In this stage of the algorithm, each appearance of a product/author/company/book name is replaced with a corresponding generalized tag '[product]', '[company]', '[title]' and '[author]' in order to produce less specific patterns.

2.2.9 Pattern-based features:

<u>Pattern extraction</u>

The researchers classify words into two types:
HFW (High Frequency Word) – words that appear in a frequency of 1K / 1M.

1. CW (Content Word) – words that appear in a frequency of 100 / 1M.

Punctuation marks and the '[product]', '[company]', '[title]' and '[author]' tags are considered as HFW.

They define a pattern as an ordered sequence of HFW and CW, according to the following rules:

1. Each pattern contains 2-6 HFW
2. Each pattern contains 1-6 CW
3. Each pattern should start and end in HFW (in order to avoid collection of patterns which capture a part of a multiword expression).

For each sentence a multitude of patterns may be produced.

Example:

For the sentence: "Garmin apparently does not care much about product quality or customer support", the following patterns are applicable:

"[company]* CW does not CW much"

"does not CW much about CW CW or"

"not CW much"

"about CW CW or CW CW.*"

*"[company]" and "." are treated as high frequency words.

Pattern selection

At this stage, the following patterns are removed:

1. Product-specific patterns
   Example: "looking for a CW camera". This pattern is specific to a type of camera.
2. Patterns that appear as a positive or negative context.
   Example: "either CW or CW". This pattern can appear both in a sarcastic and non-sarcastic sentence.

Pattern matching

presently, each pattern receives a value to create a single-feature vector. Values are calculated according to the following scheme:

1 : Exact match – all the pattern components appear in the sentence in correct order without any additional words.

1. α : Sparse match – same as exact match but additional non-matching words can be inserted between pattern components.

2. $\gamma * n/N$: Incomplete match – only $n > 1$ of N pattern components appear in the sentence, while some non-matching words can be inserted in-between. At least one of the appearing components should be a HFW.
3. 0 : No match – nothing or only a single-pattern component appears in the sentence.

$\alpha = \gamma = 0.1$

Example:

For the sentence presented in the above section:

"Garmin apparently does not care much about product quality or customer support"

The following values will be calculated:

"[title] CW does not" - the value would be 1.

"[title] CW not" – the value would be 0.1.

"[title] CW CW does not" – the value would be $0.1 * 4/5 = 0.08$.

Punctuation-based features

The features calculated in the previous section were added the following:

Sentence length in words.

1. Number of "!" characters in the sentence.
2. Number of "?" characters in the sentence.
3. Number of quotes in the sentence.
4. Number of capitalized/all capitals words in the sentence.

The resulted values were normalized to be between 0 and 1, by dividing them according to the maximal observed value.

In order to decide if a sentence in the test-set is sarcastic or not, the researches use the kNN classifier. They constructed feature-vectors for each example in the training and test-sets.

For each example in the test set they found the k nearest vectors in the training- set. Thus, the score for the test-set example is a weighted average of the k closest training-set vectors. If no matching vectors were found, the test-set example received a score of 1 (Tsur et al., 2010).

3 Data and Method

3.1 Data Description

We are interested in finding and extracting comparative sentences between two restaurants in online product reviews. We use a collection of 1037 reviews extracted from Yelp[2]. The collection contains reviews of different restaurants written by different reviewers.
Each review is for a specific restaurant. The name of the restaurant appears separately in the meta-data part of the data-set, and was extracted from the site. In case the review includes a comparison with another restaurant, the name of the other restaurant will appear in the review itself. Since our main goal is finding reviews that compare 2 or more restaurants, we would like to capture helpful sentences, which include such names.
While extracting the comparative sentences we have two main concerns:

1. The name of the restaurant mentioned in the review must be different from the restaurant the review was written about. In order to avoid this problem, we compared the restaurant's name found inside the review text to the name of the reviewed restaurant. The sentence was added to our new extracted database only if these two names were different.
2. To generalize the dataset, we automatically replaced each appearance of a restaurant's name with the corresponding generalized [rest] tags.

3.1.1 Training-set construction:

We extracted 1037 reviews from “Yelp” using the following algorithm.
We scanned all the reviews on “Yelp” and for each review:

A. We scanned for the restaurant’s name in the review text.
B. If succeeded, we checked whether the restaurant is different from the restaurant the review was written about.
C. If succeeded, we added the review to the training set.
D. We extracted the sentence that was suspected to be comparative from each review.
E. We scanned the training-set and manually tagged all the sentences. Sentences that contained comparisons between 2 or more restaurants were tagged as 1, whereas all other reviews were tagged as 0.

[2] http://www.yelp.com

3.1.2 Labeling:

The data set was manually labeled by two students, each working alone. We labeled the sentences that were extracted as following: 1 = has a comparison, 0= no comparison in the sentence. In addition, we divided the sentences into different comparison types: g= first restaurant is compared favorably to second, e= equivalent, l=first restaurant is compared as inferior to second.

3.1.3 Data summary:

	Number of sentences	Percent	Percent of valid sentences	Percent of comparative sentences
Total	1037	100%		
Sentences with problems	39	4%		
Total valid sentences	998	96%	100%	
Comparative sentences	663	64%	66%	100%
First restaurant is compared favorably to second	210	20%	21%	32%
Equivalent	146	14%	15%	22%
First restaurant is compared as inferior to second	307	30%	31%	46%
Non comparison	335	32%	34%	

3.2 Method

3.2.1 Method Description:

This paper studies the problem of identifying comparative sentences in restaurant reviews at the sentence-level. The problem is related to extracting comparative sentences from text documents. Our main objective is finding an algorithm that identifies comparison sentences in restaurants

review. We propose to study the comparative sentence identification problem without categorizing the comparative sentences into different comparison types. In this paper we present two algorithms: one is based on the Semi-supervised Algorithm for Sarcasm Identification (SASI) and the other is based on an algorithm for identifying comparative sentences in text document. Results will be presented for each algorithm separately. These algorithms will be compared in the conclusion chapter.

3.2.2 Algorithms:

To identify whether any of the keywords-based approaches is valid, we first implemented the algorithm suggested in Jindal & Liu (2006). Since the total list of the 83 keywords they used for finding comparative sentences was not published, we tried to implement their method using only the keywords that were mentioned in their work explicitly. In this stage, we analyze our data searching for JJR, JJS, RBR and RBS tags. These tags were used as keywords in Jindal & Liu (2006) algorithm. We discovered that only 163 sentences out of 998 sentences (16%) contained words with POS tags 'JJR', 'JJS', 'RBP' and 'RBS'. We also found that only 128 out of 663 (19%) comparative sentences contained words with these POS tags. Therefore, using this algorithm, we can only find 18% of the comparative sentences utmost, and are likely to find less. Due to that concern we could not proceed with this method.

In order to solve the problem that was not approached on Jindal & Liu (2006), and in order to suggest an applicable method for the identification of comparative sentences in the restaurant domain, this paper focuses on two algorithms: Pattern-based algorithm and sequence-feature algorithm.

3.2.2.1 Pattern-based Algorithm

3.2.2.1.1 Rational:

Pattern matching for classification is a broad area of research. Tsur et al., (2010) and Davidov et al., (2010) proposed a unique method for creating patterns, by finding patterns of high-frequency words (HFW) and content words (CW). Words that appear in the English language in a frequency of 1000 words per million are set as HFW, whereas words in a frequency of 100 words per million are set as CW. The researches created a long list of patterns, labeled them

manually, and defined over a large dataset new patterns according to their distance from the manually tagged patterns.

As the keyword-based approach seems to fail to perform in the restaurant domain, we try to construct patterns that differentiate comparative sentences from non-comparative sentences. In order to achieve a clear set of data that is relevant to the restaurant domain, we operated 4 stages: pattern extraction, classification, distance measuring and K selection.

3.2.2.1.2 Pattern extraction:

The first stage of the algorithm is the pattern extraction stage. Going through the restaurants reviews, we can easily notice that each review usually focuses on specific restaurants. However, using a pattern that includes a restaurant name makes the pattern specific for this restaurant reviews. Unfortunately, it is almost impossible for the algorithm to connect patterns that contain different restaurant names. In order to produce less specific patterns, we automatically replace each appearance of a restaurant name with corresponding generalized [rest] tags. Since the patterns we extract in this stage will be later used as an input data for the kNN classifier, we aimed to capture as many helpful patterns as possible.

We then extract a pattern for each sentence based on an algorithm that was first introduced by Davidov & Rappoport (2006) and was used later by Tsur et al.. Both researches proposed a unique method for creating patterns, by classifying words into two types: high-frequency words (HFW) and content words (CW). Words that appear in the English language in a frequency of 1000 words per million are set as HFW, whereas words in a frequency of 100 words per million are set as CW. According to their method, on this paper, for each sentence we replaced the content words with the [CW] tag as long as they were less important to the content of the pattern. Unlike Tsur et al., in our pattern creation we removed the punctuation marks due to the fact that many reviewers use punctuation marks out of context.
In order to decide if a word is a HFW or CW, we used a stop-word list that can be viewed in the appendix section. Words that were found in the stop-words list were labeled HFW and all the other words were labeled as CW. The concept of the HFW is very similar to stop words, since they are defined as very frequent, however not significant in the sentence. Additionally, since

Tsur et al., considered the tags ‘[product]’, ‘[company]’, ‘[title]’ and ‘[author]’ as HFWs, we treat the [rest] tag as HFWs.

We extract patterns from sentences according to the following rules, with a goal of defining the shortest pattern possible:
1) Each pattern is allowed to have 2-6 HFW and 1-6 CWs.
2) Each pattern is required to start and to end with a HFW.
Thus, a minimal pattern is of the form [HFW] [CW] [HFW], and must have at least 3 words and a maximal pattern cannot have more than 12 words.

Example:

The sentence:

The best dessert I have got since my dinner at La Ciccia.

The pattern:

have cw cw cw cw at [rest]

3.2.2.1.3 Classification:

After creating a learning-set, we use the remaining 1/6 of the data as a test set. We perform a kNN classification on the patterns in the test-set and for each pattern we find the k nearest patterns. We then count the positive and the negative patterns from the k nearest patterns that were found and decide whether the test sentence is positive or negative. If the majority of the patterns were comparative, we classify the sentence as comparative.otherwise, we classify the sentence as non-comparative. If the test sentence had an equal number of positive and negative neighbors, or had none, it was classified as negative.

3.2.2.1.4 Distance Measuring:

On the third stage, we describe the most important part in the classification, which is the way the distance between patterns was measured. We gave a score for each pattern, according to the distance measured from the tested pattern and by following these rules:

1: Exact match – all the pattern components appear in the sentence in the correct order, without any additional words.

α : Sparse match – same as exact match, but additional non-matching words can be inserted between pattern components.

$\gamma * n/N$: Incomplete match – only $n > 1$ of N pattern components appear in the sentence, while some non-matching words can be inserted in-between. At least one of the appearing components should be a HFW.

0 : No match – nothing or only a single pattern component appears in the sentence.

We will set $\alpha = \gamma = 0.1$

Example:

The sentence:

The best dessert I have got since my dinner at La Ciccia.

Pattern distance:
have cw cw cw cw at [rest] = 1.0
have cw cw at [rest] = 0.1
have cw cw cw cw at cw [rest] = 0.1*7/8

Patterns that were given the score of 1 are considered to be the closest neighbors. The farther we are from the template, the farther the score is dropped. We gather the k patterns that received the highest score and then classify the test-sentence, according to the majority of patterns that were

collected. In the k collected patterns, patterns that received a lower score were treated equally, regardless of their score.

3.2.2.1.5 K selection:

The k value that was used for the kNN classification was set to 12 according to the following calculation:

K was calculated as $\sqrt[2]{(\text{test set size})} = \sqrt[2]{160} = 12$

We experimented with different k values (k=5, k=20, k=30) and did not get better results.

3.2.2.2 **Sequence Feature Algorithm**

3.2.2.2.1 Rational:

Comparative sentences in the restaurant domain do not use pre-defined keywords (nor follow a set). However, the syntactic structure of the comparative sentences varies, but might have a similar structure, at the part of the sentence doing the actual comparison. Hence, on the second algorithm we would like to identify part-of-speech (POS) tags near and around the compared restaurant's name. We use the words within the radius of 3 words from the restaurant name as a sequence in our data, based on Jindal & Liu (2006) claim that a radius of 3 gives optimal results. Also, a sequence constructed by words within radius of 4 or above has been too specific, while using a radius of 2 or less did not give sufficient information. Our experiments produced the same results.

Example:

This is easily the best meal that I have had since La Ciccia.

('This', 'DT'), ('is', 'VBZ'), ('easily', 'RB'), ('the', 'DT'), ('best', 'JJS'), ('meal', 'NN'), ('that', 'WDT'), ('I', 'PRP'), ('have', 'VBP'), ('had', 'VBN'), ('since', 'IN'), ('La Ciccia', 'rest')

('had', 'VBN'), ('since', 'IN'), ('La Ciccia', 'rest')

We started with the ahi tuna ceviche, which I actually enjoyed more than the numerous ceviches I've had at SPQR.

('We', 'PRP'), ('started', 'VBD'), ('with', 'IN'), ('the', 'DT'), ('ahi', 'NN'), ('tuna', 'NN'), ('ceviche', 'NN'), ('which', 'WDT'), ('I', 'PRP'), ('actually', 'RB'), ('enjoyed', 'VBD'), ('more', 'JJR'), ('than', 'IN'), ('the', 'DT'), ('numerous', 'JJ'), ('ceviches', 'NNS'), ('I', 'PRP'), ("'ve", 'VBP'), ('had', 'VBN'), ('at', 'IN'), ('SPQR', 'rest')

('had', 'VBN'), ('at', 'IN'), ('SPQR', 'rest')

This example shows that even if the content of the sentences is different and there is a great variation between POS tags far from the restaurant name, the POS tags around the restaurant name are identical. This is why the main strategy of the sequence-feature algorithm is POS tags strategy.

3.2.2.2.2 POS tags strategy:

POS tags strategy was first presented by Jindal & Liu (2006) for the purpose recognizing comparative sentences. We will also use this strategy for its efficiency. On this paper, we discovered that we cannot use the words as they are in each sentence, since the content of some sentences may be very different but their meaning can be the same. Using the words as they are may not produce such patterns.

Example:

> Americano is better than SPQR
>
> NOPA is smaller than La Ciccia

Although an English speaker can clearly notice the resemblance of these two sentences, it will be very difficult for an algorithm to detect any pattern. However, replacing each word with its POS tag and each restaurant's name with the tag [rest], turns the pattern apparent to the algorithm. Therefore, from this point we will replace all the words in the comparative sentence with their POS tags and all the names of the restaurants with a [rest] tag.

3.2.2.2.3 Algorithm steps:

The sequence-feature algorithm contains 4 stages:

1) Sequence construction – on this stage we replace all the words in the sentence with POS tags and the restaurant's name in the sentence with the [rest] tag. Also, we cut the sentences to a sequence within the radius of 3 words of the restaurant name.

2) Preparing sequence vectors –on this stage we convert the sequence to a sequence-vector that will be used later in the classification learning-stage (4).
3) Applying CSR rules – on this stage we generate SCR rules, based on the sequence we generated in stage (1). We use the CSR technique for removing noise sequences. This stage is optional and will not be used necessarily.
4) Classification learning –On this stage we will perform NB and SVM machine-learning on the vector, to decide if the sentence is comparative or not.

All 4 sections are described in detail as follows:

I. Sequence construction:

We construct sequences that will be used by our algorithm. The sequences are constructed in five stages:

1) Replacing the name of the restaurant with the [rest] tag in each sentence. Given the fact that the restaurant name is a key-component of our sentence, the [rest] tag is used as a keyword in our algorithm. In order to replace a restaurant's name with a tag, we use a database that contains all restaurants names that derived from "Yelp" reviews.
2) Removing punctuation marks, since many of the reviewers use unnecessary punctuation marks.
3) Replacing all the words with a POS tag. As mentioned earlier, the actual word was not used, in order to make it easier for the algorithm to compare the patterns. Instead, we used the postag function provided by The Natural Language Toolkit (NLTK) 3.0 – an open source Python library for Natural Language Processing, to convert the words in the reviews to POS.
4) Using the words that are within the radios of 3 words of the restaurant name in the sentence as a sequence in our data.
5) Labeling each sentence by "Comparative" or "Non-comparative"

Example:

The sentence:

The best dessert I have got since my dinner at La Ciccia.

Intermediate sequence:
[('The', 'DT'), ('best', 'JJS'), ('dessert', 'NN'), ('I', 'PRP'), ('have', 'VBP'), ('got', 'VBN'), ('since', 'IN'), ('my', 'PRP$'), ('dinner', 'NN'), ('at', 'IN'), (Rest)]
Final sequence:
({PRP} {NN} {IN} {Rest}) Class: comparing between two restaurants

II. Preparing sequence vectors:

We were looking for a matching POS tags around a restaurant's name the reviewer is comparing the reviewed restaurant to. However, the approach of having a sequence feature-vector of POS tags does not take the order in which the POS tags appear into account. While we did not find any global patterns for the sentences, we noticed the existence of local patterns around the compared restaurant's name. To allow for a partial ordering over the resulted feature vectors, we created a vector in which each POS tag and each couple of adjacent POS tags have a separate entry in the feature vector. Hence, the two POS tags 'NN', 'IN' become three different features in the feature vector, as follows: 'NN', 'IN', 'NN, 'IN'.
The following is an example of the effect of the bi-grams approach on the resulted vectors.

Example:

sequence1 = 'VBP VBN IN rest'
sequence2 = 'VBP rest VBN IN'

1-gram-only vectorizer
vector1 = [1 1 1 1]
vector2 = [1 1 1 1]

(1,2)-gram vectorizer
vector1 = [1 1 1 0 1 1 1 0 1]

vector2 = [1 0 1 1 1 1 1 1 0]

The vectors received carry more information and enable a more specific representation of the comparison itself. Hence, we have repeated the entire process with all the experiments described in the previous section.

Since the NB and the SVM classifiers are designed to direct the usage of vectors, and cannot receive literal input, we converted the sequence constructed in the previous stage to sequential vectors. This data will be later used in the classification stage.

The creation of sequential vectors data is described in the following stages:

1) Each sequence is converted to numerical vector using Scikit-learn 2.0 - Machine Learning tools for data mining and data analysis in Python CountVectorizer function. We used two different methods for creating the vector:
   - 1-gram method – in this method we convert each POS tag in the sequence into a different entry in the vector.
   - 1-2-gram range method – in this method we will extract 1-gram and 2-gram from each sequence, i.e. we convert each POS tag and each couple of adjacent POS tags in the sequence into a different entry in the vector.

   The results of these two methods will be shown below.
2) "Comparative" sentences are converted to 1, and "Non-Comparative" sentences converted to 0.
3) Each sentence forms a tuple of the type (sequential vector, class) in the data.
4) Each vector created in this stage is linked to its original sequence created in the previous stage.

III. Applying SCR rules:

The CSR rules model is used to remove "noise sequences" from the training-set. Our goal is to remove sequences that are not commonly used in our database nor specific for one class only, since classification of these sequences would be arbitrary and therefore redundant for our algorithm.

In this stage we generate class-sequential rules, which meet the minimum confidence and minimum support. We use a minimum support value of 10%, and a minimum confidence value

of 10%. This is due the fact our database is not large enough for using greater minimum support and minimum confidence values. We eliminate all the sequences with support and confidence smaller than 10% and all the sequential vectors linked to these sequences from the training data. This method will be applied later on the algorithm.

IV. Classification learning:

In the final stage, a SVM and NB classification was performed on the vectors in the test-set. A sentence was classified as comparative or not according to the output of the classifier.

4 Results

The results will be presented according to the two algorithms used: pattern-based algorithm and sequence-feature algorithm.

4.1 Pattern-based algorithm results:

The precision, recall, accuracy and F-score results are shown in percentages. The table below shows the average of 6 different test-run of our algorithm. During each test-run we randomly divided the dataset using the Scikit-learn 2.0 cross-validation train-test-split into two parts: 5/6 of the data was used for learning and 1/6 was used for testing.
Using this method, we have achieved an average F-Score of 76.33% and an average accuracy of 63.38%. Although a good F-Score using this algorithm was received, the accuracy is not as high. One of the main reasons may be that some of the created patterns were very different from all other patterns and the distance between them and the other patterns cannot be measured, so the algorithm randomly classifies them as comparative sentences. Another reason for the mediocre performance of this algorithm is its focus on the HFW, whereas its drawback is by putting CW's into low priority. Many words that are very important in comparative sentences are CW's, and therefore this method may be good while seeking sarcastic sentences, but not as helpful for comparison sentences. Since using the second Algorithm yielded better results, we decided it is more suitable for our problem. Therefore, we did not proceed with this method and did not try different classification methods.

	Precision	Recall	Accuracy	F-Score
1.	73.00%	88.00%	68.00%	79.00%
2.	61.00%	92.00%	58.00%	73.00%
3.	72.00%	88.00%	66.00%	79.00%
4.	62.00%	91.00%	60.00%	73.00%
5.	70.00%	92.00%	67.00%	79.00%
6.	62.84%	93.00%	61.25%	75.00%
Average	66.81%	90.67%	63.38%	76.33%

4.2 Sequence-feature algorithm results:

The precision, recall, accuracy and F-score results for different techniques used are presented below in percentages. We randomly divided the data-set using the Scikit-learn 2.0 cross-validation train-test-split into two parts: 5/6 of the data was used for learning and 1/6 was used for testing. 6 iterations were made, while for each iteration we split the data, performed machine learning on the learning-set and tested each sentence in the test set.
The following results are the average of all 6 iterations. We used Scikit-learn 2.0 Python functions for classifications, the GaussianNB function for NB classification, LinearSVC function for linear SVM classification SVC function for non-linear SVM classification. Default parameters were used in all of these functions.
A detailed description of our experiments is as follows:

1) Using the NB classifier in order to classify the sentences to comparative and non-comparative.
2) Using the SVM classifier with a linear kernel function in order to classify the sentences to comparative and non-comparative.
3) Using the SVM classifier with rbf kernel function in order to classify the sentences to comparative and non-comparative.
4) Applying CSR to the learning-set in order to exclude sequences with low support or confidence. Then, we used the NB classifier in order to classify the sentences to comparative and non-comparative.
5) Applying CSR to the learning-set in order to exclude sequences with low support or confidence. Then, we used the SVM classifier with linear kernel function in order to classify the sentences to comparative and non-comparative.
6) Applying CSR to the learning-set in order to exclude sequences with low support or confidence. Then, we used the SVM classifier with rbf kernel function in order to classify the sentences to comparative and non-comparative.

GaussianNB (Gaussian Naive Bayes): implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian.

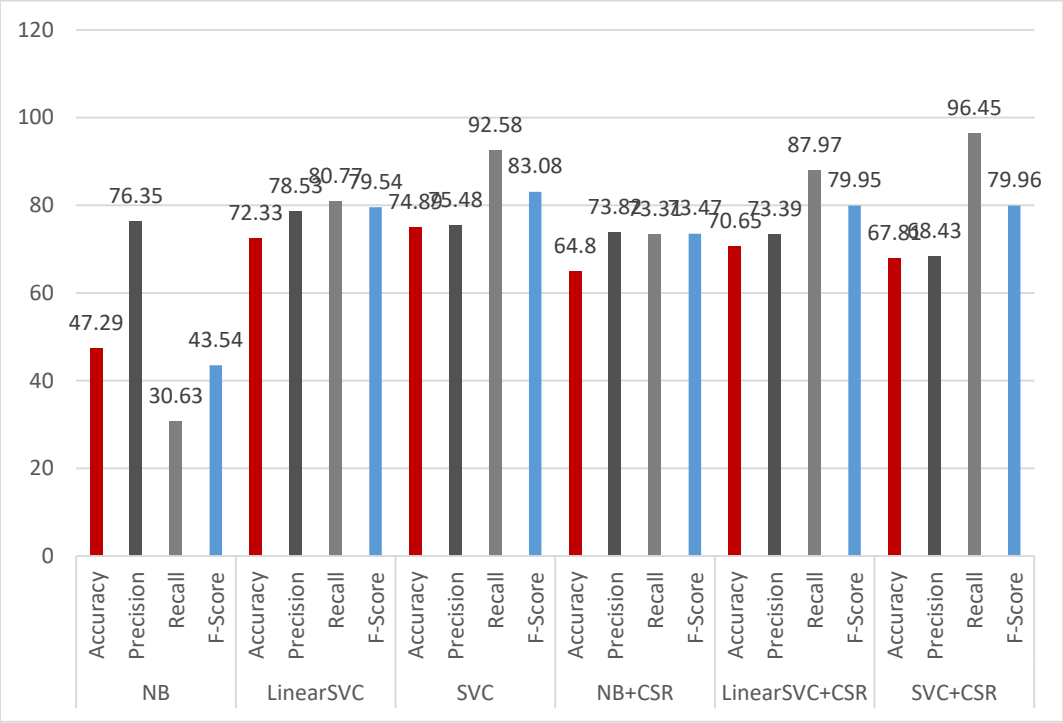
LinearSVC (Linear Support Vector Classification): Similar to SVC with parameter kernel='linear' $\langle x, x' \rangle$, but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties, function loss and should scale better (to large numbers of samples). We used the default penalty parameter of the error term which is C=0.1. We tried setting different values to the penalty parameter but did not get better results.

SVC (C-Support Vector Classification): The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples, which makes it hard to scale to dataset with more than a couple of 10000 samples. We used the default kernel type function in the algorithm which is kernel='rbf' $(-\gamma|x - x'|^2)$. We used the default value of the gamma parameter which is 3, the gamma parameter specified the γ parameter in the kernel function. We used the default penalty parameter of the error term which is C=0.1. We tried setting different values to the penalty parameter but did not get better results.

Results using 1-grams method in the vectors creating stage:

1. NB	Accuracy	49.06%	4. NB & CSR	Accuracy	64.79%
	Precision	85.11%		Precision	74.14%
	Recall	32.07%		Recall	73.48%
	F-Score	43.11%		F-Score	73.77%
2. LinearSVM	Accuracy	73.75%	5. LinearSVM & CSR	Accuracy	71.15%
	Precision	77.18%		Precision	73.74%
	Recall	86.77%		Recall	88.92%
	F-Score	81.62%		F-Score	80.59%
3. SVC	Accuracy	73.33%	6. SVC & CSR	Accuracy	71.67%
	Precision	76.86%		Precision	74.54%
	Recall	86.63%		Recall	88.17%
	F-Score	81.34%		F-Score	80.73%

Results using 1-2-range-grams method in the vectors creating stage:



1. NB	Accuracy	47.29%	4. NB & CSR	Accuracy	64.80%
	Precision	76.35%		Precision	73.82%
	Recall	30.63%		Recall	73.31%
	F-Score	43.54%		F-Score	73.47%
2. LinearSVM	Accuracy	72.33%	5. LinearSVM & CSR	Accuracy	70.65%
	Precision	78.53%		Precision	73.39%
	Recall	80.77%		Recall	87.97%
	F-Score	79.54%		F-Score	79.95%
3. SVC	Accuracy	74.89%	6. SVC & CSR	Accuracy	67.81%
	Precision	75.48%		Precision	68.43%
	Recall	92.58%		Recall	96.45%
	F-Score	83.08%		F-Score	79.96%

Better results were received using 1-2-range-gram method than the 1-gram method in the vector creating stage. The reason for this is that in comparative sentences, the word order in the sentence is also important. By using bi-grams and not only one-grams during the vector creating stage, we pay attention to word couples in the text, which constitute the order of the words in the sentence. The best results were achieved by using the nonlinear SVM classifier with rbf kernel function alone, and by using 1-2-range-gram method in the vector creating stage (F-Score of 83.08% and Accuracy of .7489%).

On one hand, we can see a very good improvement of the algorithm's performance by using the NB classifier combined with the CSR method. By using only NB classifier we achieved the accuracy of 47.29% and F-score of 43.54%, but by using NB classifier combined with the CSR method we achieved the accuracy of 64.80% and F-score of 73.47%. On the other hand, we can see deterioration in performance of the algorithm using nonlinear SVM classifier with CSR method. By using only nonlinear SVM classifier we achieved the accuracy of 74.89% and F-score of 83.08%, but by using nonlinear SVM classifier combined with the CSR method we achieved the accuracy of 67.81% and F-score of 79.96%.
One reason for this difference is the different nature of the two classifiers - while the SVM classifier ignores the noise vector, the NB classifier does not have this ability. Using the CSR method we simply remove the noise vectors from the training set. Another reason for this lack of improvement may be the fact that we use very low values for minimum confidence and minimum support in the CSR method. However, trying higher values did not improved our result. One of the hypothesis for improving the test result is trying to implement higher values for minimum confidence and minimum support while using a bigger data-set.

5 Conclusion

This paper suggested studying the problem of identifying comparative sentences in restaurant reviews at the sentence-level. We presented two algorithms for solving this problem and experimented them with a collection of 1037 reviews extracted from http://www.yelp.com. The collection contained reviews for different restaurants written by different reviewers and with different scales.

The first Algorithm presented on this paper, pattern-based algorithm, is based on the Semi-supervised Algorithm for Sarcasm Identification, whilst the second algorithm, sequence-feature algorithm, is based on an algorithm for identifying comparative sentences in text document. Using the first algorithm, we have achieved an average F-Score of 76.33% and an average accuracy of 63.38%. Better results was achieved by using the second algorithm, with a F-Score of 83.08% and an accuracy of .7489%. The base reason for this is the different nature of the two algorithms. The first algorithm focused on the HFW and put the CW's into low priority, whereas the second algorithm did not put different priorities on different types of words. Since many words in comparative sentences are considered important, this method was found to be better for seeking sarcastic sentences, but not as helpful while looking for comparison sentences. Due to the drawback of the first algorithm our main focus was on improving the second.

Empirical evaluation using diverse text data sets demonstrated that our second algorithm is indeed effective. Also, the performance of our system surpassed the algorithm of Jindal & Liu (2006). As the original method found only 18% of the sentences comparing between two restaurants, we received a F-Score of 83.08% and Accuracy of .7489%. That is to say that our system indeed recognizes comparative sentences between two restaurants that do not contain any comparative word.

In future works, we suggest improving both the precision and recall of the proposed technique, as well as studying and implementing categorization methods, in order to improve the classification of comparative sentences into different comparison types.

6 References

Blanken, H. M., De Vries, A. P., Blok, H. E., & Feng, L. (2007). Data-Centric Systems and Applications. *Multimedia Retrieval*. http://doi.org/10.1007/978-3-662-10876-5_5

Bril, E. (1995). Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Stady in Part-of-Speech. *Journal Computational Linguistics Volume 21 543-565.*

Dai, W., Jin, G. Z., Lee, J., & Luca, M. (2012). Optimal Aggregation of Consumer Ratings: an Application to Yelp.com. *NBER Working Paper*, *18567*. http://doi.org/10.3386/w18567

Davidov, D., & Rappoport, A. (2006). *Efficient unsupervised discovery of word categories using symmetric patterns and high frequency words.* Proceeding ACL-44 Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics, 297–304. http://doi.org/10.3115/1220175.1220213

Davidov, D., Tsur, O., & Rappoport, A. (2010). *Semi-Supervised Recognition of Sarcastic Sentences in Twitter and Amazon.* Fourteenth Conference on Computational Natural Language Learning, (July), 107–116. http://doi.org/10.1145/1964858.1964874

Etzioni, O., Cafarella, M., Downey, D., Popescu, A. M., Shaked, T., Soderland, S., Weld, D.S., Yates, A. (2005). Unsupervised named-entity extraction from the Web: An experimental study. *Artificial Intelligence*, *165*(1), 91–134. http://doi.org/10.1016/j.artint.2005.03.001

Gamon, M., Gamon, M., Aue, A., Aue, A., Corston-Oliver, S., Corston-Oliver, S., Ringger, E. (2005). Pulse: Mining customer opinions from free text. *Lecture Notes in Computer Science*, *3646*, 121–132. http://doi.org/10.1007/11552253_12

Ganapathibhotla, M., & Liu, B. (2008). Mining opinions in comparative sentences. *Proceedings of the 22nd International Conference on Computational Linguistics - COLING '08*, *1*, 241–248. http://doi.org/10.3115/1599081.1599112

Id, O. (2009). Online edition (c) 2009 Cambridge UP. *Information Retrieval*, (c), 1–18. http://doi.org/10.1109/LPT.2009.2020494

Jindal, N., & Liu, B. (2006). *Identifying comparative sentences in text documents.* Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '06, 244. http://doi.org/10.1145/1148170.1148215

Jindal, N., Liu, B., & Bing, L. (2006). *Mining comparative sentences and relations*. Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '06, *21*(2), 1331–1336. http://doi.org/10.1107/S0108270189000326

Kennedy, C. (2004). Comparatives , Semantics of ∗, 1–9.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, *19*(2), 313–330. http://doi.org/10.1162/coli.2010.36.1.36100

Park, D., & Blake, C. (2012). Identifying comparative claim sentences in full-text scientific articles. *... of the Workshop on Detecting Structure in Scholarly ...*, 1–9. Retrieved from http://dl.acm.org/citation.cfm?id=2391173

Santorini, B. (1990). Part-of-Speech Tagging Guidelines for the Penn Treebank Project (3rd Revision). *University of Pennsylvania 3rd Revision 2nd Printing*, (MS-CIS-90-47), 33. Retrieved from http://www.personal.psu.edu/faculty/x/x/xxl13/teaching/sp07/apling597e/resources/Tagset.pdf

Tsur, O., Rappoport, A., & Davidov, D. (2010). I*CWSM – A Great Catchy Name: Semi-Supervised Recognition of Sarcastic Sentences in Online Product Reviews.* Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media, (9), 162–169.

Xu, K., Liao, S., Li, J., & Song, Y. (2011). Mining comparative opinions from customer reviews for Competitive Intelligence. *Decision Support Systems*, *50*(4), 743–754. http://doi.org/10.1016/j.dss.2010.08.021

Yang, S., & Ko, Y. (2009). *Extracting Comparative Sentences from Korean Text Documents Using Comparative Lexical Patterns and Machine Learning Techniques*. Proceedings of the ACL-IJCNLP 2009 Conference Short Papers, , 153–156. http://doi.org/10.3115/1667583.1667631

Yang, S., & Ko, Y. (2011). *Extracting Comparative Entities and Predicates from Texts Using Comparative Type Classification.* Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1, 1636–1644. Retrieved from http://dl.acm.org/citation.cfm?id=2002472.2002668

7 Appendix

7.1 Data Creation

```
#Training set construction
import sql_func
import exel_func
import MySQLdb
import nltk

#getting comparative sentence from a review
def Get_Comp_Sentence( Data, Key ):
    Sentences = nltk.sent_tokenize( Data )
    for Sentence in Sentences:
        location = Sentence.find( Key )
        if location >= 0:
            Words = nltk.word_tokenize(Sentence.replace(Key,"rest"))
            for Word in Words:
                if Word == "rest":
                    return Sentence
    return None

connection  =
MySQLdb.connect(host="127.0.0.1",user="root",passwd="123456",db="yelp")
cur = connection.cursor()

indexses=sql_func.CreateIndexs(cur)
nameToRestIndex=indexses[0]
restToNameIndex=indexses[1]

Review_Ids = exel_func.Get_Review_Ids()
print Review_Ids

workbook=exel_func.CreateExelWorkbook()

index=0
count=0
#searching comparative sentences in the reviewz
while (index < sql_func.GetRowCount(cur) and count<=1000):
    for row in sql_func.GetRevies(index,cur) :
        if row[0] not in Review_Ids:
            for key in nameToRestIndex.keys():
                location=row[5].find(key)
                if(location>=0 and nameToRestIndex[key]!=row[2] and
restToNameIndex.has_key(row[2])):
                    Sentence = Get_Comp_Sentence( row[5], key )
                    if Sentence is not None:
                        count+=1
                        #writing the data to excel

exel_func.AddDataColumn(workbook.worksheets()[0],count,row, restToNameIndex,
nameToRestIndex[key],workbook.formats[0],Sentence)
                        break
    index+=1000
    print "index: "+str(index)+" count: "+str(count)
```

```
workbook.close()
cur.close()
connection.close()
print "END"
```

7.2 Pattern-based Algorithm

```
#This algorithm is looking statement comparing restaurants in Restaurant
Reviews
import itertools
import exel_func
import nltk
import regex
from sklearn.cross_validation import train_test_split

#punctuation list to remove from the selected sentences
Punctuation_List = [".",",","!","?",";","...",":","$","(",")","-","--",'"']

#list of stop words
Stop_Words = ['a', 'about', 'above', 'across', 'after', 'afterwards',
'again', 'against', 'all', 'almost', 'alone', 'along', 'already', 'also',
'although', 'always', 'am', 'among', 'amongst', 'amount', 'an', 'and',
'another', 'any', 'anyhow', 'anyone', 'anything', 'anyway', 'anywhere',
'are', 'around', 'as', 'at', 'back', 'be', 'became', 'because', 'become',
'becomes', 'becoming', 'been', 'before', 'beforehand', 'behind', 'being',
'below', 'beside', 'besides', 'between', 'beyond', 'bill', 'both', 'bottom',
'but', 'by', 'call', 'can', 'cannot', 'cant', 'co', 'computer', 'con',
'could', 'cry', 'de', 'describe', 'detail', 'do', 'done', 'down', 'due',
'during', 'each', 'eg', 'eight', 'either', 'eleven', 'else', 'elsewhere',
'empty', 'enough', 'etc', 'even', 'ever', 'every', 'everyone', 'everything',
'everywhere', 'except', 'few', 'fifteen', 'fill', 'find', 'fire', 'first',
'five', 'for', 'former', 'formerly', 'forty', 'found', 'four', 'from',
'front', 'full', 'further', 'get', 'give', 'go', 'had', 'has', 'hasnt',
'have', 'he', 'hence',  'her', 'here', 'hereafter', 'hereby', 'herein',
'hereupon', 'hers', 'him', 'his', 'how', 'however', 'hundred', 'i', 'ie',
'if', 'in', 'inc', 'indeed', 'interest', 'into', 'is', 'it', 'its' , 'keep',
'last', 'latter', 'latterly', 'least', 'less', 'ltd', 'made', 'many', 'may',
'me', 'meanwhile', 'might', 'mill', 'mine', 'more', 'moreover', 'most',
'mostly', 'move', 'much', 'must', 'my', 'name', 'namely', 'neither', 'never',
'nevertheless', 'next', 'nine', 'no', 'nobody', 'none', 'nor', 'not',
'nothing', 'now', 'nowhere', 'of', 'off', 'often', 'on', 'once', 'one',
'only', 'onto', 'or', 'other', 'others', 'otherwise', 'our', 'ours',
'ourselves', 'out', 'over', 'own', 'part', 'per', 'perhaps', 'please', 'put',
'rather', 're', 'same', 'see', 'seem', 'seemed', 'seeming', 'seems',
'serious', 'several', 'she', 'should', 'show', 'side', 'since', 'sincere',
'six', 'sixty', 'so', 'some', 'somehow', 'someone', 'something', 'sometime',
'sometimes', 'somewhere', 'still', 'such', 'system', 'take', 'ten', 'than',
'that', 'the', 'their', 'them', 'themselves', 'then', 'thence', 'there',
'thereafter', 'thereby', 'therefore', 'therein', 'thereupon', 'these',
'they', 'thick', 'thin', 'third', 'this', 'those', 'though', 'three',
'through', 'throughout', 'thru', 'thus', 'to', 'together', 'too', 'top',
'toward', 'towards',  'twelve', 'twenty', 'two', 'un', 'under', 'until',
'up', 'upon', 'us', 'very', 'via', 'was', 'we', 'well', 'were', 'what',
'whatever', 'when', 'whence', 'whenever', 'where', 'whereas', 'whereby',
```

```
'wherein', 'whereupon', 'wherever', 'whether', 'which', 'while', 'whither',
'who', 'whoever', 'whole', 'whom', 'whose', 'why', 'will', 'with', 'within',
'without', 'would', 'yet', 'you', 'your', 'yours', 'yourself', 'yourselves']

Reviews = exel_func.Get_Review()
Sets = train_test_split( Reviews, test_size=0.16 )
Patterns = []

#creates patterns from sentences
for Review in Sets[0]:

    Pattern_Index=[]
    Rest_Index = -1
    First_Index = -1
    Last_Index = -1
    Words = nltk.word_tokenize(Review[6].replace(Review[4],"rest"))

    Index = 0
    for Word in Words:
        if Word == "rest":
            Rest_Index = Index
        Index += 1

    Index = 0
    for Word in Words:
        if Word in Stop_Words:
            if Index < (Rest_Index - 1):
                First_Index = Index
            if Index > (Rest_Index + 1):
                Last_Index = Index
                break
        Index += 1

    if First_Index == -1:
        First_Index = Rest_Index

    if Last_Index == -1:
        Last_Index = Rest_Index

    print "First_Index: " + str(First_Index)
    print "Last_Index: " +str(Last_Index)

    #creates the different regex for every pattern
    #Pattern_Index stores the pattern regex and distance from the pattern
    if First_Index != Last_Index and Last_Index - First_Index < 10 and
Rest_Index > -1:
        Index = 0
        List = []
        for Word in Words:
            if First_Index <= Index and Index <= Last_Index:
                if Rest_Index == Index :
                    List.append('rest')
                elif Word in Stop_Words:
                    List.append( Word )
                else:
                    List.append( 'cw' )
            Index += 1
```

```
        Pattern1=""
        Pattern2=""
        for word in List:
            Pattern1+=exel_func.Get_Word_Regex(word)+"+s?[, ]\s?"
            Pattern2+=exel_func.Get_Word_Regex(word)+"+(s?[,
]\s?\w+){0,100}s?[, ]\s?"
        Pattern1=Pattern1[:-10]
        Pattern2=Pattern2[:-31]
        Pattern_Index.append((Pattern1, 1.0, Review[0], Review[7]))
        Pattern_Index.append((Pattern2, 0.1, Review[0], Review[7]))
        combs = []
        for i in range(len(List), 2, -1):
            els = [list(x) for x in itertools.combinations(List, i)]
            combs.extend(els)
        for comb in combs:
            Pattern=""
            Contain_Rest=False
            for word in comb:
                if word == "rest":
                    Contain_Rest = True
                Pattern+=exel_func.Get_Word_Regex(word)+"+(s?[,
]\s?\w+){0,100}s?[, ]\s?"
            if Contain_Rest:
                Pattern=Pattern[:-31]
                Pattern_Index.append((Pattern,(0.1*len(comb))/len(List),
Review[0], Review[7]))
        Patterns.append(Pattern_Index)

print "++++++++++++++++++LEN+++++++++++++++"
print len( Patterns )
print "++++++++++++++++++LEN+++++++++++++++"

workbook = exel_func.Create_Exel_Workbook()
worksheet = workbook.worksheets()[0]

tp = 0
fp = 0
fn = 0
tn = 0

#applying the knn classifier
index = 0
for Review in Sets[1]:
    Fixed_Review = Review[6].replace(Review[4], "rest")
    index += 1
    knn = dict()
    for Pattern_Index in Patterns:
        if Pattern_Index[0][2] != Review[0]:
            for Pattern in Pattern_Index:
                result = regex.search( Pattern[0],Fixed_Review )
                if result is not None:
                    if Pattern[1] in knn:
                        knn[Pattern[1]].append(Pattern[3])
                    else:
                        knn[Pattern[1]] = [Pattern[3]]
                    break
```

```
    print index
    print knn
    Sum = 0
    Pos = 0
    Neg = 0
    for Item in sorted(knn):
        if Sum <= 12:
            Sum += len(knn[Item])
            for Score in knn[Item]:
                if knn[Item] == 0:
                    Neg += Item
                else:
                    Pos += Item
    Match = 1 if Pos > Neg else 0
    print Match
    print Review[7]
    print "-------------------------------------------------------------"
    #tp
    tp += 1 if (Match == Review[7] and Match == 1) else 0
    #fp
    fp += 1 if (Match != Review[7] and Match == 1) else 0
    #fn
    fn += 1 if (Match != Review[7] and Match == 0) else 0
    #tn
    tn += 1 if (Match == Review[7] and Match == 0) else 0

#calculates the results
Precision = (tp * 100.00)/(tp+fp)
Recall = (tp * 100.00)/(tp+fn)
Accuracy = ((tp + tn) * 100.00)/ (tp+fp+tn+fn)
FScore = (2.0*Precision*Recall)/(Precision+Recall)

#writes results to excel
worksheet.write( 0,0,"Precision" )
worksheet.write( 1,0,"Recall")
worksheet.write( 2,0,"Accuracy" )
worksheet.write( 3,0,"FScore" )

worksheet.write( 0, 2, Precision )
worksheet.write( 1, 2, Recall )
worksheet.write( 2, 2, Accuracy )
worksheet.write( 3, 2, FScore )

workbook.close()
print "END"
```

7.3 Sequence Feature Algorithm

```
#This algorithm is looking statement comparing restaurants in Restaurant
Reviews
```

```
import nltk
import exel_func
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.cross_validation import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

#Calculates the precision
def CalcPrecision(Summary):
    return (Summary['TP']*100.00)/(Summary['TP']+Summary['FP'])

#Calculates the accuracy
def CalcTrueResult(Summary):
    return
((Summary['TP']+Summary['TN'])*100.00)/(Summary['TP']+Summary['FP']+Summary['
TN']+Summary['FN'])

#Calculates the recall
def CalcRecall(Summary):
    return (Summary['TP']*100.00)/(Summary['TP']+Summary['FN'])

#Calculates the F-Score
def CalcFScore(Summary):
    Precision = CalcPrecision(Summary)
    Recall = CalcRecall(Summary)
    return (2.0*Precision*Recall)/(Precision+Recall)

# updates the summary for calculation
def UpdateSummary(Class,Predict,Summary):
    if Class == Predict:
        if Class == 1:
            Summary['TP']+=1
        else:
            Summary['TN']+=1
    else:
        if Class == 1:
            Summary['FN']+=1
        else:
            Summary['FP']+=1

#writes the results to excel
def AddData(worksheet,X,Y,Summary):
    Precision = CalcPrecision(Summary)
    Recall = CalcRecall(Summary)
    FScore = CalcFScore(Summary)
    TrueResult = CalcTrueResult(Summary)
    worksheet.write_number(X,Y,TrueResult)
    worksheet.write_number(X,Y+1,Precision)
    worksheet.write_number(X,Y+2,Recall)
    worksheet.write_number(X,Y+3,FScore)

#writes excel headers
def AddSubHeaderColumn(worksheet,X,Y,Name):
    worksheet.write(X,Y,Name)
    worksheet.write(X+1,Y,"Accuracy")
    worksheet.write(X+1,Y+1,"Precision")
    worksheet.write(X+1,Y+2,"Recall")
```

```
    worksheet.write(X+1,Y+3,"F-Score")
    worksheet.merge_range(X,Y,X,Y+3,Name)

#writes excel column header
def AddHeaderColumn(worksheet):
    worksheet.write(0,0,"TotalItems")
    worksheet.write(1,0,"TestItems")
    worksheet.write(2,0,"SentLen")
    worksheet.write(0,2,"MinSup")
    worksheet.write(1,2,"MinConf")

    worksheet.write(3,0,"id")
    AddSubHeaderColumn(worksheet,3,1,"NB")
    AddSubHeaderColumn(worksheet,3,5,"LinearSVC")
    AddSubHeaderColumn(worksheet,3,9,"SVC")
    AddSubHeaderColumn(worksheet,3,13,"NuSVC")
    AddSubHeaderColumn(worksheet,3,17,"NB+CSR")
    AddSubHeaderColumn(worksheet,3,21,"LinearSVC+CSR")
    AddSubHeaderColumn(worksheet,3,25,"SVC+CSR")
    AddSubHeaderColumn(worksheet,3,29,"NuSVC+CSR")

#punctuation list to remove from the selected sentences
Punctuation_List = [".",",","!","?",";","...",":","$","(",")","-","--
",'"',"``","#","'",'-NONE-']

#vectorizes
vectorizer = CountVectorizer(ngram_range=(1, 2), token_pattern=r'\b\w+\b',
min_df=1)

Delta = 3
Cicle = 50
MinSup = 10
MinConf = 10

Count = 0
Test_Set = []

#gets the sentences that suspected as comparative from the manually labeled
DS
for Review in exel_func.Get_Review():
    Rest_Index = -1
    Index = 0
    Text = Review[6].replace(Review[4],"rest")
    #converting the words in the sentences to POS tags
    POSS = [POS for POS in nltk.pos_tag(nltk.word_tokenize(Text)) if POS[0]
not in Punctuation_List and POS[1] not in Punctuation_List and POS[1] is not
None]

    for POS in POSS:
        if POS[0] == "rest":
            Rest_Index = Index
        Index += 1

    #use the words that are within the radios of 3 words of the restaurant
    Index = 0
    List=[]
    for POS in POSS:
```

```
        if ( Rest_Index - Delta ) <= Index and Index <= ( Rest_Index + Delta
):
            if Rest_Index == Index :
                List.append('Rest')
            else:
                List.append(POS[1])
        Index += 1
    if Rest_Index > -1:
        Test_Set.append( ( List, Review[7], Count ) )
        Count +=1

Dict = {}
for Item in Test_Set:
    for i in range(0, len(Item[0])):
        if Item[0][i] != "Rest":
            pos = 0
            neg = 0

            if Item[1] == 1:
                pos = 1
            else:
                neg = 1

            if not Dict.has_key(Item[0][i]):
                Dict[Item[0][i]] = [ pos, neg ]
            else:
                Dict[Item[0][i]][0] += pos
                Dict[Item[0][i]][1] += neg

#building help data for the CSR rules
Dict2={}
for Item in Dict.iterkeys():
    Pos = Dict[Item][0]
    Neg = Dict[Item][1]
    Total = len( Test_Set )
    IvalPos = (Pos*100)/Total < MinSup or ( Pos*100 )/( Pos+Neg ) < MinConf
    IvalNeg = (Neg*100)/Total < MinSup or ( Neg*100 )/( Pos+Neg ) < MinConf
    if IvalPos or IvalNeg:
        Dict2[Item] = ( IvalPos, IvalNeg )

Temp = []
for Item in Test_Set:
    Doc = '';
    for POS in Item[0]:
        Doc += POS;
        Doc += ' ';
    Temp.append( Doc )

#converts the sentences to vectors
Temp2 = vectorizer.fit_transform( Temp )
Temp3 = Temp2.toarray()

Vector_Set=[]
for i in range( 0, len( Test_Set ) ):
    Vector_Set.append( [ Temp3[i], Test_Set[i][1], Test_Set[i][2] ] )

workbook=exel_func.Create_Exel_Workbook()
```

```
Size = 0

#classifies the sentences to comparative an non-comparative
for j in range(0, Cicle):
    print j
    Sets = train_test_split( Vector_Set, test_size=0.16 )

    #classifier creation
    clfNB = GaussianNB()
    clfLinearSVC = svm.LinearSVC()
    clfSVC = svm.SVC()
    clfNuSVC = svm.NuSVC()

    clfNBCSR = GaussianNB()
    clfLinearSVCCSR = svm.LinearSVC()
    clfSVCCSR = svm.SVC()
    clfNuSVCCSR = svm.NuSVC()

    #classifier training
    clfNB.fit([i[0] for i in Sets[0]], [i[1] for i in Sets[0]])
    clfLinearSVC.fit([i[0] for i in Sets[0]], [i[1] for i in Sets[0]])
    clfSVC.fit([i[0] for i in Sets[0]], [i[1] for i in Sets[0]])
    clfNuSVC.fit([i[0] for i in Sets[0]], [i[1] for i in Sets[0]])

    #applying the SCR rules to the sentences
    List=[]
    for Item2 in Test_Set:
        for Item1 in Sets[0]:
            if Item2[2] == Item1[2]:
                for SubItem in Item2[0]:
                    if Dict2.has_key(SubItem):
                        if Dict2[SubItem][0] and Item2[1] == 1:
                            List.append(Item1[2])
                        if Dict2[SubItem][1] and Item2[1] == 0:
                            List.append(Item1[2])
    Sets1 = [x for x in Sets[0] if x[2] not in List]

    #classifier training
    clfNBCSR.fit([i[0] for i in Sets1], [i[1] for i in Sets1])
    clfLinearSVCCSR.fit([i[0] for i in Sets1], [i[1] for i in Sets1])
    clfSVCCSR.fit([i[0] for i in Sets1], [i[1] for i in Sets1])
    clfNuSVCCSR.fit([i[0] for i in Sets1], [i[1] for i in Sets1])

    SummaryNB = {'TP':0,'FP':0,'FN':0,'TN':0}
    SummaryLinearSVC = {'TP':0,'FP':0,'FN':0,'TN':0}
    SummarySVC = {'TP':0,'FP':0,'FN':0,'TN':0}
    SummaryNuSVC = {'TP':0,'FP':0,'FN':0,'TN':0}

    SummaryNBCSR = {'TP':0,'FP':0,'FN':0,'TN':0}
    SummaryLinearSVCCSR = {'TP':0,'FP':0,'FN':0,'TN':0}
    SummarySVCCSR = {'TP':0,'FP':0,'FN':0,'TN':0}
    SummaryNuSVCCSR = {'TP':0,'FP':0,'FN':0,'TN':0}

    Size = len(Sets[1])

    for i in range(0, len(Sets[1])):
        #getting the classification prediction
```

```
            ResultNB = clfNB.predict([Sets[1][i][0]])
            ResultLinearSVC = clfLinearSVC.predict([Sets[1][i][0]])
            ResultSVC = clfSVC.predict([Sets[1][i][0]])
            ResultNuSVC = clfNuSVC.predict([Sets[1][i][0]])

            ResultNBCSR = clfNBCSR.predict([Sets[1][i][0]])
            ResultLinearSVCCSR= clfLinearSVCCSR.predict([Sets[1][i][0]])
            ResultSVCCSR = clfSVCCSR.predict([Sets[1][i][0]])
            ResultNuSVCCSR = clfNuSVCCSR.predict([Sets[1][i][0]])

            #updating the result summaries
            UpdateSummary(Sets[1][i][1],ResultNB,SummaryNB)
            UpdateSummary(Sets[1][i][1],ResultLinearSVC,SummaryLinearSVC)
            UpdateSummary(Sets[1][i][1],ResultSVC,SummarySVC)
            UpdateSummary(Sets[1][i][1],ResultNuSVC,SummaryNuSVC)

            UpdateSummary(Sets[1][i][1],ResultNBCSR,SummaryNBCSR)
            UpdateSummary(Sets[1][i][1],ResultLinearSVCCSR,SummaryLinearSVCCSR)
            UpdateSummary(Sets[1][i][1],ResultSVCCSR,SummarySVCCSR)
            UpdateSummary(Sets[1][i][1],ResultNuSVCCSR,SummaryNuSVCCSR)

        #write results to excel
        (workbook.worksheets()[0]).write_number(5+j,0,j)
        AddData(workbook.worksheets()[0],5+j,1,SummaryNB)
        AddData(workbook.worksheets()[0],5+j,5,SummaryLinearSVC)
        AddData(workbook.worksheets()[0],5+j,9,SummarySVC)
        AddData(workbook.worksheets()[0],5+j,13,SummaryNuSVC)

        AddData(workbook.worksheets()[0],5+j,17,SummaryNBCSR)
        AddData(workbook.worksheets()[0],5+j,21,SummaryLinearSVCCSR)
        AddData(workbook.worksheets()[0],5+j,25,SummarySVCCSR)
        AddData(workbook.worksheets()[0],5+j,29,SummaryNuSVCCSR)

#write results to excel
AddHeaderColumn(workbook.worksheets()[0])
(workbook.worksheets()[0]).write_number(0,1,len( Test_Set ))
(workbook.worksheets()[0]).write_number(1,1,Size)
(workbook.worksheets()[0]).write_number(2,1,(2*Delta)+1)
(workbook.worksheets()[0]).write_number(0,3,MinSup)
(workbook.worksheets()[0]).write_number(1,3,MinConf)

workbook.close()
print "END"
```

7.4 Excel help library

```
#Help function that work with excel
import xlsxwriter
import tkFileDialog
import nltk.data
import xlrd
```

```
import itertools

tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')

def AddHeaderColumn(worksheet):
    worksheet.write(0,0,"id")
    worksheet.write(0,1,"user")
    worksheet.write(0,2,"rest1-id")
    worksheet.write(0,3,"rest1-name")
    worksheet.write(0,4,"rest2-id")
    worksheet.write(0,5,"rest2-name")
    worksheet.write(0,6,"rating")
    worksheet.write(0,7,"date")
    worksheet.write(0,8,"text")
    worksheet.write(0,9,"comp-sentence")
    worksheet.write(0,10,"match")

def GetCompSentence(text, rest_name):
    data=tokenizer.tokenize(text)
    for record in data:
        location=record.find(" "+rest_name+" ")
        if location>=0:
            return record.decode('UTF-8')
    print rest_name
    for record in data:
        print record.find(rest_name)
        print record
    print "-----------------------------------"


def AddDataColumn(worksheet, rowNum, data, restToNameIndex, restId,
date_format, Sentence):
    worksheet.write(rowNum,0,data[0])
    worksheet.write(rowNum,1,data[1])
    worksheet.write(rowNum,2,data[2])
    worksheet.write(rowNum,3,restToNameIndex[data[2]].decode('UTF-8'))
    worksheet.write(rowNum,4,restId)
    worksheet.write(rowNum,5,restToNameIndex[restId].decode('UTF-8'))
    worksheet.write(rowNum,6,data[3])
    worksheet.write(rowNum,7,data[4],date_format)
    worksheet.write(rowNum,8,data[5].decode('UTF-8'))
    worksheet.write(rowNum,9,Sentence.decode('UTF-8'))


def AddDataColumn2(worksheet, rowNum, restId,name,count):
    worksheet.write(rowNum,0,restId)
    worksheet.write(rowNum,1,name.decode('UTF-8'))
    worksheet.write(rowNum,2,count)

def CreateExelWorkbook():
    filename = tkFileDialog.asksaveasfilename()
    workbook = xlsxwriter.Workbook(str(filename)+".xlsx")
    worksheet = workbook.add_worksheet("results")
    workbook.add_format({'num_format': 'DD/MM/YY'})
    AddHeaderColumn(worksheet)
    return workbook
```

```
def GetPatterns():
    Patterns = []
    filename = tkFileDialog.askopenfilename()
    print filename
    workbook = xlrd.open_workbook(filename)
    worksheet=workbook.sheet_by_index(0)
    num_rows = worksheet.nrows - 1
    curr_row = -1
    while curr_row < num_rows:
        curr_row += 1
        if (worksheet.cell_value(curr_row, 10)==1):
            cell_value = worksheet.cell_value(curr_row, 12).lower()
            if cell_value:
                Pattern_Index=[]
                Pattern_Index.append((cell_value,))
                sentence=nltk.word_tokenize(cell_value)
                Pattern1=""
                Pattern2=""
                sentence.remove("[")
                sentence.remove("]")

                for word in sentence:
                    Pattern1+=Get_Word_Regex(word)+"+s?[, ]\s?"
                    Pattern2+=Get_Word_Regex(word)+"+(s?[,
]\s?\w+){0,100}s?[, ]\s?"
                Pattern1=Pattern1[:-10]
                Pattern2=Pattern2[:-31]
                Pattern_Index.append((Pattern1, 1))
                Pattern_Index.append((Pattern2, 0.1))
                combs = []
                for i in range(len(sentence), 2, -1):
                    els = [list(x) for x in itertools.combinations(sentence,
i)]
                    combs.extend(els)
                for comb in combs:
                    Pattern=""
                    Contain_Rest=False
                    for word in comb:
                        if word == "rest":
                            Contain_Rest=True
                        Pattern+=Get_Word_Regex(word)+"+(s?[,
]\s?\w+){0,100}s?[, ]\s?"
                    if Contain_Rest:
                        Pattern=Pattern[:-31]

Pattern_Index.append((Pattern,(0.1*len(comb))/len(sentence)))
                Patterns.append(Pattern_Index)
    return Patterns

def Get_Review():
    Reviews=[]
    filename = tkFileDialog.askopenfilename()
    workbook = xlrd.open_workbook(filename)
    worksheet=workbook.sheet_by_index(0)
    num_rows = worksheet.nrows - 1
    curr_row = 0
```

```
    while curr_row < num_rows:
        curr_row += 1
        if worksheet.cell_value(curr_row, 12) != 1:
            #0 Id
            Id = worksheet.cell_value(curr_row, 0)
            #1 Rest1_Id
            Rest1_Id = worksheet.cell_value(curr_row, 2)
            #2 Rest1_Name
            Rest1_Name = worksheet.cell_value(curr_row, 3)
            #3 Rest2_Id
            Rest2_Id = worksheet.cell_value(curr_row, 4)
            #4 Rest2_Name
            Rest2_Name = worksheet.cell_value(curr_row, 5)
            #5 Review
            Review = worksheet.cell_value(curr_row, 8)
            #6 Santence
            Santence = worksheet.cell_value(curr_row, 9)
            #7 Match
            Match = worksheet.cell_value(curr_row, 10)
            Reviews.append((Id, Rest1_Id, Rest1_Name, Rest2_Id, Rest2_Name,
Review, Santence, Match))

    return Reviews

def Get_Word_Regex(word):
    if word=="cw":
        return "\w";
    else:
        return "\\b"+word+"\\b"

def CreateExelWorkbook2():
    filename = tkFileDialog.asksaveasfilename()
    workbook = xlsxwriter.Workbook(str(filename)+".xlsx")
    worksheet = workbook.add_worksheet("results")
    worksheet.write(0,0,"id")
    worksheet.write(0,1,"name")
    worksheet.write(0,2,"count")
    return workbook

def Create_Exel_Workbook():
    filename = tkFileDialog.asksaveasfilename()
    workbook = xlsxwriter.Workbook(str(filename)+".xlsx")
    worksheet = workbook.add_worksheet("results")
    return workbook

def Add_Excel_Column_Caption_For_Output(worksheet):
    worksheet.write(0,0,"id")
    worksheet.write(0,1,"rest1-id")
    worksheet.write(0,2,"rest2-id")
    worksheet.write(0,3,"rest2-name")
    worksheet.write(0,4,"text")
    worksheet.write(0,5,"org-sentence")
    worksheet.write(0,6,"found-sentence")
    worksheet.write(0,7,"score")
    worksheet.write(0,8,"POS1")
    worksheet.write(0,9,"POS2")
    worksheet.write(0,10,"match")
```

```
def Add_Data_Row_For_Output(worksheet, rowNum, data, Rest, Org_Sentence,
Found_Sentence, Score, POS_Match):
    #id
    worksheet.write(rowNum,0,data[0])
    #rest1-id
    worksheet.write(rowNum,1,data[1])
    #rest2-id
    worksheet.write(rowNum,2,Rest[0])
    #rest2-name
    worksheet.write(rowNum,3,Rest[1])
    #text
    worksheet.write(rowNum,4,data[5])
    #org-sentence
    worksheet.write(rowNum,5,Org_Sentence)
    #found-sentence
    worksheet.write(rowNum,6,Found_Sentence)
    #score
    worksheet.write(rowNum,7,Score)
    if POS_Match is not None:
        #POS1
        worksheet.write(rowNum,8,POS_Match[0])
        #POS2
        worksheet.write(rowNum,9,POS_Match[1])
    #match
    worksheet.write(rowNum,10,data[7])

def Get_Review_Ids():
    Review_Ids = []
    filename = tkFileDialog.askopenfilename()
    print filename
    workbook = xlrd.open_workbook(filename)
    worksheet=workbook.sheet_by_index(0)
    num_rows = worksheet.nrows - 1
    curr_row = -1
    while curr_row < num_rows:
        curr_row += 1
        Review_Ids.append(worksheet.cell_value(curr_row, 0))
    return Review_Ids
```

7.5 SQL help library

```
#Help function that work with sql
import nltk
#excel cannot work with this name 1165,146->not ascii && not rest or doubles
listOfNoiseRestaurants = set([1082, 1322, 1511, 1333, 1272, 1027, 1502, 1478,
1113, 1025, 1165, 146,
```

```
                                  1328, 1507, 1154, 1073, 1074, 1254, 1111,
1057, 1132, 1338, 1100, 366,
                                   1484, 1294, 1237, 1151, 1660, 1308, 1202,
1052, 1029, 1388, 1203, 1177,
                                   1221, 2341, 1372, 1469, 904, 7390, 8215,
1175, 1036, 1394, 1624, 893, 955,
                                   11443, 11358, 1209, 1636, 5878, 892, 2514,
2950, 1615, 1501, 1649, 1396, 1058,
                                    4413, 7373, 1565, 1128, 1157, 1087, 1056,
8563, 1556, 1442, 1043, 1045, 1491,
                                    1135, 1164, 1431, 5175, 1231, 1149, 1329,
1196, 1197, 1134, 1023, 1086, 1085, 1291,
                                     6902, 1428, 1387, 1182, 3392, 1415, 1447,
1497, 1437, 3281, 1445, 1527, 3774, 1661,
                                     865, 1487, 3886, 1295, 14297, 6192, 3924,
1053, 5408, 5929, 2919, 1327, 4509, 1185,
                                      1546, 1513, 1318, 1605, 1672, 1273,
1159, 1324, 1160, 1233, 1403, 2223, 1435, 9821, 5007,
                                      4512, 1604, 1662, 1038, 2509, 1377,
1257, 1123, 2853, 6815, 1657, 1416, 5087, 1379,
                                       5252, 1414, 2477, 2468, 1458, 5407,
2843, 967, 1116, 1215, 1395, 1633, 1457, 1096,
                                       2465, 3359, 1125, 12571, 1467, 1462,
9436, 7251, 753, 1104, 2844, 6374, 1032, 971,
                                        1069, 1498, 1121, 9774, 1432, 1183,
2942, 1512, 1516, 12825, 1525, 2426, 943, 6134,
                                        1325, 1072, 1114, 1382, 5781, 7382,
8490, 1538, 1178, 870, 7378, 3203, 1169, 939, 1640,
                                        1758, 3658, 4634, 1438, 1500, 2915,
1078, 1357, 1314, 1179, 1189, 1262, 7247, 5238,
                                        1158, 4604, 1, 150, 425, 2734, 4930,
1245, 1247, 1266, 1300, 1367, 1670, 7957, 50, 6399,
                                         439, 493, 6726, 5312, 1392, 1539,
1547, 1683, 5813, 2362, 6030, 7362,
                                        7019, 917, 900, 1427, 1712, 6138,
1608, 1299, 1002, 393, 1136, 1373, 941, 17,
                                         851, 1418, 6329, 988, 1692, 4197,
5325, 3962, 3106, 1793, 1703, 924, 5177])
def GetRowCount(cur):
    cur.execute("select count(1) from reviews")
    return cur.fetchall()[0][0]

def CreateIndexs(cur):
    cur.execute("select id, name, reviews_num from restaurants where
reviews_num>30")
    nameToRestIndex={}
    restToNameIndex={}
    countIndex={}

    listOfNoiseRestaurants1=set(['SO','Street'])

    for row in cur.fetchall():

        if (row[0] not in listOfNoiseRestaurants and row[1] not in
listOfNoiseRestaurants1 and row[2]>30):
            nameToRestIndex[row[1]]=row[0]
            restToNameIndex[row[0]]=row[1]
```

```
            countIndex[row[0]]=0

    return (nameToRestIndex,restToNameIndex,countIndex)

def Create_Rest_Set(cur):
    cur.execute("select id, name from restaurants")
    Result=[]
    for row in cur.fetchall():
        if row[0] not in listOfNoiseRestaurants and is_ascii(row[1]):
            Pattern=""
            for word in nltk.word_tokenize(row[1]):
                Pattern+='\\b'+word+'\\b'
                Pattern+='+s?[, ]\s?'
            Pattern=Pattern[:-10]
            Result.append((row[0], row[1], Pattern))
    return Result

def GetRevies(index,cur):
    sql="select * from reviews limit 1000 offset "+str(index)
    cur.execute(sql)
    return cur.fetchall()

def is_ascii(s):
    return all(ord(c) < 128 for c in s)
```

7.6 Stop words list

```
Stop_Words = ['a', 'about', 'above', 'across', 'after', 'afterwards',
'again', 'against', 'all', 'almost', 'alone', 'along', 'already', 'also',
'although', 'always', 'am', 'among', 'amongst', 'amount', 'an', 'and',
'another', 'any', 'anyhow', 'anyone', 'anything', 'anyway', 'anywhere',
'are', 'around', 'as', 'at', 'back', 'be', 'became', 'because', 'become',
'becomes', 'becoming', 'been', 'before', 'beforehand', 'behind', 'being',
'below', 'beside', 'besides', 'between', 'beyond', 'bill', 'both', 'bottom',
'but', 'by', 'call', 'can', 'cannot', 'cant', 'co', 'computer', 'con',
'could', 'cry', 'de', 'describe', 'detail', 'do', 'done', 'down', 'due',
'during', 'each', 'eg', 'eight', 'either', 'eleven', 'else', 'elsewhere',
'empty', 'enough', 'etc', 'even', 'ever', 'every', 'everyone', 'everything',
'everywhere', 'except', 'few', 'fifteen', 'fill', 'find', 'fire', 'first',
```

```
'five', 'for', 'former', 'formerly', 'forty', 'found', 'four', 'from',
'front', 'full', 'further', 'get', 'give', 'go', 'had', 'has', 'hasnt',
'have', 'he', 'hence',  'her', 'here', 'hereafter', 'hereby', 'herein',
'hereupon', 'hers', 'him', 'his', 'how', 'however', 'hundred', 'i', 'ie',
'if', 'in', 'inc', 'indeed', 'interest', 'into', 'is', 'it', 'its' , 'keep',
'last', 'latter', 'latterly', 'least', 'less', 'ltd', 'made', 'many', 'may',
'me', 'meanwhile', 'might', 'mill', 'mine', 'more', 'moreover', 'most',
'mostly', 'move', 'much', 'must', 'my', 'name', 'namely', 'neither', 'never',
'nevertheless', 'next', 'nine', 'no', 'nobody', 'none', 'nor', 'not',
'nothing', 'now', 'nowhere', 'of', 'off', 'often', 'on', 'once', 'one',
'only', 'onto', 'or', 'other', 'others', 'otherwise', 'our', 'ours',
'ourselves', 'out', 'over', 'own', 'part', 'per', 'perhaps', 'please', 'put',
'rather', 're', 'same', 'see', 'seem', 'seemed', 'seeming', 'seems',
'serious', 'several', 'she', 'should', 'show', 'side', 'since', 'sincere',
'six', 'sixty', 'so', 'some', 'somehow', 'someone', 'something', 'sometime',
'sometimes', 'somewhere', 'still', 'such', 'system', 'take', 'ten', 'than',
'that', 'the', 'their', 'them', 'themselves', 'then', 'thence', 'there',
'thereafter', 'thereby', 'therefore', 'therein', 'thereupon', 'these',
'they', 'thick', 'thin', 'third', 'this', 'those', 'though', 'three',
'through', 'throughout', 'thru', 'thus', 'to', 'together', 'too', 'top',
'toward', 'towards',  'twelve', 'twenty', 'two', 'un', 'under', 'until',
'up', 'upon', 'us', 'very', 'via', 'was', 'we', 'well', 'were', 'what',
'whatever', 'when', 'whence', 'whenever', 'where', 'whereas', 'whereby',
'wherein', 'whereupon', 'wherever', 'whether', 'which', 'while', 'whither',
'who', 'whoever', 'whole', 'whom', 'whose', 'why', 'will', 'with', 'within',
'without', 'would', 'yet', 'you', 'your', 'yours', 'yourself', 'yourselves']
```

* This list of stop words was taken from the following wab page - http://xpo6.com/list-of-english-stop-words/