NYU Technical Report 960

Diet Planner: finding a nutritionally sound diet while following (most) of a dieter's desires

Mick Jermsak Jermsurawong
Dennis Shasha

Abstract We describe the design and implementation of a diet website currently housed at http://nutrientdata.herokuapp.com/ . The site allows users or dieticians to enter nutritional constraints (e.g. at least this much calcium but not more than that amount of calcium) and objectives (e.g. minimize calories), a list of foods/brands the person likes. The site then determines, if possible, the quantities of at least some of those desired foods that would meet the nutritional constraints. If not possible, then the site guides the user in the choice of other foods that may meet the nutritional constraints. The net result is a tailored diet measured in servings.

Reviews of Existing Websites

Many nutrition websites rely on the public nutrient database from the United States Department of Agriculture (USDA), which has common food items and their nutritional content. Often the websites further integrate their own food data, which may be more brand-specific. Nutritiondata.self.com, further complements that by allowing users to create their own recipes. The website's database also includes its own nutritional indices such as a caloric ratio pyramid, which shows the distribution of calories among carbohydrates, proteins and fats, and an inflammation index which predicts the inflammatory effects of foods.

Among food logging sites in which users record the food they consume, CalorieCount.com calculates the overall calories from the food logged, and offers nutritional advice as milestones for users to achieve. By contrast, CalorieCamp makes food logging social. Users can comment on the food logs, encourage other users, and applaud other users' achievements. Paying users in CalorieCount.com can get nutritional advice from registered dieticians based on their logged diet. Another similar site is Shopwell.com which focuses on the grocery shopping experience. The user creates his/her shopping list and the website recommends healthier food options based on the condition of the users. Shopwell.com includes a bar code scanning mobile application to retrieve information of that food products while shopping. Shopwell then provides real-time feedback for healthier but similar food products to the user.

These commercial websites generally inform users and provide soft guidance for better food selections. However, they do not enforce any nutritional constraints for the users. Nor do they suggest specific amounts of each food item that should be consumed. The available nutrition websites may be helpful for people whose nutritional recommendations are not very strict. However, medical patients may have strict nutritional needs. Currently, there are no tools for quantitative food recommendation according to nutritional constraints. This leads to the second part of the report.

Need for this Diet Planner Website

Currently, nutritionists make diet recommendation based on hand-calculations. This first involves informed selection of food items based on a patient's desires. Then, further refinement for food quantities is necessary to cater to the patient's specific nutritional needs. Lastly, as medical patients' conditions change, continual adjustment of the food recommendation is required. Manual calculations in such processes are slow, error-prone, and may be limited by the range of foods with which a dietician is familiar.

Another issue is patients' preference. Often diet planned for medical patients often do not sufficiently consider patient's personal taste. It is not always the case that preferred foods are unhealthy ones. Relatively healthy foods that the patients like should be included in the diet, although the diet should still be complemented with necessary food items to satisfy the nutritional needs of the patients.

This website therefore aims to address the two problems by providing an accurate and efficient method to plan diets according to nutritional needs, while adequately catering to food preferences of medical patients.

User Interface

There are four major steps in the course of constructing a diet. First, the user fills in profile information as a guest or logs in to retrieve the his or her previously entered profile. This profile information is necessary to create default nutritional constraints. Second, the user (perhaps with the help of a dietician) can manage his/her nutritional needs and objectisves. Third, the user can browse foods and select the preferred foods. Finally, the user gets the recommended diet plans perhaps after including some additional system-recommended foods.

Fill in Profile

The user's personal information is used for calculating nutritional constraints. First, calorie needs are calculated by Harris Benedict Equation [1]:

Women: BMR = 655 + (9.6 x Weight[kg]) + (1.8 x Height[cm]) - (4.7 x age in years)
Men: BMR = 66 + (13.7 x Weight[kg]) + (5 x Height[cm]) - (6.8 x age in years)

Second, default nutritional constraints are taken from dietary reference intakes report [2] based on the user's age, condition, and gender.

For macronutrients, the lower default limits are based on Recommended Dietary Allowances (RDAs). "RDA is the average daily dietary intake level; sufficient to meet the nutrient requirements of nearly all (97-98 percent) healthy individuals in a group." The upper default limits are based on the upper values of Acceptable Macronutrient Distribution Ranges.

For micronutrients, the lower default limits are based on Estimated Average Requirements (EARs). "EAR is the average daily nutrient intake level estimated to meet the requirements of half of the healthy individuals in a group." The upper limit default limits are based on Tolerable Upper Intake Levels, which shows "the highest level of daily nutrient intake that is likely to pose no risk of adverse health effects to almost all individuals in the general population."

Manage Diet
The user can choose which nutrient to minimize or maximize. This will generate a diet that is low or high in that chosen nutrient. The user can also choose between two default nutrient plans, a basic nutrient plan or a plan with a full list of nutrients including minerals and vitamins. Then the user can modify the nutritional constraints according to the user's specific needs. These nutritional constraints will be saved to user's account for subsequent use.

Select Foods

The user can browse foods by categories or search foods by keywords. The user can sort the returned foods by nutritional contents and view the nutritional label by simply mousing over the item. The user can choose any food item by clicking it and that food item will appear in the box "Foods I Like." The list of chosen foods can be edited, and submitted for generating diet plan. All the foods in "Foods I Like" will be saved to the user's account for subsequent use.

Get Diet Plan

There are three possible states in the diet plan generated. First, "Diet Not Optimized. Following Constraints Unmet" is shown as remarks. Although some foods are recommended, their total nutritional contents do not satisfy the nutritional constraints the user has specified. The user is then offered ways to complement the diet with suggested new foods. After the user chooses various foods, the suggested foods will change. At some point, there will be no more recommendations and the user will be put in the state: "Diet Optimized. Minimum Constraints Met." However, at this point, the food recommendations may exceed the upper limits of the specified nutritional constraints. The user should add more foods by clicking at the link "Please Add More Foods." User will be directed to *Select Foods* to add more foods. Afterwards, generating the diet plan will likely to bring user to a satisfactory state.

Internal Implementation
The implementation is separated into three parts: getting the data, database development, and web application development. The code can be accessed at https://github.com/jj1192/nutrientdata

Getting the data

Library to work with is Beautiful soup. The source of data is from a commercial website. This website has the nutritional content of general food items that are mainly from the public nutrient database from USDA http://ndb.nal.usda.gov/. In addition, it has information of brand-specific food items.

The scraper is built with the beautiful soup library. There are two scrapers working on the two layers of the website. First, the scraper gets all the links of food main categories and their subcategories. The result is in “IndexCategories.txt”. The main categories numbered with negative numbers from -23 to -1, and the subcategories are from 0 to 826. Using this file, another code “getfoodNutrientDescription.py” goes through these subcategories and traverse through all the pages within each category.

In each page, there are two main sections of food’s nutritional content, basic food nutrients and extra list of nutrients. All foods come with the basic nutrients, and only some has the extra nutrients. Usually the foods taken from USDA database come with the extra nutrients. The complete list of the basic nutrients is known, while that of the extra nutrients is not. Therefore, there is a separate file to record the growing list of these extra nutrients as the scraper works through the food items.

```
baked products|chocolate chip muffins|Mini Chocoate Chip
Muffin|NA|Hostess|3|pieces|34|150|72|8.0|12|2.0|NA|NA|20|100|17.0|1.0|10.0|2.0|0|0
|2|4|^NA|150|NA|2.01|7.99|NA|17|1|10|NA|NA|NA|NA|NA|NA|NA|20|0.71|NA|NA|NA|
100|NA|NA|NA|NA|~|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|~|NA|NA|NA|N
A|NA|NA|NA|NA|2.006|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA
|NA|NA|NA|NA|20|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|
NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|
NA|NA|NA|~|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|NA|N
A|NA|NA|NA|
```

From the example above, the format of the result data is separated into two parts by ^. The first part includes the food’s details and its basic nutritional content, and the second is only the extra nutrients. Each value is separated by | (because some of the food details already include “,” in it.) In the first part, the first eight corresponds to main type, type, food item, detail, source, serving size, unit of serving size, and weight in gram respectively. What follows is the information the basic nutrient. These basic nutrients are ordered according to how they are represented in nutritional content label. Now the second part, those nutrients extra nutrients that are ordered according to this file “ex_nutrient_cat.txt”. This file is actually accidentally generated from first scraping without the recording down the detail of the food. Contrary to what is expected that the list is growing while scraping, the complete list is known before the second try of scraping. This makes it easier to populate the database later.

The information that is not available is recorded as NA the extra nutrient information should be recorded with an inde that are not available are recorded as NA and those that have almost negligible content are recorded as ~. All data files of the food are in files “result-23” to “result826.”

Database development

Postgresql is used as a choice of relational database management system with the intention to exploit its full-text search. SQLalchemy, a database toolkit for python, is also used to simplify the integration with other parts of the project done in python. It also allows manipulation of data through more familiar python language. More importantly, its Object Relational Mapper (ORM), treating data point as an object, allows more flexible coding. The data points have their own methods. For example, a food data point can return its nutrition density (nutrients per calorie) or a user data point can return his/her status. The description of the code to manage the database will be explained in the next section in Flask-SQLalchemy. Mainly, they are code to create, populate, migrate and upgrade the database.

The database has four tables: User, Food, FoodKey, and Nutri. First, User table has basic information of user such as age, height, and gender. These are required for calculating default nutritional constraints. Next, Food object represent a food item with nutritional content and their basic information as described above in getting the data. FoodKey is a table that facilitate searching of food items which will be explained in searchResult() function. Lastly, Nutri object keeps data of nutritional constraint for the user. There are relationships between User and Food, and between User and Nutri. One Food object can belong to many Users and one User can have many food objects. This keeps record of what are the foods that users like. Nutrition also has a foreign key to User, indicating the nutritional recommendation that the user last sets.

Web Application Development

Flask is used as the framework for this application. It allows the separation of functionality and the design of the application. On the functionality, it handles all the GET requests through the functions and POST functions through another extension called WTFormsthat It works with jinja2 template to allow control structure in writing html pages.

The flow of the application as explained in the user interface is the followings: fill in profile, manage diet, select foods, and generate diet. Before getting to the details of the four parts, there are two global objects that help to bring different parts together. First is session object. Session object is a dictionary that allows storing information specific to the user from one request to the next. This is one way of passing of data across web pages. Also, note that is session is different from session from SQLalchemy. Second is g object. The function "before_request()" calls the current_user to this global object. It verifies whether the user is logged into the system, and if the user is not logged in, it assigns the basic keys inside session object preparing it for other functions.

The first stage is filling in the profile (profile.html). Users have two options, to log in (login()) as current user, to fill in the profile and become a guest. After this point, all the users are logged in, but However, the system is designed to log everyone into the system, however it still can differentiate the user with column role. If the user does sign-up his role is 0 being a normal user, else he is a guest role value being 1. All the users are logged in to go through the same pipeline. The difference is that when guest leaves, his account, his User object and associated Food and Nutri objects are deleted (logout()).

In managing nutritional constraints (manage.html and manage()),CreateMinMaxForm() is used to generate forms with the default values after the user have specified the nutrient plan. The default values taken are the check on the boxes of the constraints, nutrient plan, diet plan, and the text on the submit button (This form is reused again in get diet.) When this form is submitted, the new nutritional constraints are updated on Nutri object and the session is committed after that.

Next, selecting food includes two main functions, resultSearch() and resultCategory(). Each follows the same flow for users to select the food. However, they diverge in the method to return food results. Method to get food from a chosen category is just a simple filtering query Food.type == chosen Category. When the user click on one of the food categories give, the chosen type will be passed with the URL to this function. However, the search function from input keywords is slightly more complicated.

In function searchFood(), upon getting search term from the form SearchForm(), it requires a new table FoodKey(keyID, word). Each record is a unique and lower-case keyword from the food object which that food id. Given a search term "boiled eggs," this term is split and each is searched against FoodKey. All the FoodKey that has these keywords is made into subquery.

```
        a = FoodKey.query.filter(FoodKey.word.in_(keywords)).subquery()
```

Then joining Food items with that ids from the FoodKeyfilter(Food.id==a.c.keyid) and group them by .group_by(Food.id), imposing condition that Food returned has all the keywords .having(func.count(distinct(a.c.word)) == len(keywords))

```
q=Food.query.filter(Food.id==a.c.keyid).group_by(Food.id).having(func.count(distinct(a
.c.word)) == len(keywords))
```

To rank search results according to relevancy, sorting the food according to length sufficiently does the job. Because since all the foods have all the keywords given, the shortest one is likely to be more relevant. For example, searching "chocolate" will give "chocolate" before "chocolate milk."

In resultSearch(), the given search term is also compared against the food categories to give potential food categories that the user might need. In essence, upon searching, user gets specific results and the scope of potential results.

The two functions resultSearch() and resultCategory() converges to the same flow in ranking the food by nutritional content. The filter is remembered only until new search result is given, or new food category is chosen.

Also in selecting the food to "Food I Likes" box, a linking functions selectFood() and "selectFoodFromSearch() are called. The functions do the same jobs, simply add the food id to the session dictionary, and return respectively to resultSearch() or resultCategory()

After selecting all the foods, the user can manage their preferred foods (foodsILike()). In this function, the same filter to that on the food items is applied. However, the default is sorting foods by categories.

Lastly, the linear programming part uses pulp and glpk solver in optimize(). The function to do actual lp work is linearOptimize(). It takes the list of Foods, the nutrients that the user wants to constraint, list of constraints, and diet plan as arguments. The documentation of this lp module can be found here
http://www.coin-or.org/PuLP/CaseStudies/a_blending_problem.html
It is usually unlikely that the foods first submitted can be optimized. Often it cannot satisfy certain constraints because the food items are too limited. However, the lp

module does give infeasible solution, and the nutritional constraints unsatisfied are then examined. Concretely, if the initial status given is infeasible the objective is temporally changed to maximize and linearOptimize() is called again just to examine which lower constraints are not satisfied. These constraints are raised to the user and the user is suggested to complement them by going to resultSuggest page. This corresponds to "“Diet Not Optimized. Following Constraints Unmet” as remarks explained in the user interface.

resultSuggest() follows a similar flow to resultSearch() and resultCategory(), in terms selecting foods. However, for a given nutrient, the food suggested are the ones that guarantee that minimum constraint can be satisfied with reasonable portions. As user selects food, a function selectFoodFromSuggest() is called to examine the current set of chosen foods whether it satisfy the nutrients raised.

If the solution is still infeasible and that all lower constraints are met, this means that the upper constraints are too strict. The upper constraints are adjusted to very high value of 5000 so that the food diet can be optimized. Lp solvers are called reiteratively as these upper constraints are lowered, until solution becomes infeasible. The diet will exceed certain nutrients, but by a reasonable amount. This corresponds to “Diet Optimized. Minimum Constraints Met”, as explained in the user interface.

Table 1 below summarizes all the functions used in the web development.

Functions	Return	Details
Select Food		
getSearchEntry(brandEntry,searchEntry)	searchEntry [string]	Formats the search terms from the two forms of search boxes, food items and brands
getSearchTerms(searchTerms)	foods and its brand [string]	Inverses getSearchEntry()
getMatchingCat(searchEntry,foodTypes)	matchingCat [list strings]	Gets matching categories from search Term
searchFood(searchTerm, brandTerm, Food,FoodKey)	foodIDs [list int]	Get ids of foods that tag has all the search terms
searchFoodBrand(brandTerm, Food)	foodIDs [list int]	Get ids of foods with the specified brand
resultSearch()	renders resultSearch.html	Search page
selectFoodFromSearch(foodIDFromSearch)	redirects to resultSearch.html	Inserts food chosen into list of preferred foods
resultCategory(categoryChosen)	renders sresultCategory.html	Main browse food page
selectFood(foodChosen)	redirects to resultCategory.html	Inserts food chosen into list of preferred foods
mainCat(mainCatChosen)	redirects to resultCategory.html	Passes the chosen category through query strings
saveFood()		Commits to database the foods that user likes
foodsILike()	renders	Allows sorting of foods I like and

	foodsILike.html	to view these foods more easily
Mange Diet		
getKeysBounds(nutriObject ,override)	check [list 0's and 1's], nutriField [list field objects], defaultGenlowerBound, defautGenupperBound [list string]	Gets bounds from default nutri object, constraints in binary digit.
getCal(height, height2, weight, USorMetric, gender, activity,age)	calories [float]	Calculates calories based on Harris Benedict Equation
getageGroup(age)	ageGroup [string]	Gets the ageGroup for "look-up table" of nutri objects
manage()	renders manage.html	Manage diet page
Get Diet Plan		
reportRatio(constraints, foodItems, nutri)	givenCal [float], failedBestFood [food], nutRatioMinNew [list float], nutRatioUnmetNew [list int]	Given list of foods and constraints, finds which nutrients are still lacking (also with information from infeasible solution)
reportTotal(constraints, outputFoodAmount, foodItems)	TotalNut [list float]	Gets total nutritional content in order of the constraints given
showExceed(xNut, totalNut, foodAmount, foodItems)	majorPercent[list string]	Given a nutrient exceeded, gives an ordered contribution of each food item
linearOptimize(listFoodObject,constraints, constraintsGivenmin, constraintsGivenmax, opt_maxormin, opt_nut)	outputFood [list string], outputFoodAmount [list float] , status [string] ,objective [float], nullNut [list int])	Gets diet plan according to constraints and objective
optimize()	renders optimize.html	Generates diet using linear programming
resultSuggest()	renders resultSuggest.html	Suggests more foods more the user to select
selectNut(chosenNut)	redirects to resultSuggest.html	Stores the chosen nutrient to be satisfied
selectTypeILike(chosenType)	redirects to resultSuggest.html	Stores the type of foods to complement the diet
selectFoodFromSuggest(foodIDFromSuggest)	redirects to resultSuggest.html	Adds to the list of foods submitted and shows the nutrients that has yet to be

		satisfied
Admin		
getUserProfileDisplay(user)	userProfile [string]	Generate profile detail on the navigation on top
profile()	renders profile.html	Home page that user can edit profile
login()	renders login.html	Allows logging in current users or creating new profile for guests
logout()	redirects to login.html	Logs out user – and if it is guest delete his data
signup()	renders signup.html	Allows user to give username and password to a guest account

Table 1: Summary of Functions Used in the Web Application

Future Work

Currently, the user can only choose from the prepared meals or basic food items. We would like the user to be able to input his/her own recipes, calculate its overall nutritional content based on those ingredients, and save them as preferred food items. In addition, an option to suggest a set of preferred foods for users to start with based on their profile will lessen the work on the user.

Acknowledgement

We owe very much to Maria Petagna for her valuable feedback on the system. Advice given by Joe Jean has been a great help in deploying this web application on a cloud host. We also would like to acknowledge support given by Juan Felipe Beltran and Ling Liang Zhang.

Works Cited

[1] J. Arthur, Harris, and Benidict Francis G. *A biometric study of basal metabolism in man* . Washington Carnegie Institution of Washington, 1919. Print.

[2] United States. Institute of Medicine of the National Academies. *Dietary Reference Intakes DRI The Essential Guide to Nutrient Requirements*. 2006. Web. <http://www.nal.usda.gov/fnic/DRI/Essential_Guide/DRIEssentialGuideNutReq.pdf>.