

# Cooking On The Margins: Probabilistic Soft Logics for Recommending and Adapting Recipes

Johnathan Pagnutti and Jim Whitehead

{jpagnutt, ejw}@ucsc.edu  
 University of California, Santa Cruz  
 1156 High St  
 Santa Cruz, CA, 95064

**Abstract.** This paper introduces InclinedChef<sup>1</sup>, for the mixology and open challenges as part of the Computer Cooking Competition. Inclined-Chef uses Probabilistic Soft Logics (PSLs), a logic formalism that relaxes boolean logic operations to probabilities. PSLs have had good success in recommendation engines, but have yet to be applied to the Case Based Reasoning (CBR) domain. They show a lot of promise for their ability to handle contradictory information, multiple data sources and shifting user constraints.

## 1 Introduction

When it comes to cooking with computers, one of the core challenges is dealing with user preferences. The decision to decide to prepare a certain meal over another one is multifaceted, with often competing preferences. The Eating Motivation Survey contains a comprehensive model of what motivates eating and food selection, comprised of 78 individual motivations collected into 15 general clusters[10]. In this study, top food selection motivations include liking a food (taste), habits, health, convenience, and pleasure. Other research confirms taste, convenience and price as being at least as important as healthiness when people pick food choices[11].

This leads to a set of hurdles when it comes to recommending and adapting recipes for users. Information about each of these motivations often is located in separate ontologies, databases, or websites. Each of these ontologies may have contradictory information, for example: a ‘snack’ in the What We Eat In America (WWEIA) survey is different from what a ‘snack’ is in the wiki-Taaable ontology[1,3]. To complicate the problem even further, these preferences do not often line up: a healthy meal choice may be the least convenient option. Simple rule based formulations and single-heuristic optimizations cannot capture the fuzziness of the recipe domain.

We propose to use Probabilistic Soft Logics (PSLs)[2] to encode ontology information and a PSL solver to recommend and adapt recipes. A PSL program is a set of logical implications, written in a Prolog-like first-order logic syntax. This

<sup>1</sup> <http://tinyai.net/cocktails>

paper will introduce *InclinedChef*, a system that uses PSLs for recommending and adapting recipes. We will provide a short introduction into PSLs, how to encode ontology information as PSL atoms and predicates, using a PSL solver to find highly probable results for a query and highly probable adaptations for those results. We will conclude by looking at future potential applications and improvements.

## 2 Related Work

The use of first order logic to represent Case Based Reasoning (CBR) tasks is not novel. Delgrande added the ‘default’ operator to a classical first-order logic to help handle common-sense reasoning tasks[4]. CHEF[7], something of the granddaddy of CBR cooking systems, had rules that could be represented as first-order logic statements (both CHEF’s rule-based simulations of cooking and its adaptation reasoning could be expressed as a implications). Nearly every CBR system in the computer cooking competition has used rule-based formalisms to some degree, with implication (a implies b) being a core component.

Other computer cooking systems have used bottom-up learning approaches, which at their core, are inferring likely ingredient substitutions from datasets of recipes. PIERRE[9] uses a neural net to learn a regression model to find highly rated combinations of ingredients. It then uses the regression model along with a genetic algorithm to come up with new recipes.

PSLs express facts, relationships and implications using first-order logic, but with the twist that all facts and relationships have an associated probability of being true, and implications have associated weights. PSL programs are internally converted into hinge-loss Markov random fields, which are evaluated as a kind of convex optimization problem. PSLs have been used to combine several evaluation metrics for recommending restaurants and songs[8], and for learning weights on implication rules from data for textual sentiment[5].

## 3 Probabilistic Soft Logics

Probabilistic soft logics are composed of two parts, a set of predicates that describe potential facts about the world, and a set of logical implications that relate these predicates together. PSLs are different from other logic based programming paradigms in that:

1. Logical predicates are soft. Instead of predicates returning if an atom is true or false, they return the probability that an atom being true or false.
2. PSLs require that all potential atoms be represented, even those that are impossible. Therefore, PSL formulations of problems tend to be space inefficient.
3. Going along with soft predicates, implications have weights. These weights can be learned based on ground truth, or they can be used to infer the probability of new predicates given a current set. *InclinedChef* uses hand-tuned implication weights to infer the probability of new predicates.

For more detail on how InclinedChef works, see System Overview (section 4).

A PSL solver converts weighted implication rules and predicates to a Hinge-Loss Markov Random Field (HL-MRF). To discuss how, we need to first look at a PSL predicate.

$$\text{Friends}(\text{"Jon"}, \text{"Delilah"}) = 0.7$$

$$\text{Friends}(\text{"Delilah"}, \text{"Philip"})$$

The first atom states that Jon and Delilah are friends (they're arguments to the Friends predicate) with a probability of 70%. The second line states that Delilah and Philip are friends, but we don't yet know how likely that predicate is. Any atoms that are given a probability before the solver runs are treated as hard constraints. The probability is held fixed while finding the probability of the remaining predicates. This allows PSLs to pull in knowledge. Using an example from InclinedChef:

$$\text{IngGeneralizes}(\text{"Game"}, \text{"Meat"}) = 0.748$$

This states that we can generalize "game" as a "meat" with fairly high confidence. This value comes from the wikiTaaaable ontology (namely, Generalization\_costs). More information is in System Overview (section 4).

Because we'd like to discuss atoms without needing to write out their arguments, we'll use the convention Friends/2, which states that the Friends predicate takes two atoms as arguments. Lets look at a PSL implication rule. For example:

$$3 : \text{Friends}(X, Y) \wedge \text{Friends}(Y, Z) \implies \text{Friends}(X, Z)$$

This rule states that friendship is transitive: if Jon is friends with Delilah and Delilah is friends with Philip, it's likely that Jon is also friends with Philip. We also have a weight on this rule, which is how important it is in relation to other rules. Rules can also be hard constraints that must be upheld when inferring the probability of unknown atoms; hard constraints follow a slightly different syntax, using another example from an older version InclinedChef:

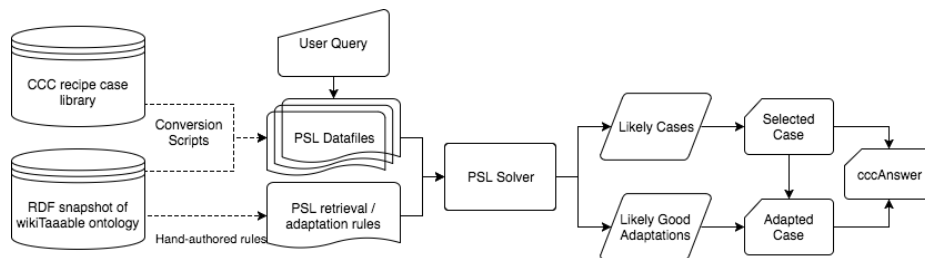
$$\text{MustContain}(I) \implies \text{TargetIngredients}(I).$$

This rule states that all MustContain/1 predicates must also be TargetIngredient/1 predicates. InclinedChef uses these constraints to insure that certain ingredients are present or not present in the adapted recipes, with more information in System Overview (section 4).

Given a set of predicates and implications, the PSL solver relaxes the boolean logic operators to continuous variables. For example, to the PSL solver,  $a \implies b$  is relaxed to  $\max(a - b, 0)$  and  $a \wedge b$  is relaxed to  $\max(a + b - 1, 0)$ . This converts the logic operators and predicates to a set of continuous valued variables, which allows the solver to define an HL-MRF over the set of unknown  $Y$  variables conditioned on the known  $X$  variables, such that the probability of  $Y$  given  $X$  is related to a hinge-loss potential function, which the solver solves using convex optimization. Due to the nature of the problem, this is parallelizable and decently fast.

## 4 System Overview

The system work can be divided into two parts—offline conversion and online recommendation and adaptation. Because PSL programs need to be in their own syntax, information from datasets and ontologies needs to be converted to atoms and rules. After this step, the solver can use the rules to perform recommendation and adaptation.



**Fig. 1.** Diagram of ImpliedChef. Dashed lines are offline, solid lines are online. Using conversion scripts and hand authoring, the XML case library and RDF ontology snapshot are converted to PSL atoms and rules. A user query is converted to a set of PSL atoms as well. This is passed to the solver, which returns as PSL atoms a set of likely good cases and likely good adaptations. These are used for case retrieval and case adaptation, which are then passed back as an answer to the query.

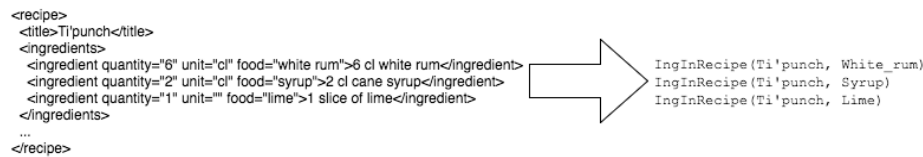
Figure 4 shows a boxes and arrows diagram of ImpliedChef. Offline (before any user queries are processed), we use a set of conversion scripts and hand authoring to get a set of PSL atoms, predicates and rules from the wikiTaaable ontology and Computer Cooking Competition case library. Online, a user query is converted to a set of PSL atoms that interact with PSL logic rules. All of this is passed to the PSL solver, which is given two important targets: a set of atoms that represent each case in the case library and a set of atoms that represent recipe adaptations. These are not the only atoms that the solver is inferring probabilities for, but are used in the next steps.

The case atoms are used to select a case from the library. The adaptation atoms are used to figure out how to tweak the case to fit the constraints provided by the user query as well as the the challenge at hand. Finally, these are wrapped up as a XML document that follows the cccSystemOutput XML Schema definition for the competition.

### 4.1 Offline Components

These parts of ImpliedChef are performed only once, and are done before any user queries are ever fielded by the system. The first few of these convert recipes into sets of ingredient atoms, as shown in figure 4.1.

We specify that the `IngInRecipe/2` predicate is closed, which means the solver should treat all predicates of that type as observed, true data, and furthermore,



**Fig. 2.** Converting CCC XML case libraries into PSL atoms. We focused on ingredients for ImpliedChef

should not attempt to infer probabilities on them. *IngInRecipe/2* simply encodes which recipes contain which ingredients. We discuss, under Future Work, how PSL might consider the amounts of each ingredient in its solving steps. We also use a set of scripts that converts information from the RDF snapshot of the *wikiTaaable* ontology to PSL atoms, for the *IngGeneralizes/2* predicate. A few examples are:

$$\begin{aligned}
 \text{IngGeneralizes}(\text{Pork\_side\_cuts}, \text{Pork}) &= 0.968 \\
 \text{IngGeneralizes}(\text{Red\_martini}, \text{Martini}) &= 0.500 \\
 \text{IngGeneralizes}(\text{Fresh\_bean}, \text{Vegetable}) &= 0.422
 \end{aligned}$$

These atoms are calculated by subtracting the *Generalization\_cost* of an ingredient or ingredient class by 1. These *Generalization\_costs* are retrieved from the RDF snapshot. However, not all ingredients have *Generalization\_costs* in the *wikiTaaable* ontology. For those that do not, we use the *subclassOf* feature and assign only a probability of 0.5 to this being a correct way to generalize about an ingredient.

The result is that, as the *Generalization\_cost* decreases, the more the solver considers a generalization as likely. Greater generalization costs (representing ingredient classes further away from each other) are less and less likely. The RDF snapshot only provides costs for ingredients next to each other, but it can be useful for retrieval and adaptation to consider generalizations further away, such as *IngGeneralizes*(Rye\_whiskey, Alcohol) or *IngGeneralizes*(Veal\_leg\_cut, Meat). We append these case generalizations without providing probabilities, and use the following rule to let the solver figure out how likely each of these cases should be:

$$\text{IngGeneralizes}(C1, C2) \wedge \text{IngGeneralizes}(C2, C3) \implies \text{IngGeneralizes}(C1, C3)$$

## 4.2 Online Components

After a user query is parsed, it is turned into two sets of atoms for the predicates *MustContain/1* and *MustNotContain/1*. These contain the ingredients specified by the user query about ingredients they want and do not want in a resultant recipe. For retrieval, these are considered soft constraints, as we can adapt any ill-fitting recipes, but for adaptation, they are hard constraints. In addition, to fit with the cocktail challenge, we added the reduced list of ingredients as a set of *MustContain/1* atoms.

For retrieval, we'd like to consider each recipe in terms of the classes that its ingredients generalize to. We capture this with a few rules:

$$\begin{aligned}
& \text{IngInRecipe}(R, I) \wedge \text{IngGeneralizes}(I, C) \implies \text{RecipeClasses}(R, C) \\
& \text{RecipeClasses}(R, C1) \wedge \text{IngGeneralizes}(C1, C2) \implies \text{RecipeClasses}(R, C2) \\
& \text{MustContain}(I) \wedge \text{IngInRecipe}(R, I) \implies \text{RecommendTarget}(R) \\
& \quad \text{MustContain}(I) \wedge \text{IngGeneralizes}(I, C) \wedge \\
& \quad \text{RecipeClasses}(R, C) \implies \text{RecommendTarget}(R) \\
& \text{MustNotContain}(I) \wedge \text{IngInRecipe}(R, I) \implies \neg \text{RecommendTarget}(R) \\
& \quad \text{MustNotContain}(I) \wedge \text{IngGeneralizes}(I, C) \wedge \\
& \quad \text{RecipeClasses}(R, C) \implies \neg \text{RecommendTarget}(R)
\end{aligned}$$

The first two rules let us consider a recipe based on the ingredient classes that it's composed of. The next set of rules let us use that information. We want to make recipes that contain ingredients in a user's query more likely. Furthermore, we also want to establish that any recipes that have ingredients that generalize to the same classes as an ingredient a user wants are more likely. The inverse goes for ingredients that a user does not want.

For adapting a retrieved case, we need to be a little creative. Because PSLs require all atoms implied by their predicates, we can't simply derive a probability for any two potential ingredients to swap. Considering only the 155 ingredients used in the CCC cocktail case library, we would need to infer probabilities on 155! predicates.

To get around this bottleneck, we consider two rules of thumb. It's best to perform the bare minimum number of swaps to satisfy a query and we only need to either swap in or out ingredients that are part of the user's query

Therefore, we inspect a user's query and generate the atoms in the Swap/2 predicate for each query. Each atom in Swap/2 contains a ingredient that the user has specified they wish to have or not have in the resultant recipe. We perform swaps with the following rules:

$$\begin{aligned}
& \text{MustContain}(I1) \wedge \text{RecommendTarget}(R) \wedge \text{IngGeneralizes}(I1, C) \wedge \\
& \quad \text{IngGeneralizes}(I2, C) \wedge \text{IngInRecipe}(R, I2) \implies \text{Swap}(I1, I2) \\
& \text{MustNotContain}(I1) \wedge \text{RecommendTarget}(R) \wedge \text{IngGeneralizes}(I1, C) \wedge \\
& \quad \text{IngGeneralizes}(I2, C) \wedge \text{IngInRecipe}(R, I2) \implies \text{Swap}(I2, I1)
\end{aligned}$$

Swaps are always read the same way, the first atom in the predicate is replacing the second. Due to the fact that the Swap/2 predicate is built on the fly, we don't need to specify a hard constraints that an element must be present. Probabilities are only inferred on the atoms present in the Swap/2 data files, and all of those atoms are related to a user's query. We then build the answer XML file based on the RecommendTarget/1 and Swap/2 predicates, as shown in figure 4.2.

The recommendation rules give us a set of probabilities on the Recommend-Target/1 predicates, however, unless the user was very, very specific with a query, several atoms are equally likely to fit. We take the set of the highest probable

```

RECOMMENDTARGET(Gin_fizz_easy) = 0.877
RECOMMENDTARGET(Cocktail_paradise) = 0.750
RECOMMENDTARGET(The_kiss_of_Smurfette) = 0.994
RECOMMENDTARGET(Cocktail_de_fruits) = 0.474
RECOMMENDTARGET(Eggnog) = 0.000

SWAP(Apple_juice,Sour_cream) = 0.000
SWAP(Apple_juice,Pear_liqueur) = 0.750
SWAP(Apple_juice,Beer) = 0.891
SWAP(Apple_juice,Orange_syrup) = 0.750
SWAP(Apple_juice,Pineapple) = 0.000

```

**Fig. 3.** Some example values of a query looking for a cocktail containing Apple juice

atoms and chose one between them. We then retrieve that case from the CCC case library and use it as part of the retrieve half.

The adaptation rules give us a set of probabilities on potential swaps. We scan the Swap/2 predicate for the highest probability swaps that involve both the user’s query and the retrieved recipe. We adapt the recipe by keeping swap quantity units and amounts the same, but changing the resultant ingredients.

## 5 Conclusion

We hope that ImpliedChef shows how PSLs can be used for CBR tasks like retrieval and adaptation. However, there are many extensions possible while using PSL as a framework.

The overview of PSL (and the rules that ImpliedChef uses) have kept to logical, boolean operations. PSL also supports arithmetic operations, such as sum constraints (the values of atoms need to sum to a particular value). PSL even affords substituting an arbitrary number of atoms as part of a logical rule with sum-augmented predicates, which work like placeholders for the sum of an arbitrary number of atoms. Select-statements restrict which atoms can be swapped in for sum-augmentation, so one can imagine encoding the amount of each ingredient in a recipe as a percentage, then using sum-augmentation and selection to be sensitive to ingredient amounts when adapting recipes.

In addition, PSL supports the use of arbitrary functions as part of implication rules, as long as those functions can take in string arguments and return a real value from  $[0, 1]$ .

$$Name(P1, N1) \wedge Name(P2, N2) \wedge Similar(N1, N2) \implies Same(P1, P2)$$

The above rule, for example, relates two people atoms ( $P$ ) by their names ( $N$ ). Similar/2 is a functional predicate, an external function that takes in two names and returns how similar they are from  $[0, 1]$ . This sort of technique allows for the unification of many ways to measure similarity, from WordNet comparisons to similarity metrics built from Long-Short Term Memory networks. More importantly, though, using several different rules with a variety of recommendation heuristics, we can tune the weights on the rules to fit a variety of user preferences.

In the current iteration of ImpliedChef, all of its atoms and rules come from the wikiTaaaable ontology. Other food ontologies exist that also have entities with labelings, such as the Foodon ontology [6]. Converting the relevant parts of other ontologies and integrating them into ImpliedChef is currently ongoing work.

Reflecting back on the opening problems, we can see that ImpliedChef shows how PSLs provide approaches for solving them. Recipe data and ontology information can be converted into PSL rules and predicates, and PSLs can be used for retrieval and adaptation CBR tasks. Furthermore, PSLs are able to reason over ingredient amounts, able to combine conflicting heuristic scores and able to pull in reasoning from multiple ontologies. They seem to be a powerful, general framework for tackling the semantically rich space of recipe generation.

## References

1. Ahuja, J., Montville, J., Omolewa-Tomobi, G., Heendeniya, K., Martin, C., Steinfeldt, L., Anand, J., Adler, M., LaComb, R., Moshfegh, A.: Usda food and nutrient database for dietary studies, 5.0. US Department of Agriculture, Agricultural Research Service, Food Surveys Research Group (2012)
2. Bach, S.H., Broecheler, M., Huang, B., Getoor, L.: Hinge-loss markov random fields and probabilistic soft logic arXiv:1505.04406 [cs.LG] (2015)
3. Badra, F., Cojan, J., Cordier, A., Lieber, J., Meilender, T., Mille, A., Molli, P., Nauer, E., Napoli, A., Skaf-Molli, H., et al.: Knowledge acquisition and discovery for the textual case-based cooking system wikitaable. In: 8th International Conference on Case-Based Reasoning-ICCBR 2009, Workshop Proceedings. pp. 249–258 (2009)
4. Delgrande, J.P.: An approach to default reasoning based on a first-order conditional logic: revised report. *Artificial intelligence* 36(1), 63–90 (1988)
5. Foulds, J., Kumar, S., Getoor, L.: Latent topic networks: A versatile probabilistic programming framework for topic models. In: International Conference on Machine Learning. pp. 777–786 (2015)
6. Griffiths, E., Brinkman, F., Dooley, D., Hsiao, W., Buttigieg, P., Hoehndorf, R.: Foodon: A global farm-to-fork food ontology
7. Hammond, K.J.: Chef: A model of case-based planning. In: AAAI. pp. 267–271 (1986)
8. Kouki, P., Fakhraei, S., Foulds, J., Eirinaki, M., Getoor, L.: Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems. In: Proceedings of the 9th ACM Conference on Recommender Systems. pp. 99–106. ACM (2015)
9. Morris, R.G., Burton, S.H., Bodily, P.M., Ventura, D.: Soup over bean of pure joy: Culinary ruminations of an artificial chef. In: Proceedings of the 3rd International Conference on Computational Creativity. pp. 119–125 (2012)
10. Renner, B., Sproesser, G., Strohbach, S., Schupp, H.T.: Why we eat what we eat. the eating motivation survey (tems). *Appetite* 59(1), 117–128 (2012)
11. Steptoe, A., Pollard, T.M., Wardle, J.: Development of a measure of the motives underlying the selection of food: the food choice questionnaire. *Appetite* 25(3), 267–284 (1995)