# **Demonstrator Design**

## Overview

The basic idea behind the demonstrator ‘Semantic Product Reviewer’ is that this product crawls through the website <http://www.productreview.com.au> . Currently the demonstrator doesn’t search for the product using any sort of searching algorithm, as it might get too complex. However that functionality is intended to be used in the **Big Picture (kindly refer to the Big Picture** **section**).

We have selected 5 proteins supplement providers and 2 Internet service providers (ISPs) to analyze the comments of, as follows:

* Venom Protein
* Whey Gold Standard Protein
* Vital Protein Pea Protein Isolate
* Whey Protein Isolate
* Swisse Men’s Ultivite Formula 1 Protein
* Beagle Telecom ISP
* Bigpond ISP

The product will generate the complete analysis of each product based on the user selection. The attributes generated for each products are as follows:

* Product Name
* Average product rating & out of rating
* Total Positive, Negative and Undetermined comments
* Top comment by each user on the website
* The interpretation of each top comment (Positive Or Negative Or Undetermined)
* Individual user rating

Based on the number of positive comments and negative comments about the product, a user can determine whether the product held a positive feedback or a negative feedback.

The product developed is basically a website. We are using the following platform for software development.

* Framework Used - .NET Framework 4.0
* Language Used - ASP.NET
* Backend Language – C#, Web APIs
* Frontend Languages – CSS, HTML, Javascript, JSON

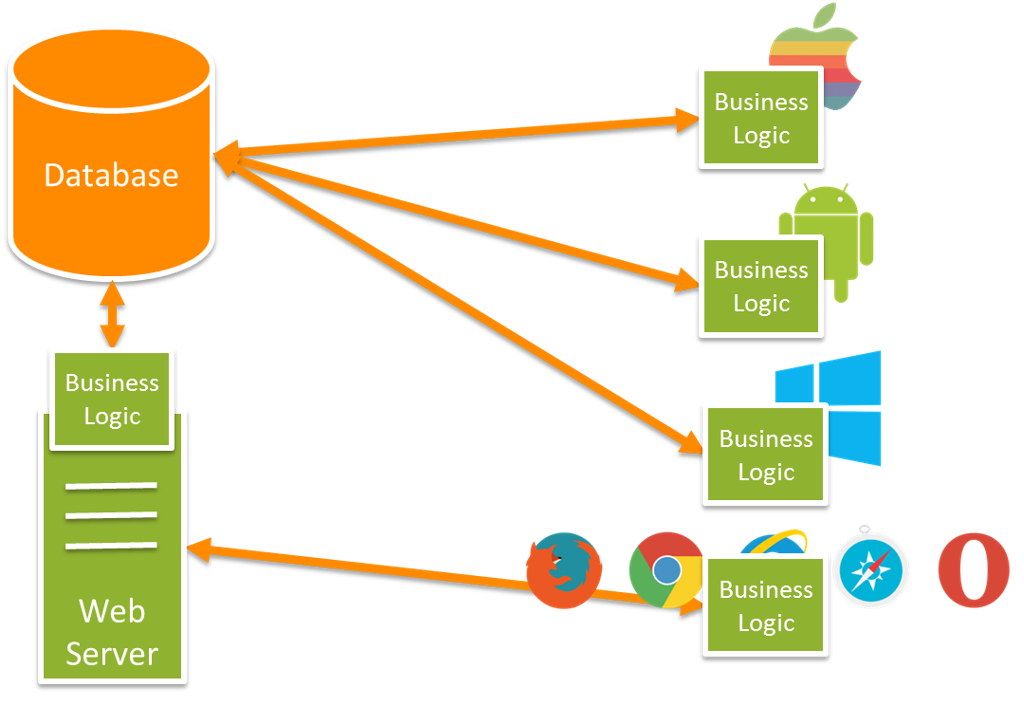
## Design

The basic design pillars of the demonstrators are as follows:

A KIMONO Web API (Existing Agent):

In order to get the data as an input to the Natural Language Analyzer, We decided to use ‘Kimono Web APIs’ as data extractor from the product review website.

**A Web API** is a service which has an ‘in-built’ business intelligence which executes necessary set of rules and provides a useful data in an XML or JSON format which can be directly used by other applications or in the development code.



The ‘KIMONO LABS’ is a start-up company who specialize in producing Web APIs which can crawl through any website and get the data from specific sections of the website and outputs them in JSON format. One has to enroll into their website in order to create an account and avail their services.

We selected <http://www.productreview.com.au/p/venom-protein.html> this website for our first protein. Kindly note that the procedure/ structure of the JSON Dataset for all the other products is similar

The application of the link will generate a JSON link which produces the data in JSON format. For your reference <https://www.kimonolabs.com/api/du33b7qw?apikey=rIUTL1gnwlZf0c0S8aDdLfGpMPGblfhN>

Upon clicking this link, you would be able to view the JSON data sample. The format of the JSON object created was something like following.

*Fig. 3.3 (REFERENCE PROPERLY)*

Now we convert this JSON data in to the class structure of C# in order to communicate this data with our application.

**Values Added by of KIMONO Web API:** The important aspect of this external agent was that we didn’t have to create a program to extract a type of data from the website we were analyzing. It saved us a lot of time for the work which is just an entrance point of this project. Kindly note that data gathering is not the major and intelligent part of this system, hence we proceeded with this method.

Also this API generates real time data every 15 minutes. Hence it can be said it is as real time as we can get.

Class clsJSON

For communicating with the ASP.NET Application, we need the JSON data to be in Object oriented format, so that it can be used in a C# application at the backend. Hence we used serialization concept to convert JSON into class structure. Serialization converts any JSON data in a UML class diagram structure of C# code. The class diagram generated in the process is shown below.

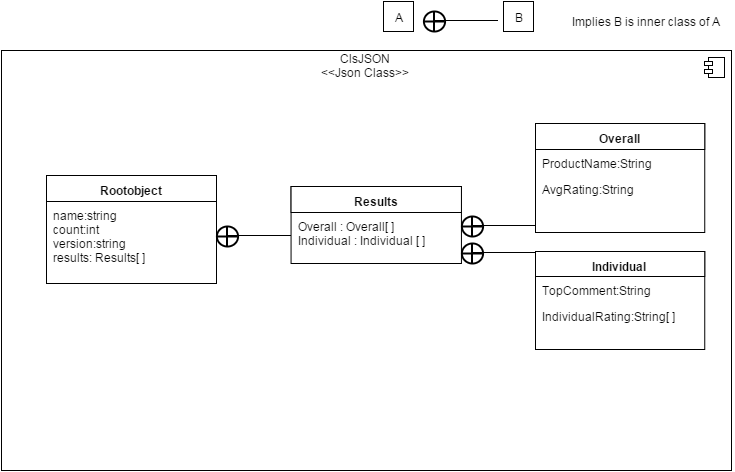


Fig. 3.3 (REFERENCE PROPERLY)

The above functionality converts JSON data into a ‘VAR’ object of c#, which can be used in any format. Currently we use the VAR object into strings to show on the front end.

## Naïve Bayes Classifier Algorithm

The Bayesian Classifier is fit for figuring the most likely yield contingent upon the data. It is conceivable to include new crude information at runtime and have a superior probabilistic classifier. A Naïve Bayes classifier accept that the vicinity (or nonappearance) of a specific component of a class is random to the vicinity (or nonattendance) of whatever other element, given the class variable. For instance, a natural product may be thought to be an apple on the off chance that it is red, round, and around 4" in width. Regardless of the possibility that these components rely on upon one another or upon the presence of different elements, a credulous Bayes classifier considers these properties to autonomously add to the likelihood this organic product is an apple.

Despite a lot of controversies, it has proved to be a lot successful in the classification terminologies.

The formula goes like this:

http://www.codeproject.com/KB/recipes/318126/da9282959fd82f789e0725509f1985a4.png

Probability Model

### **Naïve Bayes Interpretation:**

In the Bayesian interpretation, likelihood measures a level of conviction. Bayes' hypothesis then calculates the level of confidence in a suggestion previously, then after the fact representing confirmation. For instance, assume some individual suggests that a one-sided coin is twice as liable to land heads as tails. Level of faith in this may at first be half. The currency is then flipped various times to gather proof. Conviction may ascend to 70% if the proof backings the suggestion. For proposition A and proof B,

P (A), the earlier, is the starting level of confidence in A.

P (A | B), the back, is the level of conviction having represented B.

P (B | A)/P (B) speaks to the bolster B accommodates A.

### **Paul Graham Spam Filter:**

Using the probability theorem, Paul Graham put forward a strategy to deal with spams. It is quite a tedious and difficult task to differentiate spam email from a regular email. Though this could be achieved more effectively by Paul Graham’s spa filter algorithm. But how this could be used for **Natural Language Sentiment analysis**? It is explained in detail in the upcoming sections. But first it is necessary to understand the spam filter algorithm we used for our project. Each class with their description is mentioned in the below subsections. The whole Naïve Bayes Classifier algorithm is in the ‘**BAYSIAN’** folder of the source code. The step by step approach used in the project is given below.

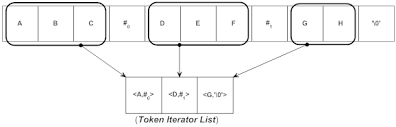
1. **Creating Nodes:** This is the first step in the process. For a classifier to work, classification classes are necessary. For example, a shape can be classified as a Square or a Circle. In this example Square and Circle are the classes which will determine the input’s likelihood of resemblance to themselves. These will be called **Nodes** in our approach. In our case, the 2 nodes are Positive and Negative as we need to determine the polarity of a word or sentence.

The classes required for this algorithm are listed below:

* **Baysian.Node**: Creates Nodes
* **Baysian.MemNodes**: Allocates space to nodes
* **Baysian.Initializer**: Initializes any incoming document string.

1. **Tokenization:** Every sentence is converted to various tokens. For example,

**‘**I like this product’ will be converted to 4 tokens. ‘I’, ‘Like’, ‘this’, ‘product’



The actual process of tokenization includes a lot of complications including looking for symbols like apostrophe, comma, exclamation marks etc.

The classes required for this algorithm are listed below:

* **Baysian.Initializer**: For Parsing the string into tokens
* **Baysian.MemNodes**: calculates total tokens allocated in the given Node.
* **Baysian.NodesTable**: Keeps track of all the tokens added to all the nodes in a tabular format.

1. **Porter Stemming Algorithm:** Some of the tokens are various forms of the single word. For example,

Liking, Liked, Like -> Like

Sadly, Sadden, Sad -> Sad

Hated, Hating, Hate -> Hate

Every word has its root origins. In order to train the Machine Algorithm that is to be implemented in our project, we need to stem each word and then train the dictionary.

For finding out the root of the word, we used a stemming algorithm called **‘Porter Stemmer Algorithm’**. Kindly note that not all the roots of words might have meaning. For example,

Abate, abated, abatement, abatements, abates -> abat

The algorithm works in following steps:

1. First of all, the stemming algorithm will get rid of all the plurals of the word along with ‘–ed’ and ‘-ing’ suffices. Hence, Swimming -> swim, Likes / Liked -> like.
2. Turns terminal ‘y’ to ‘I’ when there is another vowel in the stem. For example, Furry -> furri
3. Maps double suffixes to single ones. For example, Operational -> Operate, Realization -> realize, Possibly -> Possibli -> Possible
4. Deals with suffixes, -full, -ness etc. For example, Felicity -> felicity -> felic, Practical -> Practic, Largeness -> Large
5. Takes off -ant, -ence, etc. For example, Operational -> Operate -> oper, Controllable -> control
6. Removes a final –e. For example, Deflate -> deflat etc.

However, this algorithm is not applicable to all the words, for example Satisfied -> Satisf and Satisfy -> Safisfi are not the same. There are some limitations to this algorithm, but in most cases it works just fine.

The classes required for this algorithm are listed below:

* **IStemmer** Interface: Used to define a method Stem(string)
* **Stemmer** class: Performs all the 6 steps mentioned above
* **Among** class: Used to perform substring operations to determine if the substring is a part of the other main string.

1. **Negation Algorithm for NLP:**

After taking care of the stemmers, we need to define an algorithm which would be necessary for our Natural Language Sentiment Analysis.

As we are not considering very complex statements for our demonstrator, we will be considering simple statements as we are only going to analyze the top comment of the review website.

The algorithm runs in the following way.

If (Found(,) in the string)

Split Sentences based on , /\*(Commas)\*/

Perform the function again on each split string

If ( Found(!) or Found(?) or (/) or (\) )

Remove these special signs.

If ( Found (‘not’) or Found (‘n’t’) or Found (‘not’) or Found (‘never)

Add ‘NOT\_’ before every word until the end.

Example of this algorithm would be like this:

‘I don’t like this’ -> I don’t NOT\_like NOT\_this.

After applying this algorithm, every word is added in Dictionary. For every word added in a not, its negation is added in to its opposite node. For example, if the word ‘Like’ is added in Positive word dictionary, ‘NOT\_like’ is added in the Negative dictionary.

New flow of operation for adding words in dictionary is as follows.

The classes required for this algorithm are listed below:

* **Dictionary** Class: Used to prepare the dictionary and apply the negation NLP algorithm

1. **Baysian Analyzer / Probability Calculator:**

After adding the words/ sentences to the dictionary, or training the dictionary, we need to analyze the statement and determine whether it belongs to Positive node or Negative Node. The algorithm for determining that is as follows.

1. Get the input string.
2. Tokenize the string.
3. Stem each token.
4. Apply Negation NLP and append ‘NOT\_’ to some of the words as per the negation algorithm.
5. Assign Variable Positivecount = No. of appearances of the token in Positive dictionary.
6. Assign Variable Negativecount = No. of appearances of the token in Negative dictionary.

bw = Positivecount / PositiveTotalCount;

gw = Negativcount / NegativeTotalCount;

pw = ((bw) / ((bw) + (gw)));

s = 1f,

x = .5f,

n = cat1count + cat2count;

fw = ((s \* x) + (n \* pw)) / (s + n);

1. The ‘fw’ term gives the Baysian probability of that token being a positive word.
2. If the combined probability of all the tokens is >= 0.55 then the sentence belongs to **positive** dictionary.
3. If the combined probability of all the tokens is <= 0.45 then the sentence belongs to **negative** dictionary.
4. If the combined probability of all the tokens is > 0.45 and < 0.55 then the sentence belongs to **undetermined** dictionary.

As you keep on training the dictionary well, It will increase the total no of words found in the respective dictionary more, increasing the probability amount and making it sure of the Node. The well you train, the better you get the accuracy.

The training of the dictionary is not limited to words, it can be an idiom, Saying, sentence or even a whole paragraph.

For example, ‘I like it so bad’ is a positive sentence. But ‘Like’ comes under Positive dictionary, and ‘Bad’ comes under a negative dictionary. Still if you train your Positive dictionary by Adding a saying ‘Liking badly’ to the positive dictionary, will train the dictionary to add more number of root words ‘Like’ and ‘Bad’ in the positive dictionary, hence it will only produce positive interpretation of the sentence when ‘I like it so bad’ sentence is input.

Hence, now the final flow chart will be as follows.

The classes required for this algorithm are listed below:

* **Baysian.NodesTable** Class: Used to hold the Nodes and words count in the table dictionary
* **Baysian.Algorithm** Class: Used for calculating probability and decision making whether it belongs to Negative or Positive node.

## Front End

The front end in the application has been developed in ASP.NET editor Visual Studio 2013. It uses html, Javascript and bootstrap css.

# **Criticism of the Demonstrator**

‘Semantic Analyzer Reviewer’ is an intelligent agent as It uses machine learning concepts, keeps on training and enhancing its analytical performance based on the training provided to the dictionaries, and it crawls through the website in order to fetch the useful data as well. The characteristics shown by the demonstrator are as follows.

**Situatedness**: This demonstrator is a website, hence it can be hosted on any machine and act as a standalone product providing uninterrupted service.

**Persistence & Reactivity:** Thisdemonstrator keeps on learning by next mistakes, hence it doesn’t give up on wrong outputs. It is pretty fast as the Web API gathers & processes the data within milliseconds

**Proactivity:** Asthe data is coming in per 15 minutes, this demonstrator is as real time and proactive as possible. It satisfies a huge goal of humanity too by understanding human language.

**Learning & Autonomy:** As it is based on machine learning, it is autonomous and will keep on improving as the training is given.

As we know no system is full proof, especially when there is artificial intelligence along with natural language processing is involved in the application development. But the demonstrator that we developed has shown above 70 – 80 % accuracy on sentiments we received from the website. Still there is a lot of room for improvement for the demonstrator. Some of the limitations are listed below.

* The demonstrator fails to consider the complicated statements such as ‘I don’t say no to this product’. The sentiment of the product is supposed to be slightly positive, but we believe by training the demonstrator with such examples along with more complicated NLP negation algorithm, the performance can be attained above 90%.
* The demonstrator also fails to take account of the spelling mistakes, punctuation mistakes made by the user as the rules for grammar and the dictionary words are based on the English language rules.
* The demonstrator is considering all the top comments made by every visiting user on that page. As this demonstrator can’t distinguish between an authentic and fake user, the demonstrator can’t generate useful statistics in that case.
* The demonstrator is currently statically getting the website data, may be in future by changing some of the code or algorithm, it can be made dynamic search engine enhanced version.
* When the stemmer algorithm generates faulty roots of the words, the Semantic Analyzer Reviewer fails to produce correct outputs. For example ‘Satisfied’ produces Undetermined because the dictionary has a word ‘Satisfy’ which roots to Satisfi and not ‘Satisf’.
* As the big picture says, the demonstrator should be used and trained more on a bigger platform such as Facebook posts or comments or Tweets from social networking sites as it will generate a very useful data and revolutionize the product review.