Recommendations for cocktail recipes

Sigurd Sippel
Hamburg University of Applied Sciences, Department of Computer Science,
Berliner Tor 7, 20099 Hamburg
sigurd.sippel@haw-hamburg.de

August 28, 2014

1 Introduction

When cooking, people tend to use a limited collection of recipes, which they try to remember. In a family with only a single cook, known as the nutritional gatekeeper [PGK11, p. 105], he is limited by this circumstances. A huge quantity of cooking recipes is available in the form of books or on the Internet and it is possible to select a new meal every day by using these sources [STIM09, p. 9]. Because of such a diverse, dizzying range of choices, a system of automatic recommendations can help deal with such diversity [XYL10, p. 254].

Recommender systems [RV97] make it possible for a user to identify a context and use specific recommendations. Cocktail recipes are simple cooking recipes, with a small list of ingredients and a single action — stir or shake. This paper considers the main aspects of recommender systems, including the algorithm of and modeling techniques for recipes. Since there are no papers on cocktails, theoretical approaches to cooking recipes will be applied to cocktail recipes.

The following application can be envisaged: In a bar, a guest seeks a recommendation for a drink. As a nutritional gatekeeper, the bartender can go beyond the limited choices at hand and can use a recommender system to make a less subjective and more convenient recommendation to the specific guest.

The focus is on learning things that are similar to another in a database that might be quite large. Learning is an aspect of artificial intelligence. Currently, when large amounts of data, called big data, are considered, it is common to call this as a black box. The next sections will consider big data as a white box.

What is learning? It is not enough to say that learning is a process in which one becomes better with the next attempt [WMJ03, p. 518]. A knife that has been ground will cut better, but has not learned anything new. It is necessary to know the goal to ground a system in the right way. It is simpler to define a learning task that works in a single area. If there is a specified input and an expected output, a system will successfully learn under the following condition: The input corresponds to particular specifications and the system is able to automatically produce the expected output. The learning task forms guidelines for the following sections.

Section 2 explain the idea behind a recommender system [RRSK10]. In the following section 3, the KDD process [FPSS96a] is described, which enable comparisons between data sets and detects correlations between them. It follows the example of a feature extraction with an ontology. Based on this, clustering algorithm and Euclidean distance measure [JMF99] are considered in accordance with the cocktail domain. Section 4 shows the modeling of a feature vector with combination and substitution of ingredients on the basis of a connected graph [TLA12]. An alternative shows section 5 with computed balance [KF10] to represent a recipe in an abstract form, which is independent of any special ingredient. In section 6 an analysis for obtaining user preferences with ratings follows [HLE12]. Based on this, a user profiling approach with Folksonomies [YGS08] is described in Section 7. It includes a nearest-neighbor classification [AJOP11]. An approach to cooking recipes uses Folksonomies [XYL10]. Finally, the conclusion and future works are in section 8.

2 Recommender systems

A recommender system — called RS — assists a user in finding something [RRSK10, p. 2]. For example, in a web shop, a user gets a recommendation of some products, which are similar to the last added item in his shopping basket. From the shop-owner's point of view, an RS increases the number of items sold. Another way of motivating a user is to give him the opportunity to select a item from a huge catalog, but it is necessary to know what the user likes.

An RS is strongly connected to a user [RRSK10, p. 6]. If a recommendation is accurate, it will satisfy him more. If a user gets what he wants, he will come back. The recommender system will have to learn how to improve itself with each user visit. The service owner can optimize his service (like stocking

management), if an RS tells him what users are interested in. Because of user satisfaction and loyalty, an RS is a part of human-computer interaction. Search engines could be considered as recommender systems and used in two ways: A user can find a interesting website with a search engine, and he can also check how important a website is, in accordance with his interest.

At the center of data are users and items [RRSK10, p. 10], the relation between them enables to predict how useful a item is for a user. With an estimation $R(user, item) \rightarrow Real$, a limited set of items with the best estimation can be assigned as a prediction to a user. In order to predict an item for a user, the item has to be comparable to other items. The concept is as follows: If a user likes item x, he will probably like an item y near to x. Because the recommendations contains only user-related items, recommender systems are also called collaborative filtering [RV97, p. 56]. Comparison depends on the quality of knowledge. Knowledge is extracted out of simple text or heavily structured relational databases.

3 Knowledge discovery in database and data mining

Manual analysis of huge volumes of data is a slow, expensive and also subjective process [FPSS96a, p. 28]. Knowledge discovery in database and data mining process — called KDD process — aims at automatically extracting useful knowledge from huge volumes of data [FPSS96a, p. 27]. A company's periodic report, based on facts in relational databases, is an example of an application of KDD. The computed knowledge, in this case, is: Sales of a product in a quarter are lower than the last five-year period. Raw data in a database contains much information [MHC06, p. 3]; the end product of the process of information discovery is the knowledge extracted [FPSS96b, p. 39].

Preconditions for the KDD process (Figure 1) are: Understanding a domain, locating available background knowledge, and identifying a goal. In the domain of cocktails, the background knowledge could be an ontology of ingredients. For example, a goal is to find duplicate recipes from different sources. The KDD process is divided into five steps [FPSS96a, p. 30]: The first step is selecting a target data set, which will be used to extract knowledge. The set of facts in the database comprises the data [FPSS96b, p. 41]. In a recipe database, these could be titles, authors' names and ingredients of recipes. For identifying similar ingredients, information such as titles and authors' names are not relevant. Only relevant information should be in the target data set. The next step is the preprocessing for reducing noise and outliers. Removing a set of stop words, which is not relevant, is a good method of reducing noise. If a plausibility check of the remaining data is undertaken, then useless data can be weeded out. A recipe without any ingredient is useless. In the third step, data is transformed into patterns. For example, an ingredient, as a raw string, can be transformed into an ingredient by finding the string in an ingredient database — the background knowledge. A pattern is a feature vector; each component of a pattern is a feature [JMF99, p. 269f]. With a distance function like an Euclid (subsection 3.2), the distance between two feature vectors are computable. A pattern should be useful and as simple and understandable as possible, since otherwise the distance would be arbitrary.

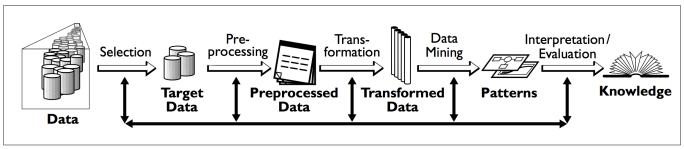

Figure 1: KDD process [FPSS96a, p. 29 Figure 1]

The next step is data mining. In accordance with the model, the composition of the target data, and the goal, there are four possible model functions: Regression, clustering, classification and summarization.

A regression analysis is used to find a function to describe the feature vectors. For example, in a time sequence, the regressions are able to predict the future trends because of the known feature vectors [FPSS96b, p. 44]. Clustering computes distance with each feature vector combination and creates a group of feature vectors — a cluster — with a smaller distance than a specific threshold [JMF99, p. 274]. Classification maps feature vectors to predefined classes [FPSS96b, p. 44]. A summarization process detects the most important parts of a document and computes the correlations to other documents [CWML13, p. 527]. For example, it reduces a document to only the words most frequently used. After a function is chosen, a concrete algorithm has to be selected and run for data mining. The result is the correlation of the input feature vectors.

The last step is the evaluation and interpretation of the mining results. Feature vectors which do not have any correlation or domain-specific semantic sense have to be removed. The technical feature vector cannot be easily read by a user. A visualization, as in a connected graph, makes it readable. If the accuracy of the result is good enough, potentially

useless features can be removed to optimize the performance. A feature is useless, if accuracy without this feature is at least as good as with the feature.

KDD as a development process is not a downfall model. The steps can be repeated at every moment, if it is necessary to make the knowledge extraction process more accurate.

Data mining is a statistical method of analyzing data. In statistics, a random sample is considered significant if it is collectively valid [FKPT07, p. 33]. A problem is that if someone searches long enough in a set of statistical data, he might find a significant pattern, which may not have any link with reality [FPSS96b, p. 40].

KDD is a timeless and adaptive approach for extracting knowledge out of data because no algorithm is predefined and it is domain independent. It is a process with loosely coupled steps. Not every step is sophisticated, but these steps are necessary. The disadvantage is that every developer, who wishes to use KDD, will have to find his own focus, like distance measurement or data visualization. Every step is a potential money sink.

3.1 Feature extraction of cocktail recipes

Two recipes, r_1 and r_2, are at a distance between each other. It is necessary to convert these recipes, which exist as raw strings, into a comparable form. These comparable forms have to be as precise as possible to get a usable distance [JMF99, p. 271]. In the next example, only the quantities of ingredients are considered. These two Manhattan recipes are out of historic cocktail books and have been a bit simplified. The texts with instructions for making the preparations have been removed. In the second recipe, concrete measurement units have been added because then it would be possible to say something about the volume.

Manhattan Cocktail

(1882 Harry Johnson, Bartenders Manual p. 182)
1 dash of gum syrup, very carefully;
1 dash of bitters (orange bitters);
1 dash of curacao, if required;
1/2 wine glass of whiskey
1/2 wine glass of sweet vermouth

MANHATTAN COCKTAIL, SWEET

(1937 W. J. TARLING, Cafe Royal p. 127)
1/2 oz Martini Sweet Vermouth.
1/2 oz Rye.
Serve Maraschino cherry.
A dash of Angostura may be added, if desired.

A Levenshtein distance [PROA12, p. 706] gives the number of characters that have to be changed to transform a string into another one. The transformation entails addition, removal and switching of characters. The distance between two strings *Rye* and $Whiskey$ is higher than between *Rye* and *Gin* (Equation 1), though in a semantic way, a rye is a special kind of whiskey and rye and gin are absolutely different. So, when characters are considered, no semantic comparability is possible.

$$levenshtein(rye, whiskey) = 8 \quad (1)$$
$$levenshtein(rye, gin) = 3$$

For preprocessing, the stop words are removed to reduce noise (Equation 2).

$$stopwords = \{of, very, carefully, (,), ;, ., \quad (2)$$
$$may, be, added, if, desired, A, Serve\}$$

The result is:

Manhattan Cocktail
1 dash gum syrup
1 dash bitters orange bitters
1 dash curacao
1/2 wine glass whiskey
1/2 wine glass sweet vermouth

MANHATTAN COCKTAIL, SWEET
1/2 oz Martini Sweet Vermouth
1/2 oz Rye
Maraschino cherry
dash Angostura

In transformation steps, the background knowledge is used. Background knowledge contains synonyms such as *gum* is *sugar* [TLA12, p. 302]. The other part of the knowledge background is an ontology [PGK11, p. 108] for recognizing mappings like $bitters \rightarrow orange\ bitters$, $bitters \rightarrow Angstura$ or $Whiskey \rightarrow Rye$.

Manhattan Cocktail
1 dash [sugar]
1 dash [bitters, orange bitters]
1 dash [curacao]
1/2 wine glass [whiskey]
1/2 wine glass [sweet vermouth]]

MANHATTAN COCKTAIL, SWEET
1/2 oz [sweet vermouth]
1/2 oz [rye]
[maraschino cherry]
dash [bitters, angostura]

The ingredients have been transformed, but the units of measurement and the quantities are missing. The measurement units have to be recognized to transform it into a single unit: $1\ wine\ glass = 3\ cl = 1\ oz$. The ontology is also used to decide whether a unit is scalable or not. Units that are not scalable remain unaffected. Quantities that are not explicit are added as 1 to identify *dash* as $1\,dash$ and *maraschino cherry* as $1\ maraschino\ cherry$.

Manhattan Cocktail
1 dash [sugar]
1 dash [bitters, orange bitters]
1 dash [curacao]
1/2 cl [whiskey]
1/2 cl [sweet vermouth]

MANHATTAN COCKTAIL, SWEET
1/2 cl [sweet vermouth]
1/2 cl [rye]
1 [maraschino cherry]
1 dash [bitters, angostura]

3.2 Clustering with Euclidean distance measure

A process of clustering detects groups — called clusters — in a set of feature vectors [JMF99, p. 265]. Clustering is unsupervised learning, since it is not necessary to have a specific learning set. Depending on the quality of feature extraction, groups contain feature vectors that are more similar to each other than to outside feature vectors. Similarity is defined as a distance function such as a Euclidean distance (Figure 2). The extracted features are components of the feature vector. A Euclidean distance computes the difference between each component, squares it and takes the 2-nd root of the sum.

$$d_2(\mathbf{x}_i, \mathbf{x}_j) = (\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2)^{1/2}$$

Figure 2: Euclidean distance [JMF99, p. 271]

The validity of the result — the located clusters — have to be meaningful and comprehensible, but this is subjective [JMF99, p. 268]. An indication of this is that clusters might be too big or too small, which are noticed as an annoying artifact. Another way of understanding this is through data abstraction. A cluster, as a set of feature vectors, has a centroid. For example, a centroid is a feature vector closest to the center of a triangle, which describes the center (Figure 3). A centroid is also a compact representation of a cluster.

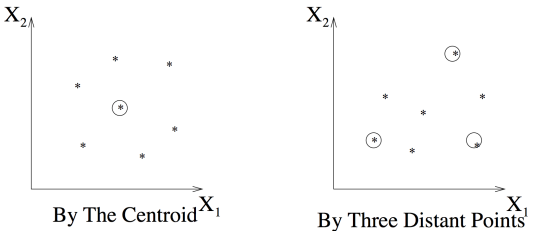

Figure 3: Centroid of a cluster [JMF99, p. 282]

The initial state of a clustering algorithm is either agglomerative or divisive [JMF99, p. 274]. An agglomerative algorithm creates for each feature vector a new cluster. The clusters will merged together. A divisive algorithm creates one cluster for all feature vectors, which will be split. Both these methods need a stopping criterion, a threshold, to decide whether to merge or split.

For computing clusters, there are hierarchical and partitional algorithms. The hierarchical and agglomerative approach seeks the nearest pairs and uses these pairs to find the nearest pairs of pairs. The result is a nested cluster — a tree (Figure 4). One of the clusters is useless, but this cluster can be cut at every depth to get the end result. It has a high complexity in time and space [JMF99, p. 277].

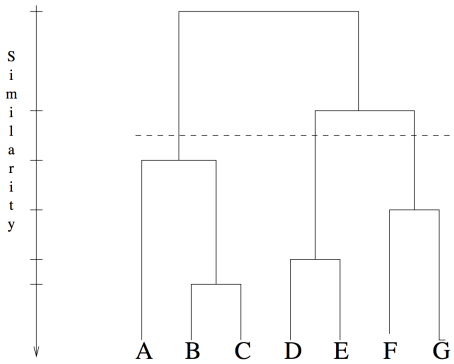

Figure 4: Hierarchical clustering [JMF99, p. 276]

The partitional approach considers the feature vectors as one partition [JMF99, p. 278]. The k-means algorithms is an example. For initialization purposes, it chooses a feature vector randomly for k clusters and considers the feature vector as the centroid, because it is the only one. Then a loop starts: The clusters are split or merged. The centroid of each cluster is recomputed. The loop will stop if the clusters are not, or only minimally, changed. The k-means algorithm has a low complexity of $\mathbf{O}(n)$, but it needs isotropic features for delivering a good result.

Up to this point, a feature vector is in only one cluster. The clusters are disjunct sets. This is called hard clustering; the alternative is fuzzy clustering [JMF99, p. 281]. The main difference is that the assignment

of a feature vector to a cluster is not finished if the closest cluster is found. It needs a set of clusters that are close enough (Figure 5).

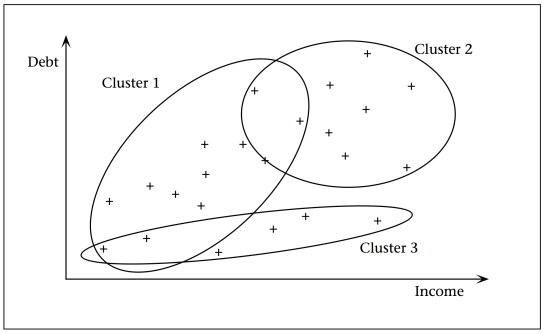

Figure 5: Clustering [FPSS96b, p. 45 Figure 5]

In order to use a clustering algorithm, the feature vectors have to be constructed. For computing the distance between two feature vectors, domain dependent knowledge is needed for selecting the features and distance function [JMF99, p. 289].

The preprocessed and transformed (subsection 3.1) recipes are converted to feature vectors $r1$ and $r2$. The ingredients are the components. For measuring the Euclidean distance, the dimensions of both have to be equal [PGK11, p. 108]. If a component does not have a counterpart in the other recipes, an empty ingredient [_] will be added on the side. Equally, the acidity is normalized: $[whiskey, rye]$ has an counterpart such as $[whiskey, \_]$.

$$\begin{pmatrix} [whiskey, \_] \\ [sweet\ vermouth] \\ [bitters, orange\ bitters] \\ [\_] \\ [sugar] \\ [curacao] \\ [lemon\ peel] \end{pmatrix} \begin{pmatrix} [whiskey, rye] \\ [sweet\ vermouth] \\ [bitters, angostura] \\ [maraschino\ cherry] \\ [\_] \\ [\_] \\ [\_] \end{pmatrix}$$

For computing a Euclidean distance, each component of $r1$ is subtracted by the same component of $r2$. An ingredient is not a number, so the subtraction has to be defined. This is first done for one element (Equation 3).

$$sub(x, y) := if(x == y) : 0\ else\ 1 \qquad (3)$$
$$sub(bitters, bitters) = 0$$

For a component, the average of all elements is computed (Equation 4).

$$sub([bitters, angustura], [bitters, \_]) = (0 + 1)/2 = 0.5 \qquad (4)$$

The Euclidean distance between $r1$ and $r2$ is computable with a defined subtraction (Equation 5).

$$d(r1, r2) = (0.5^2 + 0 + 0.5^2 + 1 + 1 + 1 + 1)^{1/2} = 2.25 \qquad (5)$$

Two recipes that are equal have a distance of $d = 0$. The recipes $r1$ and $r2$ are not equal, but very similar. For this reason, a low distance is expected. All features have the same weight. A specific weight for every feature, which indicates how important it is, makes the distance more precise.

There are three types of features [JMF99, p. 270]. Ordinals, such as military rank or sound intensity (loud, quiet), can be represented by a defined ordered list. Qualitative features like colors or ingredients have arbitrary, probably unordered, values, and quantitative features like discrete values, weights or volumes are scalable.

Ingredients with quantities and units are usually scalable. Some units are unscalable, such as dashes. With quantities as feature weight and a standard weight of 0.1 for unscalable units, it is possible to calculate a more precise distance (Equation 6).

$$d(r1, r2) = (0.5 * 0.5^2 + 0.5 * 0 + 0.1 * 0.5^2 \qquad (6)$$
$$+0.1 * 1 + 0.1 * 1 + 0.1 * 1 + 0.1 * 1)^{1/2} = 0.75$$

Of course, quantity is not equal to intensity. If the aim is to recommend an appropriate flavor, an intensity is needed. It is difficult to consider each ingredient as a new feature, because the correct order is needed. It is an intuitive, but not a very efficient way, of comparing two recipes.

4 Recipe modeling with combination and substitution

A recipe modeling approach [TLA12, p. 298] uses ingredient networks for obtaining a more efficient recommendation. A database of recipes, available on the Internet — these are posted by a big social community — are considered to be sources that contain what people like. They collect recipes from their families and put these into the database, where they are discussed by others.

The database contains recipes in the form of unstructured text. The recipe contains ingredients, steps for preparing them, quantities, temperatures and cooking times. Additional information includes user-based reviews and ratings. There is also a list of users' favorites and a classification of the recipes, which lists the meal times that are suitable and the regions from which they come.

The goal is to find the ingredients and remove all the additional information [TLA12, p. 299]. For identifying the ingredients, they removed the stop words, and obtained background knowledge to recognize synonyms such as *Bush's Original* = *baked beans*. The background knowledge does not

contain a list of ingredients, since an attempt failed as too less ingredients were identified. The list was always too small. Additional information, such as quantities and temperature, were also removed for merging similar ingredients. The following are equal: $10g\ cold\ butter = 100g\ smooth\ butter$.

As a consequence, there are many ingredients that are very similar, such as *cheddar cheese* and *shared cheddar cheese*. Such ingredients are merged in to a single ingredient based on string similarity. As the next step, the ingredient list is sorted by frequency, because the top 1,000 ingredients are being sought. The ingredients that are removed are the special brand names, misspelled items or very special items (like yolk-free egg noodles).

$$\mathrm{PMI}(a, b) = log \frac{p(a, b)}{p(a)p(b)}$$

Figure 6: Point wise mutual information [TLA12, p. 300]

Based on the frequency of the ingredients, statistical mutual information is computed for every ingredient (Figure 6): $P(a)$ is the probability of how often an ingredient a appears in the recipe database. The frequency of the ingredients is divided by the recipe count. $P(a, b)$ is the probability of the combination (a, b). The probability of the combination, divided by the multiplied single ingredients, is the point wise mutual information, called PMI (Figure 6).

The result of an PMI for every ingredient combination is a connected graph (Figure 7). An edge represents a combination where the PMI has to exceed a threshold. The font size of a node represents the size of the PMI. On the left side, the graph contains more sweet ingredients, and on the right side, there are more herbal ingredients. In the middle, there are ingredients such as water, oil, salt and lemon juice, which link both sides. A clustering analysis detects a big cluster on the sweet side, which contains only the ingredients for mixed drinks, such as lime juice or spirits [TLA12, p. 300].

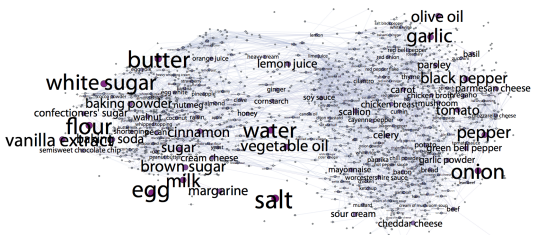

Figure 7: Connected graph for combination [TLA12, p. 301 Figure 2]

On the basis of user reviews or comments, which contain modifications, it is possible to find out how flexible a recipe is. There are signaling words such as *add*, *extra* or *instead*, which make it easy to recognize a modification. A $PMI(a \rightarrow b)$ computes the probability of substituting an ingredient a with b (Figure 8).

$$\mathrm{PMI}(a \rightarrow b) = \log \frac{p(a \rightarrow b)}{p(a)p(b)}$$

Figure 8: PMI for substitution [TLA12, p. 303]

A substitution graph (Figure 9) represents substitutions as edges $a \rightarrow b$, which connected the nodes. The nodes are the ingredients.

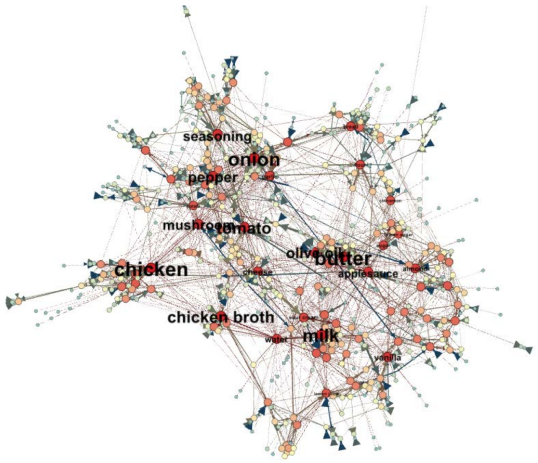

Figure 9: Connected graph for substitution [TLA12, p. 302 Figure 5]

Combination and substitution are two kinds of summarization methods, which can be used in KDD. PMI is a measure. In a database with huge volumes of data, a PMI of the combinations indicates how common it is. It is an indicator of how well they go together. A PMI can be used for a feature vector for characterizing a recipe. This is particularly simple for the purposes of transfer to cocktail recipes, because they have a smaller list of ingredients. When transferred to a graph, the number of combinations is equal to the number of edges. It is a complete graph, because all nodes are connected to all other nodes. If there are n nodes, there are $K_n = \frac{n(n-1)}{2}$ edges. The recipe $r2$ contains five ingredients and because of that a manageable number of 10 ingredient combination. An occurrence of *rye* in combination with *sweet vermouth* is then a ratable fact. A threshold can be defined to prevent the number of occurrences from becoming too few (Equation 7).

$$(rye, sweet\ vermouth) \rightarrow PMI(rye, sweet\ vermouth) \tag{7}$$

5 Balance with a nutritional pyramid

Another modeling approach is on the basis of nutritional balance [KF10, p. 56:1]. The goal is to generate healthy meal plans. The user can get a completely auto-generated meal plan and can choose favorites, including self-monitoring of balance changes. It is based on the Japanese nutritional pyramid (Figure 10). The pyramid is divided into six food groups: Water, grains, vegetables, fish/meat, milk and fruits.

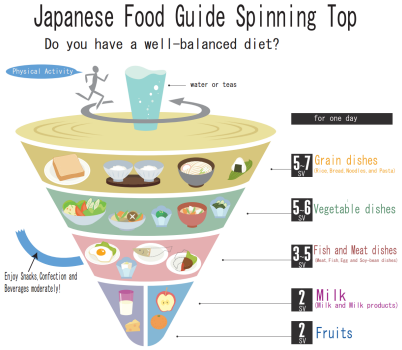

Figure 10: Japanese nutritional pyramid [Jap14]

With the help of a domestic science handbook, a dictionary is created, which contains foods classified into food groups such as *meat* → *pork* [KF10, p. 56:4]. Because all the ingredients of a cooking recipe are classified and the quantities are available, it is possible to compute the ratio of every food group referred to in a recipe. All the ratios of ingredients of a particular food group are computed and summarized. All six food group ratios are together a recipe balance, which is visualized as a red rhombus (Figure 11).

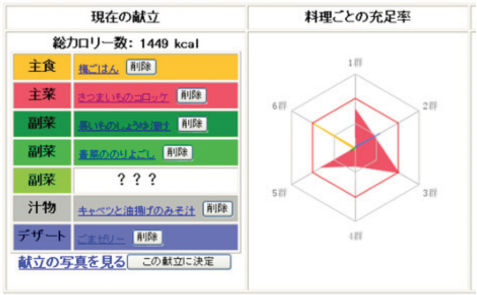

Figure 11: Balance [KF10, p. 56:4 Figure 4]

The meal planning uses the balance to find meals, which together represent an optimal intake of foods per day. The intake per day is specific to age, gender and food group. It is a part of the Japanese food standard.

This is an approach toward optimizing diets from the point of view of healthfulness. The question that arises is this: How can the balance of such a recipe be characterized to make it comparable to another one? It is useless to know that an ingredient of a huge database does not exist in a specific recipe. But if the ingredients are classified into a small number of groups and only the groups are considered. The knowledge is important that one group does not exist in a recipe. This can be computed, because all ratios are available, and the sum of all ratios is 1. It is important that this group covers as many areas as possible. It is an indicator in the Figure 7 that the biggest ingredient nodes, such as water or milk, represent the food groups. Meat and fish are exceptions. These nodes are smaller, because there are more nodes, each of which represents a specific kind of meat or fish. User don't distinguish between several kinds of milk.

A cocktail recipe is more restricted, since there are no ingredients like fish and meat. Milk products are rare in classical cocktail recipes. Acid from fruits, water (like soda or melting ice), sweets (like liqueur, syrup) and alcohol (like spirits) are groups of ingredients that are frequently used. For a cocktail recipe, it is possible to compute the ratios in an ingredient group. All the ratios together form the balance. For example, in a Collins recipe (Equation 8), there are ingredients with quantities and ratios of groups.

$$\begin{aligned} Collins = \{&5\,cl\,Gin\,(47\,\%\,alcohol, 53\,\%\,water), \\ &3\,cl\,lemon\,juice\,(5\,\%\,acid, 95\,\%\,water), \\ &2\,cl\,sugar\,syrup\,(2/3\,sugar, 1/3\,water), \\ &20\,cl\,soda\,(100\,\%\,water)\} \end{aligned} \tag{8}$$

The total volume is $30\,cl$; hence the group ratios can be computed in a simple way (Equation 9). The ratio is not an intensity; it has to be normalized for attaining greater precision. The feature vector $balance^{-1}$ contains only numerical values, as the subtraction has already been defined. A balance is available for every recipe and a similarity measure with a balance does not depend on special ingredients like *rye* or *sweet vermouth*.

$$\begin{aligned} c(alcohol) &= (5 * 0,47)/30 = 0,078 \\ c(sugar) &= (2 * 2/3)/30 = 0,044 \\ c(acid) &= (3 * 0,05)/30 = 0,005 \\ c(water) &= (5 * 0,53 + 2 * 0,95 + \\ &\quad 2 * 1/3 + 20)/30 = 0,841 \\ balance &= (c(alcohol), \\ &\quad c(sugar), c(acid), c(water))) \end{aligned} \tag{9}$$

6 User preferences for recommendation

It is necessary to know the user's preferences for making an appropriate recommendation [HLE12, p. 18]. One way of getting to know what people like is to ask them: Users rated a randomly chosen recipe between one and five stars. A rating shows what users like. Apart from the recipe rating, user modification options in the comments show that the user has identified and worked with a recipe [TLA12, p. 300]. User activities on a website — such as page visits and favorite recipes — are also a part of a fingerprint, which can be used to know what users like [WGH11, p. 50].

Following this, the users should explain their ratings in three categories. The categories are: Health, preparation and individual preferences, which have the reasons in the form of a check box like *too many ingredients* or *my favorites ingredients*. The reasons are divided into positives and negatives. The recipes are not randomly chosen — basically meta data is available for filtering recipes for vegetarian or lactose-intolerant users. It is assumed that the user consumes the food immediately; hence, there is a time-dependent filter. Only those dishes are chosen that are appropriate for the actual time of the day, such as breakfast in the mornings. The learning data comprises the results, the ratings and the reasons.

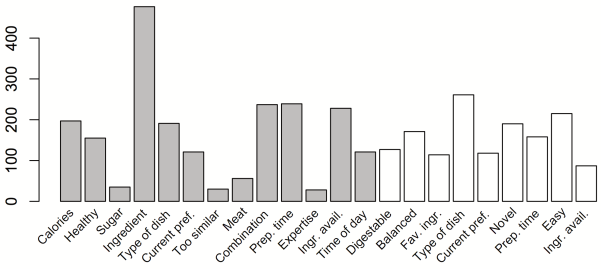

Figure 12: Frequencies of reasons [HLE12, p. 21 Figure 2]

The diagram shows the frequencies of the reasons (Figure 12). The gray bars on the left side are the negative reasons, the white bars on the right side are the positive reasons. The *ingredients* that are disliked catch the eye in the diagram, while wrong *combinations* and *preparation time* that are too long are also often selected by the users. One the positive side the *kind of dishes*, *preparation time* and *easy preparation* are used frequently.

The problem with this result is that the dependencies of the data are not clear. *preparation time* and *easy preparation* are dependent on the actual context; the reasons are not always valid. In a linear model analysis, *ingredients* and *combination* are the most significant factors [HLE12, p. 20].

Another example of data dependencies are the ratings of correlated calories (Figure 13). Users who select health reasons are often classified as being part of the health-conscious user group, while the rest comprise the unhealthy group. As a result, unhealthy users give high ratings to recipes with more calories, while the healthy users give low ratings to such recipes.

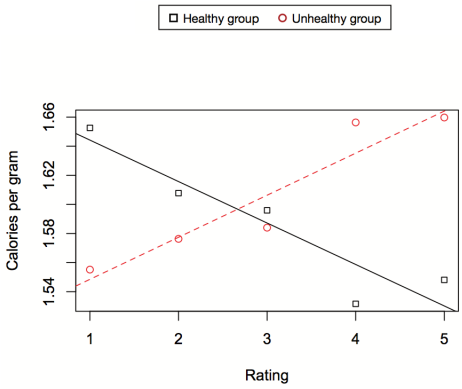

Figure 13: Correlation between calories and ratings [HLE12, p. 21 Figure 3]

Because the recommendations based on the most important ingredients and ingredient combinations are more precise, they are independent of the actual context [HLE12, p. 21].

Let us consider a cooking recipe with a preparation time of 0.5h. A user b might say that the preparation time is too long. Then a rule can be created (Equation 10). The rule represents a user profile. If a user matches the user profile $fastfood$, then the user is classified under $fastfood$. All recipes with small preparation times match users who are classified under this rule.

$$user(b, preparation(< 0.5h)) \Rightarrow like(b, fastfood) \tag{10}$$

When the connection between a user and his/her favorites is known, a recommender system can recommend recipes that are similar to the user's favorites. Instead of using just a single user, a user profile can be generated to obtain all the favorites of the classified user. The recommendation requires a minimum distance, but the distance has to be as small as possible. It is an appropriate learning task to identify clusters, since it is then possible to recommend a neighboring cluster.

But there is no further information about the users, as this is only a collection of users. It does not contain personalized information, such as demographic information (age or gender) or culture or origin [PGK11,

p. 110]. An extension to this model includes the contextual information, such as the weather or the season, which are used in recipes [PGK11, p. 113]. With supporting services, such as a calendar or a weather service, such contextual information can be added to the fingerprint of a user. A recommendation in a bar works in a similar way: A guest (such as a business man) in an actual context (summer, $30°C$) seeks a recommendation. If the RS knows a recipe with such a personalization and context, it has the ability to recommend a cocktail.

7 Folksonomies for user profiling

Folksonomies is the approach of a personalized search engine to documents [YGS08, p. 70]. A user can choose keywords to search documents. A keyword can have more than one meaning. A bank could be a financial institution or a seat. The meaning depends on the context. A consequence is that the users have to ferret out the relevant documents from a long list of results. The position on a results page is wasted because of irrelevant documents. If a user context is provided, the search engine can filter the results page. The idea is this: Users with the same context use words with the same meanings. Keywords are considered tags, which describe documents and are used by known users. Because of this, Folksonomies is also called collaborative tagging. Folksonomies F are a tuple and contain users U, tags T, documents D and a relation A called annotation (Equation 12).

$$F = (U, T, D, A) \tag{11}$$
$$A \subseteq U \times T \times D$$

Documents that are connected to the same tag t are in a set D_t. The document sets are grouped by the users who used the tag. A cluster $X_{t,i}$ represents documents with one tag t and one user i. For every $X_{t,i}$, there is a set $T_{t,i}$ of tags, which are used for a document. The clusters X are very small and there are clusters that contain tags with the same meaning.

Because of this, the most frequent tags are searched and the referenced clusters are merged, if they overlap above a specific threshold α. The overlap is defined as the count of the intersection of tags used by the users i and j and divided by the count of both tag sets. A high overlap means that two users mostly used the same tags for the same documents.

$$overlap(T_{t,i}, T_{t,j}) = \frac{|T_{t,i} \cap T_{t,j}|}{|T_{t,i} \cup T_{t,j}|}$$

Figure 14: Overlap [YGS08, p. 72]

If $overlap(T_{t,i}, T_{t,j}) > \alpha$, the document clusters $(X_{t,i}, X_{t,j})$ are merged together. These clusters are used as classes for classifying the nearest neighbor [YGS08, p. 72]. The idea is as follows: Documents that are indexed by a search engine are represented by tags. If a user searches with a tag, there is a result set S_t that contains documents, which contains this tag. Based on the document tags, a distance to the document classes is computable. The search result can be classified in accordance with the different meanings. This classification works only if the context is contained in the learning data.

7.1 Nearest neighbor classification

Classification is different from clustering. It requires learning data, which map feature vectors to labels. Because it uses learning data, classification is a kind of supervised learning [AJOP11, p. 48]. Learning describes a function that determines whether or not a randomly chosen feature vector is in a class [WMJ03, p. 523].

$$trainingset = \{(x_1, l_1)...(x_n, l_n)\} \tag{12}$$
$$neighbours \subseteq trainingset \tag{13}$$
$$neighbours = \{(y_1, l_1)...(y_k, l_k)| \tag{14}$$
$$min(\sum_{1}^{k} d(q, y_k), trainingset)\} \tag{15}$$

The k-nearest neighbor classifier — called k-NN — is an instance-based algorithm, which uses the training set only for classification [WMJ03, p. 534]. An unclassified feature vector q is classified with its nearest neighbors [AJOP11, p. 48]. The k defines how many neighbors are being considered. Assuming that the training set has a size n and contains tuples of feature vectors x and mapped labels l, the k neighbors are a subset of *trainingset* with the following condition: The sum of all distances d between q and the *neighbors* is minimal.

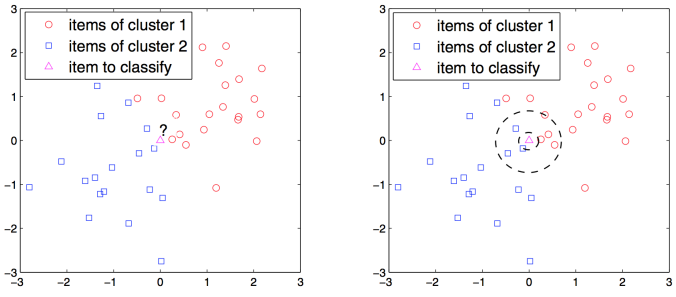

Figure 15: Example of an query on K-nearest neighbor classifier [AJOP11, p. 49]

If k is too small, the classifier may yield a bad result because there might be too much noisy data in

the neighborhood. If k is too high, too many different neighbors come into the neighborhood. With $k = 1$ in the example (Figure 15), the feature vector is classified as a square label; with $k = 5$ the result is a circle label. This algorithm is a lazy learner because it does not run a training phase before a random feature vector is classified, it just uses the neighborhood of an query q.

7.2 Hybrid semantic item model

The hybrid semantic item model combines four modellings to recommend cooking recipes, including Folksonomies [XYL10, p. 257]. The first model, called cooking flow graph, is represented by a directed graph. Its ingredients and actions are nodes and cooking actions, and ingredient flows are different kind of edges. Additionally, there are some constraints, like cooking temperature or cooking time.

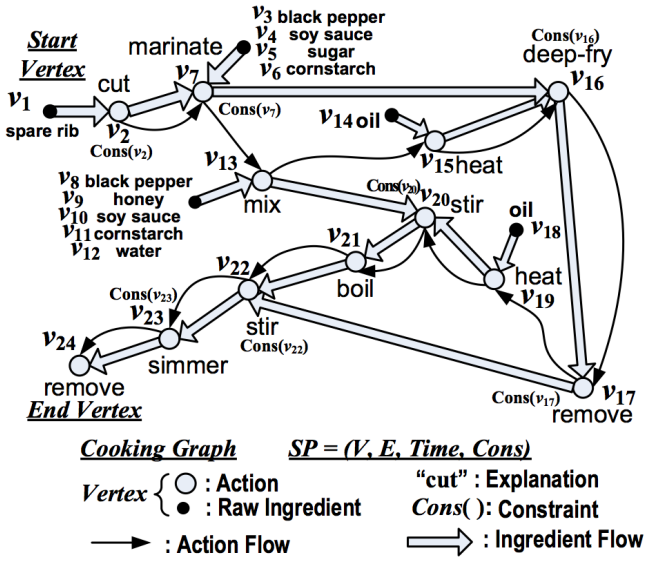

Figure 16: Cooking flow graph [XYL10, p. 256 Figure 2]

The next model is on eating, and represents taste, textures and temperatures. But that depends on a subjective point of view, and is very difficult to verify. Because of this difficulty, eating is not explicitly represented, it is represented through user fingerprints, such as ratings and comments on recipes. The last models have nutritional information and media, such as images. For every model, a distance function exists, which has been summed to a single distance.

The eating model is an application of Folksonomies. The documents are recipes and the tags are represented by a user's fingerprint [XYL10, p. 256]. The fingerprint connects the user with the recipes.

These models cover a broad semantic space; they are not just ingredient models or only user models. That is why it is called a hybrid semantic model. In particular, the use of a personalization, which does not model a user explicitly and is not dependent on a domain, makes this approach adaptable to a cocktail recommender system. The cooking flow graph is too sophisticated for a cocktail model, because a cocktail recipe is smaller, and has only two actions: Shake and stir.

8 Conclusions and future work

The term recommender systems is a good guide for defining an area where one can work. It shows that a recommendation depends on users and needs comparable items. The KDD process contains the development steps required to obtain the knowledge about the items by making items comparable and finding correlations. Future works will focus on feature extraction. Data visualization is not in focus. The model functions like clustering and classification are the central part for recommendation because it has the ability to detect similar items.

Modeling, such as combination, substitution and balancing, are important to make the model function in a more precise manner. The key is to find a model that can characterize a recipe with as little information as possible. A big feature vector does not correspond to a precise result, because the noise is bigger and the danger of redundant information is higher. Of course, substitution requires many user comments, which contain suggestions about the substitution of ingredients. Experiments with modeling, different data sets such as recipe books or social media content, and algorithms for model functions will show how much precise it can be.

Explicit user modeling is very disillusioning because, as the study shows, just ingredients are significant in getting to know what people like or dislike. Implicit user modeling with a user fingerprint is more attractive, but also needs a great deal of user data for good results. Folksonomies are an interesting approach for obtaining user profiles for every domain. With user profiles, a recommender system would be able to easily classify a recipe in accordance with a user class. But a recommender system, which only gets a single recipe to find similar ones, is an easier approach, and does not depend on users or user profiles. Because of this, future work will concentrate on recipes and not on users.

References

[AJOP11] Amatriain, Xavier ; Jaimes, Alejandro ; Oliver, Nuria ; Pujol, Josep M.: Data mining methods for recommender systems. In: *Recommender Systems Handbook*. Springer, 2011, S. 39–71

[CWML13] Chang, Yi ; Wang, Xuanhui ; Mei, Qiaozhu ; Liu, Yan: Towards Twitter Context Summarization with User Influence Models. In: *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*. New York, NY, USA : ACM, 2013 (WSDM '13). – ISBN 978–1–4503–1869–3, 527–536

[FKPT07] Fahrmeir, Ludwig ; Künstler, Rita ; Pigeot, Iris ; Tutz, Gerhard: *Statistik*. Springer-Verlag Berlin Heidelberg, 2007 (Springer-Lehrbuch). `http://books.google.de/books?id=ZinjP103iRcC`. – ISBN 9783540697398

[FPSS96a] Fayyad, Usama ; Piatetsky-Shapiro, Gregory ; Smyth, Padhraic: The KDD Process for Extracting Useful Knowledge from Volumes of Data. In: *Commun. ACM* 39 (1996), November, Nr. 11, 27–34. `http://dx.doi.org/10.1145/240455.240464`. – DOI 10.1145/240455.240464. – ISSN 0001–0782

[FPSS96b] Fayyad, Usama M. ; Piatetsky-Shapiro, Gregory ; Smyth, Padhraic: Advances in Knowledge Discovery and Data Mining. Version: 1996. `http://dl.acm.org/citation.cfm?id=257938.257942`. Menlo Park, CA, USA : American Association for Artificial Intelligence, 1996. – ISBN 0–262–56097–6, Kapitel From Data Mining to Knowledge Discovery: An Overview, 1–34

[HLE12] Harvey, Morgan ; Ludwig, Bernd ; Elsweiler, David: Learning user tastes: a first step to generating healthy meal plans? In: *First International Workshop on Recommendation Technologies for Lifestyle Change (LIFESTYLE 2012)* (2012), S. 18

[Jap14] *Japanese Food Guide Spinning Top*. Website, 2014. – Online Abruf (06.06.2014) www.mhlw.go.jp/bunya/kenkou/pdf/eiyou-syokuji5.pdf

[JMF99] Jain, A. K. ; Murty, M. N. ; Flynn, P. J.: Data Clustering: A Review. In: *ACM Comput. Surv.* 31 (1999), September, Nr. 3, 264–323. `http://dx.doi.org/10.1145/331499.331504`. – DOI 10.1145/331499.331504. – ISSN 0360–0300

[KF10] Karikome, Shihono ; Fujii, Atsushi: A System for Supporting Dietary Habits: Planning Menus and Visualizing Nutritional Intake Balance. In: *Proceedings of the 4th International Conference on Uniquitous Information Management and Communication*. New York, NY, USA : ACM, 2010 (ICUIMC '10). – ISBN 978–1–60558–893–3, 56:1–56:6

[MHC06] Maulik, U. ; Holder, L.B. ; Cook, D.J.: *Advanced Methods for Knowledge Discovery from Complex Data*. Springer, 2006 (Advanced Information and Knowledge Processing). `http://books.google.de/books?id=OOOSx1X2-_sC`. – ISBN 9781846282843

[PGK11] Pinxteren, Youri van ; Geleijnse, Gijs ; Kamsteeg, Paul: Deriving a Recipe Similarity Measure for Recommending Healthful Meals. In: *Proceedings of the 16th International Conference on Intelligent User Interfaces*. New York, NY, USA : ACM, 2011 (IUI '11). – ISBN 978–1–4503–0419–1, 105–114

[PROA12] Papaioannou, Thanasis G. ; Ranvier, Jean-Eudes ; Olteanu, Alexandra ; Aberer, Karl: A Decentralized Recommender System for Effective Web Credibility Assessment. In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. New York, NY, USA : ACM, 2012 (CIKM '12). – ISBN 978–1–4503–1156–4, 704–713

[RRSK10] Ricci, Francesco ; Rokach, Lior ; Shapira, Bracha ; Kantor, Paul B.: *Recommender Systems Handbook*. 1st. New York, NY, USA : Springer-Verlag New York, Inc., 2010. – ISBN 0387858199, 9780387858197

[RV97] Resnick, Paul ; Varian, Hal R.: Recommender Systems. In: *Commun. ACM* 40 (1997), März, Nr. 3, 56–58. http://dx.doi.org/10.1145/245108.245121. – DOI 10.1145/245108.245121. – ISSN 0001–0782

[STIM09] Shidochi, Yuka ; Takahashi, Tomokazu ; Ide, Ichiro ; Murase, Hiroshi: Finding Replaceable Materials in Cooking Recipe Texts Considering Characteristic Cooking Actions. In: *Proceedings of the ACM Multimedia 2009 Workshop on Multimedia for Cooking and Eating Activities*. New York, NY, USA : ACM, 2009 (CEA '09). – ISBN 978–1–60558–763–9, 9–14

[TLA12] Teng, Chun-Yuen ; Lin, Yu-Ru ; Adamic, Lada A.: Recipe Recommendation Using Ingredient Networks. In: *Proceedings of the 3rd Annual ACM Web Science Conference*. New York, NY, USA : ACM, 2012 (WebSci '12). – ISBN 978–1–4503–1228–8, 298–307

[WGH11] Wagner, Juergen ; Geleijnse, Gijs ; Halteren, Aart van: Guidance and Support for Healthy Food Preparation in an Augmented Kitchen. In: *Proceedings of the 2011 Workshop on Context-awareness in Retrieval and Recommendation*. New York, NY, USA : ACM, 2011 (CaRR '11). – ISBN 978–1–4503–0625–6, 47–50

[WMJ03] Wrobel, Stefan ; Morik, Katharina ; Joachims, Thorsten: Maschinelles lernen und data mining. In: *Handbuch der künstlichen Intelligenz* 3 (2003), S. 517–597

[XYL10] Xie, Haoran ; Yu, Lijuan ; Li, Qing: A Hybrid Semantic Item Model for Recipe Search by Example. In: *Multimedia (ISM), 2010 IEEE International Symposium on*, 2010, S. 254–259

[YGS08] Yeung, Ching-man A. ; Gibbins, Nicholas ; Shadbolt, Nigel: A k-Nearest-Neighbour Method for Classifying Web Search Results with Data in Folksonomies. In: *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*. Washington, DC, USA : IEEE Computer Society, 2008 (WI-IAT '08). – ISBN 978–0–7695–3496–1, 70–76