‘WHAT’S COOKING?’

A SNEAK PEEK INTO CLASSIFIERS AND WEB SCRAPING

CDS 2015, Group 4
Bodhisattwa Prasad Majumder (11)
Dattatreya Biswas (13)
Keshav Sehgal (20)
Rachit Tripathi (33)


500PX
‘EVERY METHOD HAS ITS OWN *FLAVOR*!’

MENU

Starters

Introduction to Kaggle Dataset
Data Preparation, Feature Engineering

Main Course

Application of different algorithms
Naïve Bayes
XGBoost
Indian Database
Introduction to Scrapy
Data preparation ~ spell-check, stemming
Feature Engineering ~ Spice Dictionary
New Cluster set
Algorithms considered

Dessert

Importance Matrix and Importance Plot
Challenges and Future Scope

INTRODUCTION TO kaggle DATASET

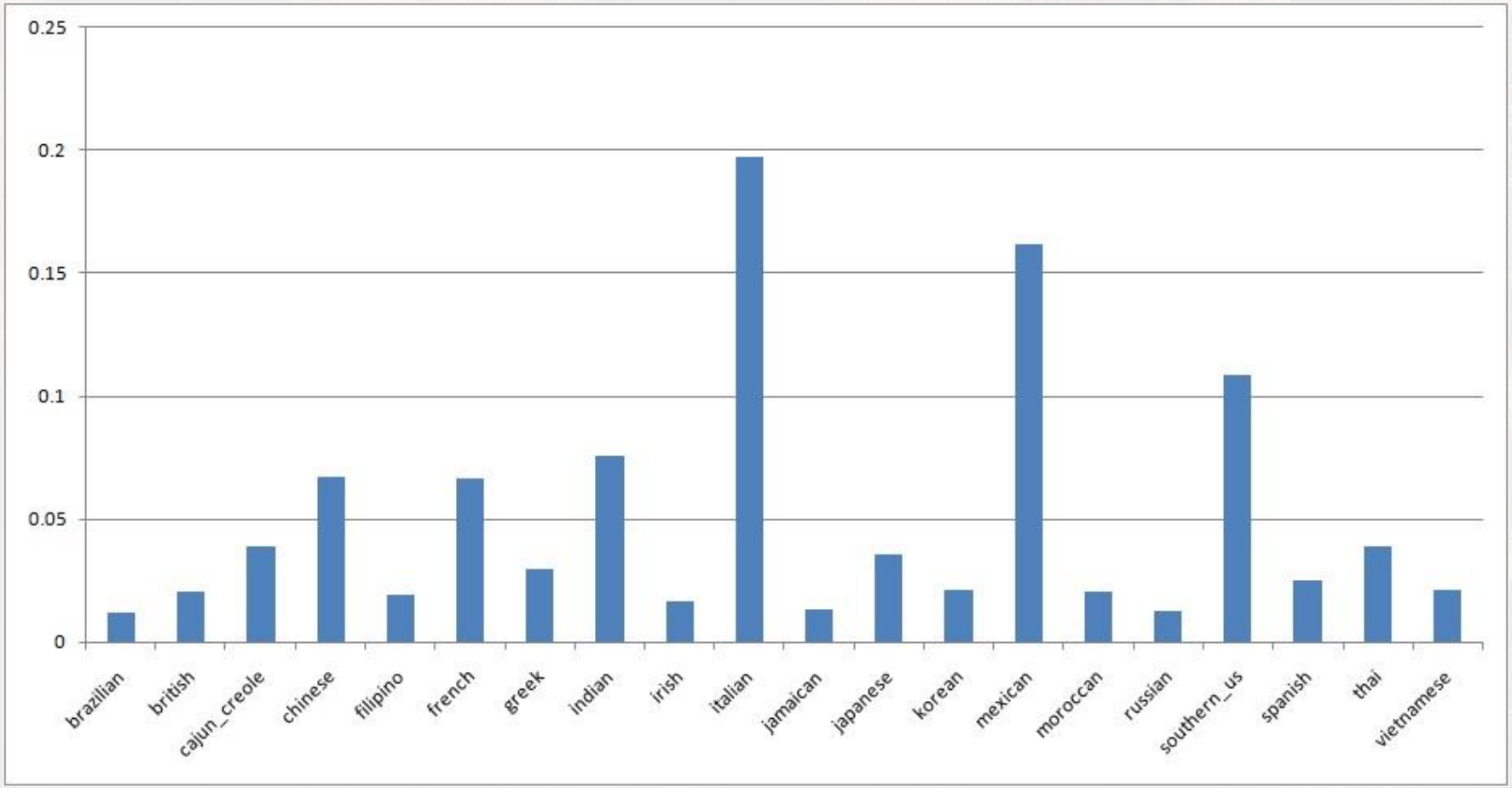
Objective of the Kaggle competition named "What's Cooking" was to

Use recipe ingredients to categorize the cuisine

- Train data consists of ***39774*** dish-ids along with their ingredients and cuisine
- Test data consists of ***9944*** dish-ids along with their ingredients
- There are ***20*** cuisines listed in data.

INTRODUCTION TO kaggle DATASET

- There are ***20*** cuisines listed in data. An comprehensive view of dishes per cuisine would be:



DATA PREPARATION

- **Tokenization**:

We have tokenized all the ingredients as a single word to emphasis the pattern present in data. For example: Red Chili Pepper would appear after tokenization as Red_Chili_Pepper. This would minimize misclassification error from taking Red, Chili and Pepper as three separate tokens.

- **Case modification**:

We deliberately changed all text data in lower case to avoid inconsistency.

- **Sparse term modification**:

After creating document term matrix we removed terms those occurred for less than ***3*** times. This was a ***heuristic*** choice which we achieved plugging different numbers. This method is useful when tokenization ***hasn't*** been used.

- **Porter's stemming algorithm**:

This is a process for removing the commoner ***morphological*** and ***inflexional*** endings from words in English.

Example: tomatoes, tomato – will be considered same.

FEATURE ENGINEERING

- As the number of ingredients for dishes varies from ***1*** (minimum) to ***135*** (maximum), we decided to take ***number of ingredients per dish*** as a feature.

- Our assumption of this being a important feature can be verified from importance plot.


SO?
‘LET’S
COOK’

DOCUMENT TERM MATRIX

- From the text data (set of ingredients), we have created document term matrix (DTM).
- 'tm' package in R creates DTM
- From the corpus of ingredients, the method sets '***1***' when that particular ingredient appears for a dish and '***0***' otherwise.
- Sparse terms from DTM has been removed where frequency of those terms for all dishes are ***less than 3***
- Number of ingredients per dish has been added as an extra feature in DTM.

ALGORITHMS CONSIDERED

- *Decision Tree (CART)*
- *Random Forest*
- *Naïve Bayes Classifier*
- *Modifications of Naïve Bayes*
- *Extreme Gradient Boosting – XGBoost*

NAÏVE BAYES BASIC PROBABILITY THEOREM

- Bayesian classifiers use Bayes theorem, which says

$$P(C_j|d) = P(d|C_j) * \frac{P(C_j)}{P(d)}$$

- $P(d)$ = Probability of occurrence of particular dish.
- $P(Cj)$ = Probability of occurrence of particular Cuisine.
- $P(Cj|\ d)$ = Probability of particular dish being in cuisine Cj. (we want to find out this)
- $P(d|\ Cj)$ = Probability of occurrence of particular dish given a Cuisine. (we can find this using Training data)

ASSUMPTIONS FOR APPLYING NAÏVE BAYES

- Assumption 1: The occurrence of ingredients is ***not correlated***. This means that the probability of occurring of ingredient is independent of other ingredient present in the dish.

$$P(I_i|I_j) = P(I_i) \quad \forall\, i \in [1,m]\,, j \in [1,m]$$

- Assumption 2: The probability of occurring of a dish in a cuisine is product of the probabilities of all the ingredients in a dish, i.e. ***dishes are independent***.

$$P(d|C_j) = P(I_1|C_j) * P(I_2|C_j) * P(I_3|C_j) * P(I_4|C_j) \ldots\ldots * P(I_m|C_j)$$

- The main task left is calculating $P(I_j\,|C_i)$

NAÏVE BAYES CLASSIFIER

As the DTM is concerned:

All the ingredients appearing in the Training data

All the Dishes in the training data

Id	Milk	Olive_oil	Salt	Corn	Sugar	Bread	……	Meat
10259	0	0	1	1	0	1		0
25693	0	1	1	1	0	0		1
76420	1	0	0	1	1	0		0
79310	0	0	0	0	0	1		0
.....	1	1	1	0	0	0		0
12831	0	1	1	0	0	0		0

All the matrices appearing in following slides is just for representation, any resemblance to real life data is purely coincidental

THE PROBABILITY MATRIX

- We took the number of appearance of each ingredient for each cuisine.

All the ingredients appearing in the Training data

All 20 Cuisine in the training data

Id	Milk	Olive_oil	Salt	Corn	Sugar	Bread	……	Meat
Greek	121	213	1242	231	720	121		98
Indian	723	98	1702	432	931	34		123
Italian	123	213	2753	1231	1231	131		312
Mexcian	312	534	764	67	76	42		98
.....	…	…	….	…	…	…		..
French	234	321	1632	232	324	756		123

Take the column sum and divide each column by its column sum to obtain column stochastic matrix.

THE PROBABILITY MATRIX

Matrix *P*: ***Probability Matrix***

All the ingredients appearing in the Training data

All 20 Cuisine in the training data

Id	Milk	Olive_oil	Salt	Corn	Sugar	Bread	……	Meat
Greek	0.121	0.0213	0.1242	0.231	0.0720	0.0121		0.098
Indian	0.072	0.098	0.1702	0.432	0.0931	0.034		0.0123
Italian	0.123	0.0213	0.0275	0.1231	0.1231	0.0131		0.0312
Mexican	0.031	0.0534	0.0764	0.067	0.076	0.042		0.098
.....	…	…	….	…	…	…		..
French	0.234	0.0321	0.01632	0.0232	0.0324	0.0756		0.0123

All the columns sum to one.

NAÏVE BAYES CLASSIFICATION

- The i, j^{th} element of P matrix is $P(I_j | C_i)$.

$$P(d|C_j) = P(I_1|C_j) * P(I_2|C_j) * P(I_3|C_j) * P(I_4|C_j) \ldots\ldots * (I_m|C_j)$$

- Using the above equation and Probability matrix calculated we can find the probability of a given cuisine for a particular dish.

$$P(C_j|d) = P(d|C_j) * P(C_j)$$

- A given dish is classified to a cuisine which gives maximum probability of belonging to a particular cuisine.
- **Note**: Though the probability of each cuisine can be calculated from data, i.e. the sample proportion of each cuisine, but we need to be careful since that may not reflect the ***population proportion***.

MODIFICATION IN NAÏVE BAYES

- When we are classifying through Naïve Bayes probability matrix, if in training data probability of a ingredient appearing in the cuisine is zero then the whole probability will be zero regardless of other probabilities.
- Solution can be taking the geometric mean of the entries in a row which are non zero.
- In a situation, if there are only few non zero entries in a row which will make geometric mean higher than above and in these situations, we can get wrong classifications.
- To avoid previous situation, we maintained a threshold which is defined as the total number of non zero ingredients should be greater than particular value.

RESULT FOR NAÏVE BAYES CLASSIFIERS

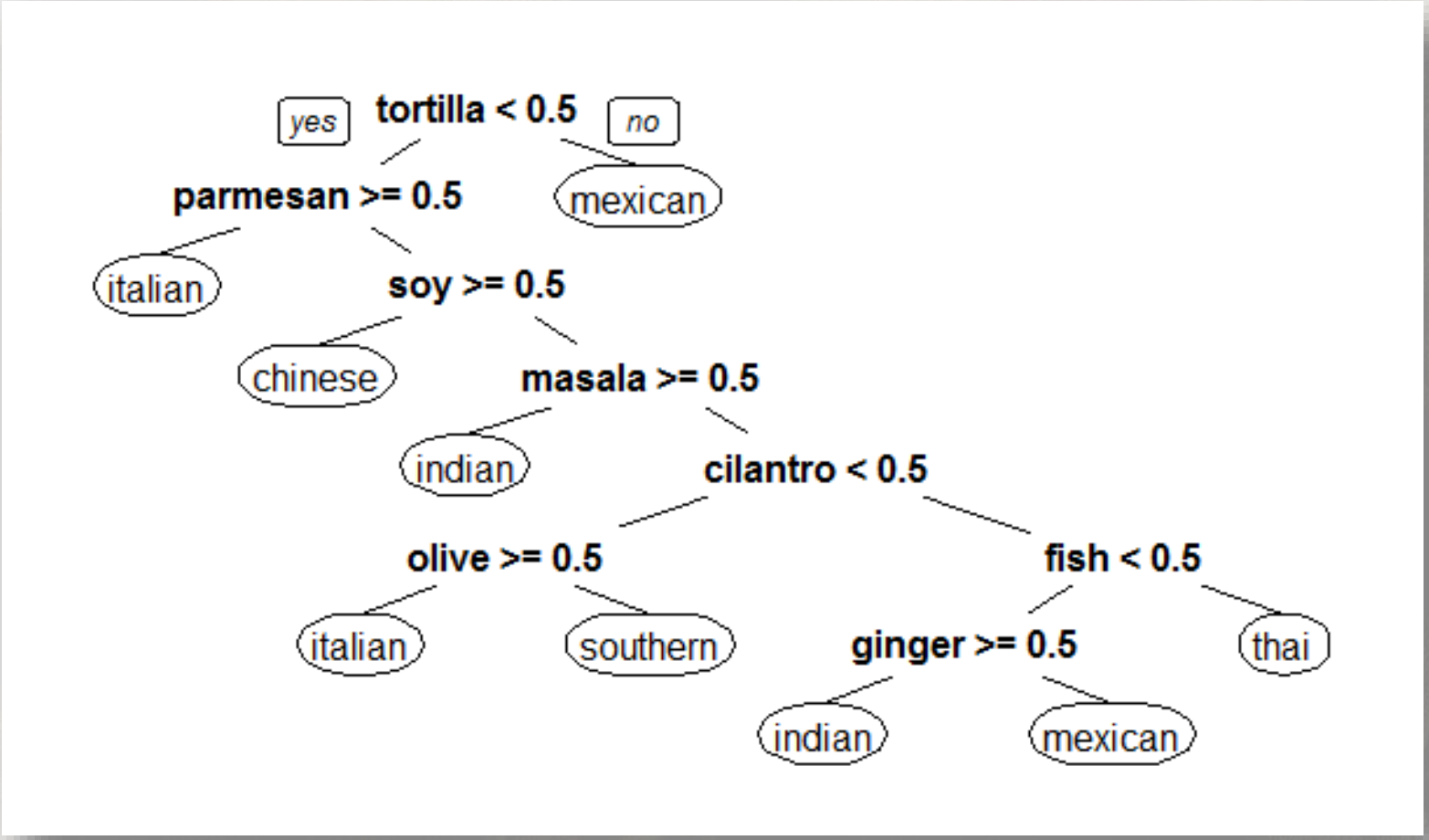
Methods	Kaggle score (Accuracy)
Probability multiplication	0.58558
Probability addition	0.44571
Population proportion of cuisines (assuming uniform)	**0.59584**
Modified Naïve Bayes with geometric mean	0.56074
Modified Naïve Bayes allowing minimum number of non-zero terms as 4	0.56114
Modified Naïve Bayes allowing minimum number of non-zero terms as 10	0.56647

RESULT FOR RANDOM FOREST AND CART MODEL

- For classification, CART model and Random forest are well known methods. Specially on categorical variable, trees work well.
- Our attempt for Random forest shoot up the accuracy of predicting test data to ***0.76348***
- Decision tree over-fits the data hence gives bad accuracy, as applied on test data and generated an accuracy of ***0.417***

RESULT FOR RANDOM FOREST AND CART MODEL

- Possible structure of single decision tree would be:




500px
‘XGBOOST!’

MOTIVATION

- The motivation for boosting was a procedure that combines the outputs of many "***weak***" classifiers to produce a powerful "***committee***."
- The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a ***sequence of weak classifiers***
- Boosting is a way of fitting an additive expansion in a set of elementary "***basis***" functions. Here the basis functions are the individual classifiers

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

REDEFINING TREES

- To start the supervised learning, consider tree as our weak learner.
- Perspective:

  Regression tree is a function that maps the attributes to the score.

Let's refine the definition of tree:

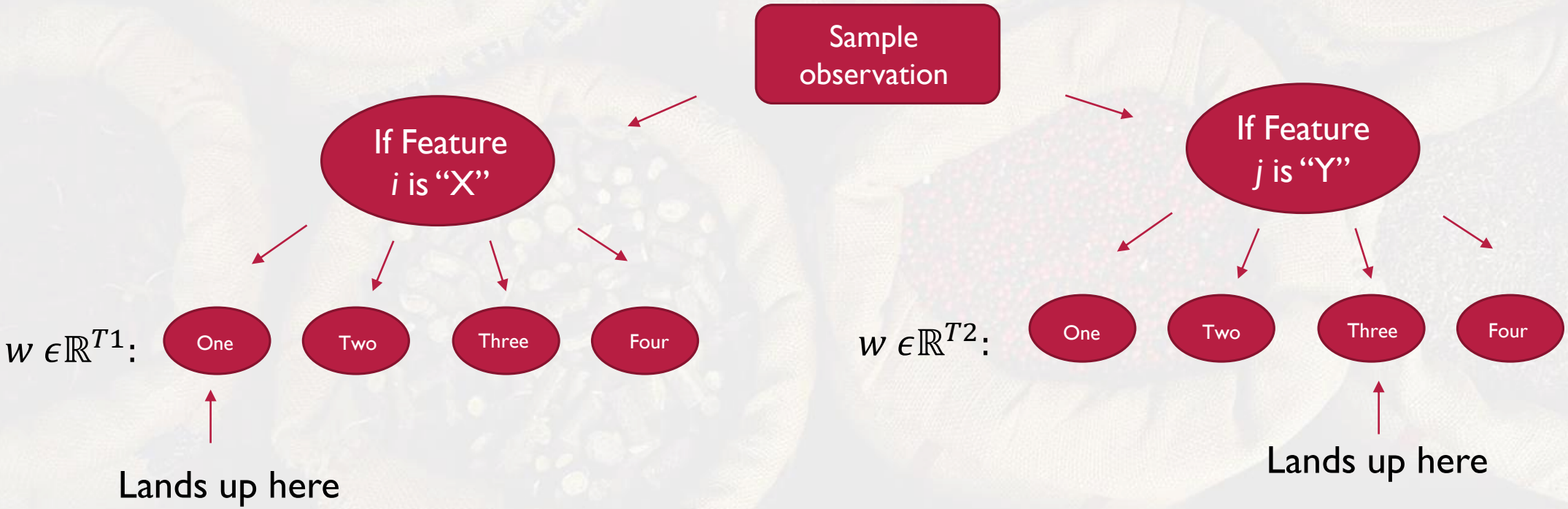
- We define tree by a ***vector of scores in leafs***, and ***a leaf index mapping function*** that maps an instance to a leaf

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}$$

Leaf weight on the leaves (scores)

Structure of the tree

TREE ENSEMBLE



- The ***structure*** of individual tree $(q(x))$ guides a sample to a leaf and the ***associated score*** w_i is assigned as the prediction for that sample for that tree

MATHEMATICAL MODELLING

- The ***prediction model*** $(\hat{y})$ can be written as the aggregation of all the prediction score for each tree for a ***sample*** (x). Particularly for i-th sample,

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

where K is the number of trees, f is the function in the functional space $\mathcal{F}$ and $\mathcal{F}$ is the all possible set of trees having prediction score in each leaf, slightly different from decision tree which only contains decision values.

OBJECTIVE FUNCTION

- Objective to minimize:

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

- Optimizing training loss encourages ***predictive*** models
- Optimizing regularization encourages ***simple*** models to have smaller variance in future predictions, making prediction ***stable***

OBJECTIVE (CONTD.)

- In tree ensemble model, the obvious choice of loss function is the square loss function. Hence,

$$obj(\Theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

So far so good..

But how this model is *different* from ***Random Forest model***?

- The fact is Random Forest and Boosted Trees are not different in terms of model, the difference is how we train them.
- The major reason is in terms of training objective, Boosted Trees tries to add new trees (additive training) that compliments the already built ones. This normally gives you ***better accuracy with less trees***.

ADDITIVE TRAINING

- Consider the task as a regression over a set of function which is much harder than the traditional optimization techniques (***Stochastic Gradient Descent***)
- An ***additive*** strategy has been taken to add a new tree at each iteration. Starting from constant prediction, the task is to add a new function each time (iteration).

$$
\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}
$$

ADDITIVE TRAINING

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$
$$= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$$

$$Obj^{(t)} = \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \Omega(f_i)$$
$$= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + constant$$

$$Obj^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

$$g_i = \partial_{\hat{y}_i^{(t)}} l(y_i, \hat{y}_i^{(t-1)})$$
$$h_i = \partial^2_{\hat{y}_i^{(t)}} l(y_i, \hat{y}_i^{(t-1)})$$

Optimization goal

MODEL COMPLEXITY

- Model complexity comprises ***number of leaves*** in a tree and ***L2 norm of the scores on leaves*** which ensures normalization of leaf scores.

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

To derive an expression for structure score, the re-written objective function in terms of scores would be:

$$Obj^{(t)} \approx \sum_{i=1}^{n} [g_i w_q(x_i) + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T$$

OPTIMAL SCORE AND OBJECTIVE REDUCTION

- In each iteration, we are able to choose an optimized tree which optimizes the objective function which has been already optimized partly up to previous iteration, which ensures better accuracy.
- The ***optimal score*** would look like:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

where $G_j = \sum_{i \in I_j} g_i$, $H_j = \sum_{i \in I_j} h_i$

- The optimal score is the best score function for a given structure of tree and ***optimal objective reduction*** measures how good a tree structure is.

$$Obj^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

Measures goodness of a tree

PRACTICAL GREEDY APPROACH

- Due to impossibility of enumerating all the tree from the function space, a greedy approach is of practical use which ensure ***an optimal split***.
- Gain for a split can be formulated by:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

- the score on the new left leaf
- the score on the new right leaf
- the score on the original leaf
- regularization on the additional leaf

- If gain is smaller than γ, we would better not to add that branch, which is nothing but ***prunning***!

RESULT FOR XGBOOST

XGBoost with different parameters	Kaggle score
Eta (learning rate) = 0.25, maximum depth= 10, L2 regularization	0.77826
Eta (learning rate) = 0.10, maximum depth= 25, L2 regularization	0.78962
Eta (learning rate) = 0.10, maximum depth= 25, L1 regularization	**0.79133**

kaggle **Leaderboard rank 175 among 978 participants**

kaggle **Highest accuracy achieved 0.82271**

PARAMETER TUNING AND COMPARISON

- The present data is not in accordance of the basic assumption of Naïve Bayes, i.e. ***independence of ingredients*** (or cannot be guaranteed).
- Like milk and sour things can not occur together, similarly ***various relationship*** holds between various ingredients which can not be explained by Naïve Bayes.
- A single decision tree tends to ***over-fit*** the model and having features over two thousands, the expected ***accuracy*** for a single decision tree would be ***low***.
- Random Forest nearly gives accuracy like XGBoost since the ensemble of weak learner predict far better than a single decision tree. But the assumption of randomness is not always desired. There is ***no control*** for the complexity of the weak learners hence no control on over-fitting. To achieve higher accuracy than random forest each tree needs to be optimally chosen such that the loss function is minimized at its best.
- Thus XGBoost comes into play. XGBoost is an ***optimized random forest***. Advantage of boosted tree is the algorithm works very fast on a distributed system (XGBoost package does)

PARAMETER TUNING AND COMPARISON

- The idea of ***reinforcement learning*** is somewhat analogical here in case of XGBoost which is not present in Random forest classifiers
- We tuned parameter ***eta*** which is used to prevent over-fitting by making the boosting process more conservative. Low eta value means model more robust to over-fitting but slower to compute. In our case, we are over cautious to over-fitting hence eta **0.1** gave the best result
- Optimizing function XGBoost used to do multiclass classification is the ***softmax objective***. Softmax objective (cost function) is minimised through XGBoost.
- ***L1 regularization*** on leaf weights performs better than L2 regularization because it encourages the lower weighted features to be dropped while modelling, making model ***simpler***


‘SOMETHING
REFRESHING!’

DATABASE ON INDIAN DISHES

- Above experiment encouraged us to dig out our own database for Indian cuisines
- We have scraped our data from a cooking website www.vahrehvahchef.com founded by Sanjay Thumma.

- This website is extremely popular among Indian expatriates in Europe, Australia, and North America.
- It presents recipes of different dishes from 28 states of India.

HOW TO GET DATA? *SCRAPY* IS HERE!

- **What is a *web crawler* (web spider)?**

A web crawler can systematically extract data from multiple webpages by crawling, or accessing, webpages and returns the data you need for each one of them.

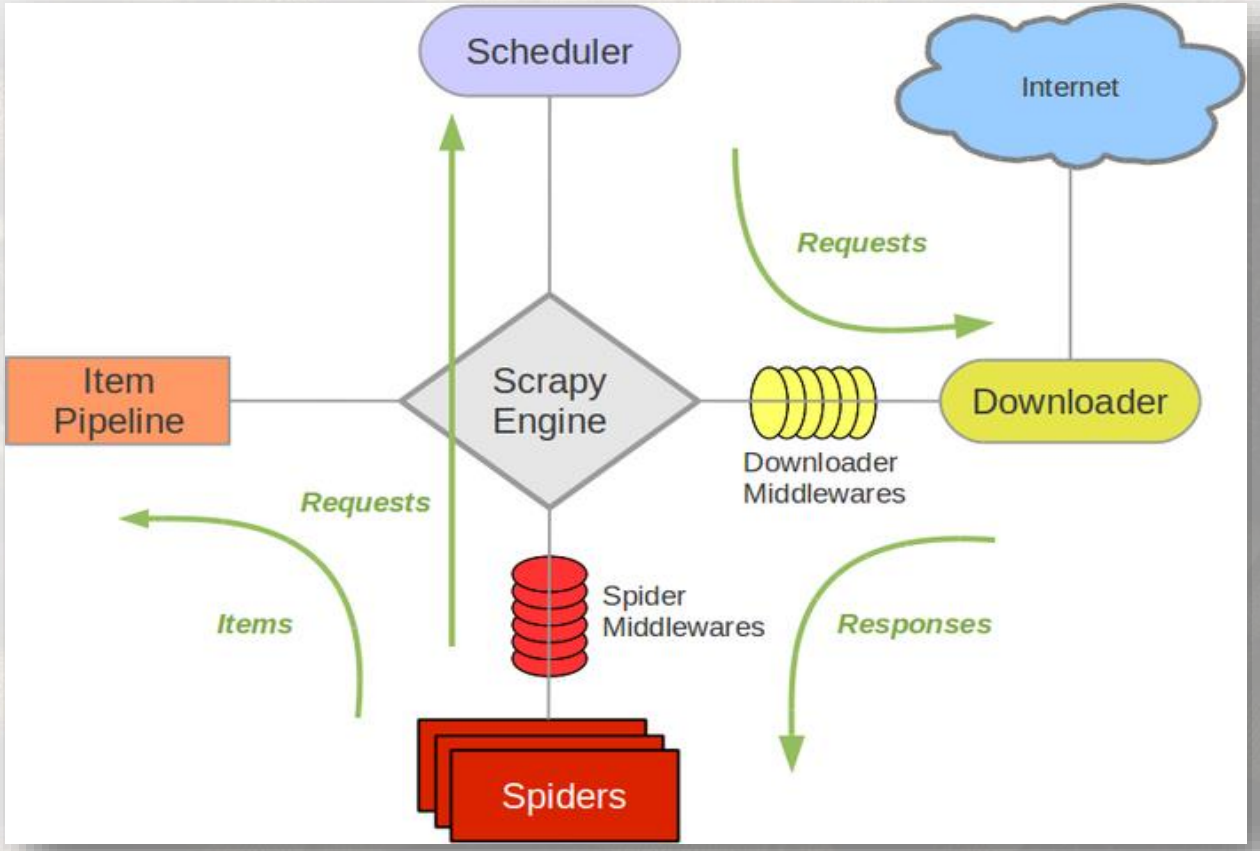
- In order to pull data Scrapy has been used in our project.

- **What is *Scrapy*?**

Scrapy is an application framework for crawling

SCRAPY : DIGGING DEEP

- ***Components*** of Scrapy –
- Scrapy Engine
- Scheduler
- Downloader
- Spiders
- Item Pipeline
- Downloader middlewares



INSIDE SCRAPY: *SPIDERS* ARE NOT ALWAYS NASTY!

- ***Scrapy engine*** is responsible for controlling the data flow between all components of the system
- The ***Scheduler*** receives requests from the engine and enqueues them for feeding them later
- The ***Downloader*** is responsible for fetching web pages and feeding them to the engine which, in turn, feeds them to the spiders.
- ***Spiders*** are custom classes written by Scrapy users to parse responses and extract items (aka scraped items) from them or additional URLs (requests) to follow. Each spider is able to handle a specific domain (or group of domains).
- The ***Item Pipeline*** is responsible for processing the items once they have been extracted (or scraped) by the spiders. Typical tasks include cleansing, validation and persistence (like storing the item in a database).
- ***Downloader middlewares*** are specific hooks that sit between the Engine and the Downloader and process requests when they pass from the Engine to the Downloader, and responses that pass from Downloader to the Engine.

EXAMPLE

- First off, create a project – scrapy startproject tutorial which will create a directory with following contents
- We then create an Item (containers that will be loaded with the scraped data). We create a class ***scrapy.item*** and define attributes as ***scrapy.field*** objects. As we want to capture ingredients from **www.vahrehvahchef.com** , we define field for this attribute. To do this we edit items.py file in the tutorial directory.
- Spiders are classes that you define and Scrapy uses to scrape information from a domain/website. To create a spider you must create subclass ***basespider*** and define some attributes.

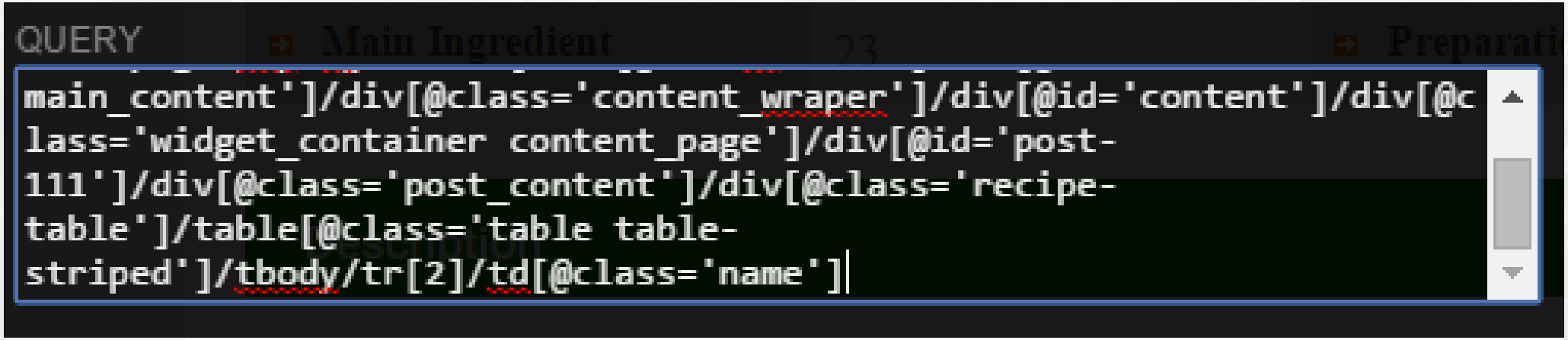
➢ Name – Identifies the spider and must be unique.

➢ start_urls – a list of URLs where the Spider will begin to crawl from.

➢ Parse() - a method of the spider, which will be called with the downloaded response object of each start URL. The response is passed to the method as the first and only argument. This method is in charge of processing the response and returning scraped data .

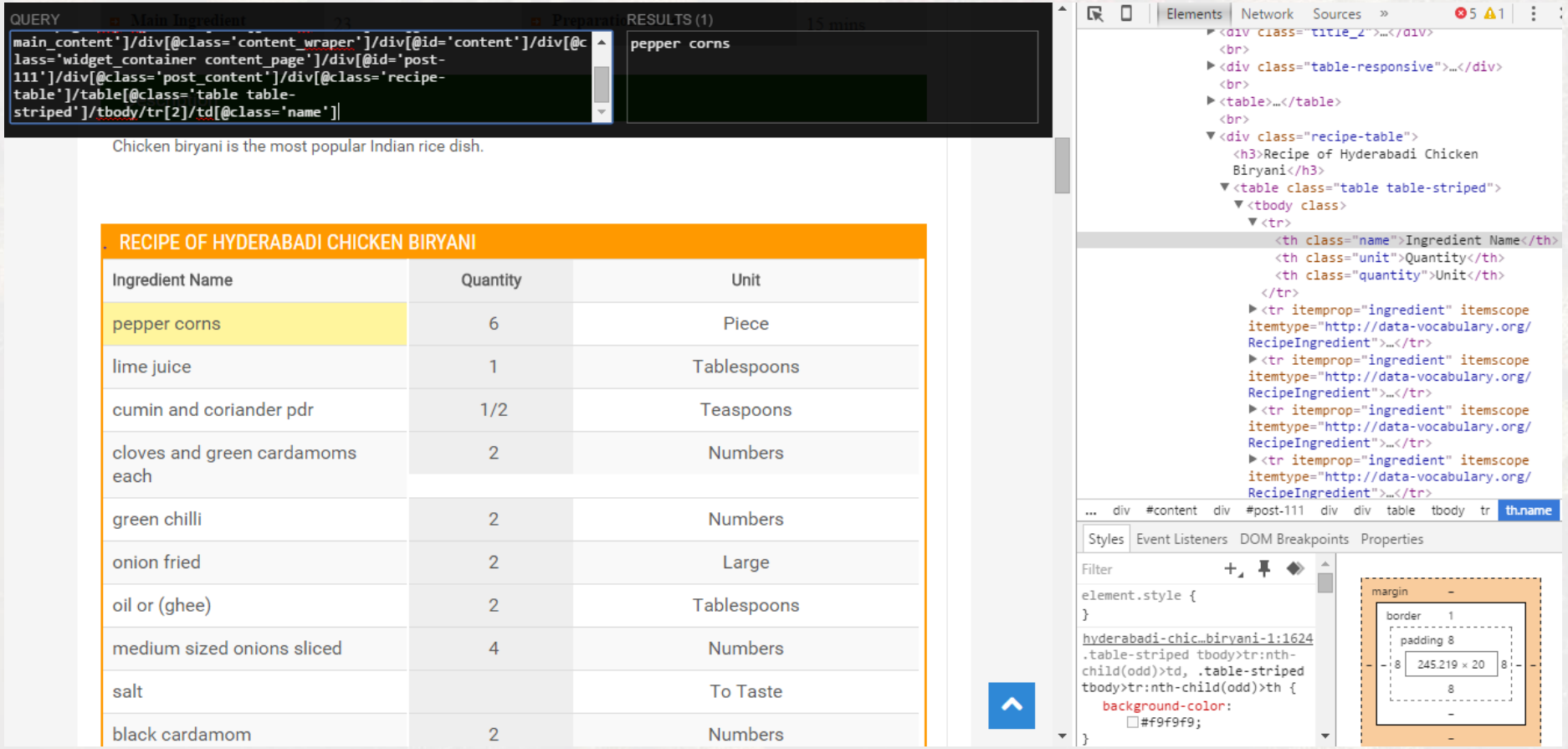
EXAMPLE

- A method parse_fixtures is written which actually does the scraping of the data from the site and then saves these values in item fields under item['ingredients']. In order to fetch the data we use xpath selectors. ***Xpath*** is a language to select nodes in XML and HTML.(similar to what SQL is in DBMS). Then the crawl command is used to crawl and pull the data from the website.



EXAMPLE

- The data is stored in the form of data table under <table> tags in HTML. *item['ingredient']=tr.xpath('td[@class="name"]/text()').extract()* will extract all the values present under the tag td and after extracting will store in the item ['ingredients'].




‘NOW *HANDS* ON!’

DATA PREPARATION

- Snippet from our database of ***1809*** dishes:

	B	
1	States	Ingredients
2	Andhra Pradesh	Milk,vinegar,Sugar,Dessicated coconut,Cardamom powder,Almonds,Cashew nuts,all_purpose_flour,Rice flour,Salt,Oil,W
3	Andhra Pradesh	Ghee,Drumstick leaves,Cumin seeds,Green chillies,Boiled rice,Garlic,Idly podi,Salt
4	Andhra Pradesh	Red gram,Oil,Cloves,Black cumin seeds,Cinnamon,Cardamom,Star anise,Bay leaf,Onion,Mint leaves,Green chillies,Salt,G
5	Andhra Pradesh	Mustard oil,Amla,Asafoetida,Garlic,Chana dal,Dry red chillies,Coriander seeds,Fenugreek powder,Salt
6	Andhra Pradesh	Thummi kura leaves,Salt,Oil,Urad dal,Chana dal,Mustard seeds,Cumin seeds,Curry leaves,Tender tamarind,Green chillies
7	Andhra Pradesh	Ghee,Semolina,Pumpkin,Sugar,Cardamom powder,all_purpose_flour,flour,Salt,Oil,Water
8	Andhra Pradesh	Sugar,Water,all_purpose_flour,Milk,Cashew nuts,Almonds,Ghee,Poppy seeds
9	Andhra Pradesh	Amla,Oil,Garlic,Cumin seeds,Coriander seeds,Curry leaves,Salt,Fenugreek powder
10	Andhra Pradesh	Mutton,Chana dal,Masoor dal,Salt,Turmeric powder,Oil,Mustard seeds,Cumin seeds,Cloves,Cinnamon,Dry red chillies,G
11	Andhra Pradesh	Oil,Eggs,Black cumin seeds,Cinnamon stick,Clove,Cardamom,Biryani phool,Bay leaf,Star anise,Mace,Marathi moggu,Gre
12	Andhra Pradesh	Chana dal,Field beans,French beans,Green chillies,Cumin seeds,Curry leaves,Salt,Onions,Pressed rice,Rice flour,Oil
13	Andhra Pradesh	Chickpea flour,Rice flour,Oil,Green chillies,Salt,Curry leaves,Coriander leaves,Onions
14	Andhra Pradesh	Corn,Moong dal,Ginger,Green chillies,Salt,Potato,Oil,Yogurt,Water,Black salt,Red chilly powder,Cumin powder,Fennel p
15	Andhra Pradesh	Moong dal,Flax seeds,Cardamom powder,Jaggery,Ghee

DATA PREPARATION

DATA PREPARATION

- The primal problem in the scraped data was ***inconsistency***
- Use of ***Hindi versions*** of different ingredients made the data inconsistent
- ***Spelling mistakes*** was a hindrance for correct feature extraction
- Use of plural form of proper nouns, similar meaning words, adjectives (***different POS***) were additional noise in the data
- To ***deal*** with the problem, we ***replaced*** all ***Hindi*** words by their ***English*** counter parts
- Spelling mistakes have been corrected
- Porter's ***stemming*** algorithm gave data with uniform consistent set of words
- ***Removing sparse terms*** after creating DTM removed most of the adjectives those are less frequent. Otherwise, all other adjectives were deliberately removed from the data.

FEATURE ENGINEERING

- **Number of ingredients** – We considered number of ingredients as an important feature as it varies from 2 to 29, similarly as we considered for Kaggle data

- **Number of spices per dish** – We have created an ***Indian Spice dictionary*** which contains all possible (commonly used) spices. We created a ***score*** i.e number of spices per dish. From our subjective knowledge, it seems to be an importance feature as there are regions in India where use of spices is too high.

APPLICATION OF XGBOOST

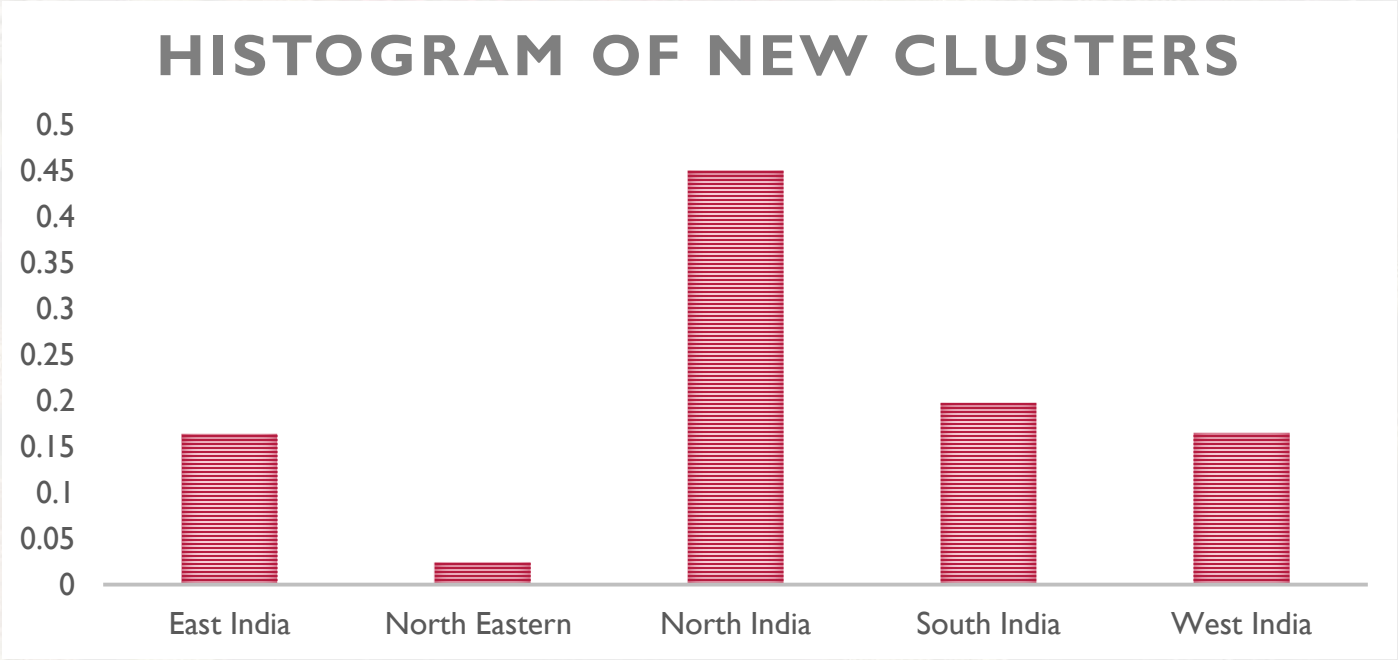
- Application of ***XGBoost softmax*** algorithm yielded accuracy of ***18.99%*** for Indian database.

'An epic fail!'

Possible reason: ***Data doesn't even have 26 distinct clusters!***

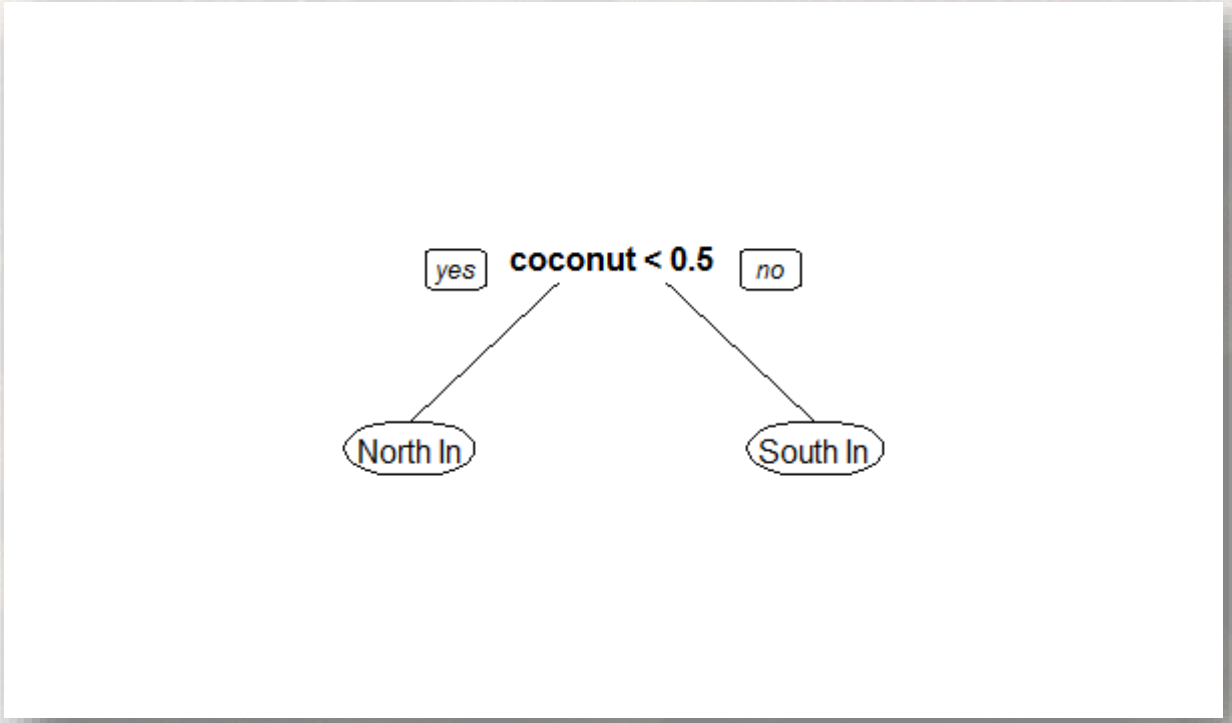
SEARCH FOR NEW SET OF CLUSTERS

- Based on our subjective knowledge, we redefined the label for each dish. Our assumption was dividing all the dishes based on their geographic origin
- Five main clusters have been chosen :
- 1. North India
- 2. South India
- 3. West India
- 4. East India
- 5. North Eastern



RESULT FOR NEW SET OF CLUSTERS

- Basic decision tree (CART) couldn't give satisfactory result as the prediction was almost obvious and intuitive.



RESULT FOR NEW SET OF CLUSTERS

- Random Forest almost worked well for Indian data and gave an accuracy of ***47.5%*** for a test data of size ***300*** (16% of the data)
- XGBoost also gave nearly similar accuracy of ***48.1%***
- As it seems, due to the poor data condition the accuracy cannot go beyond 50%
- The interesting output from XGBoost was relative importance of each feature for classifying a dish

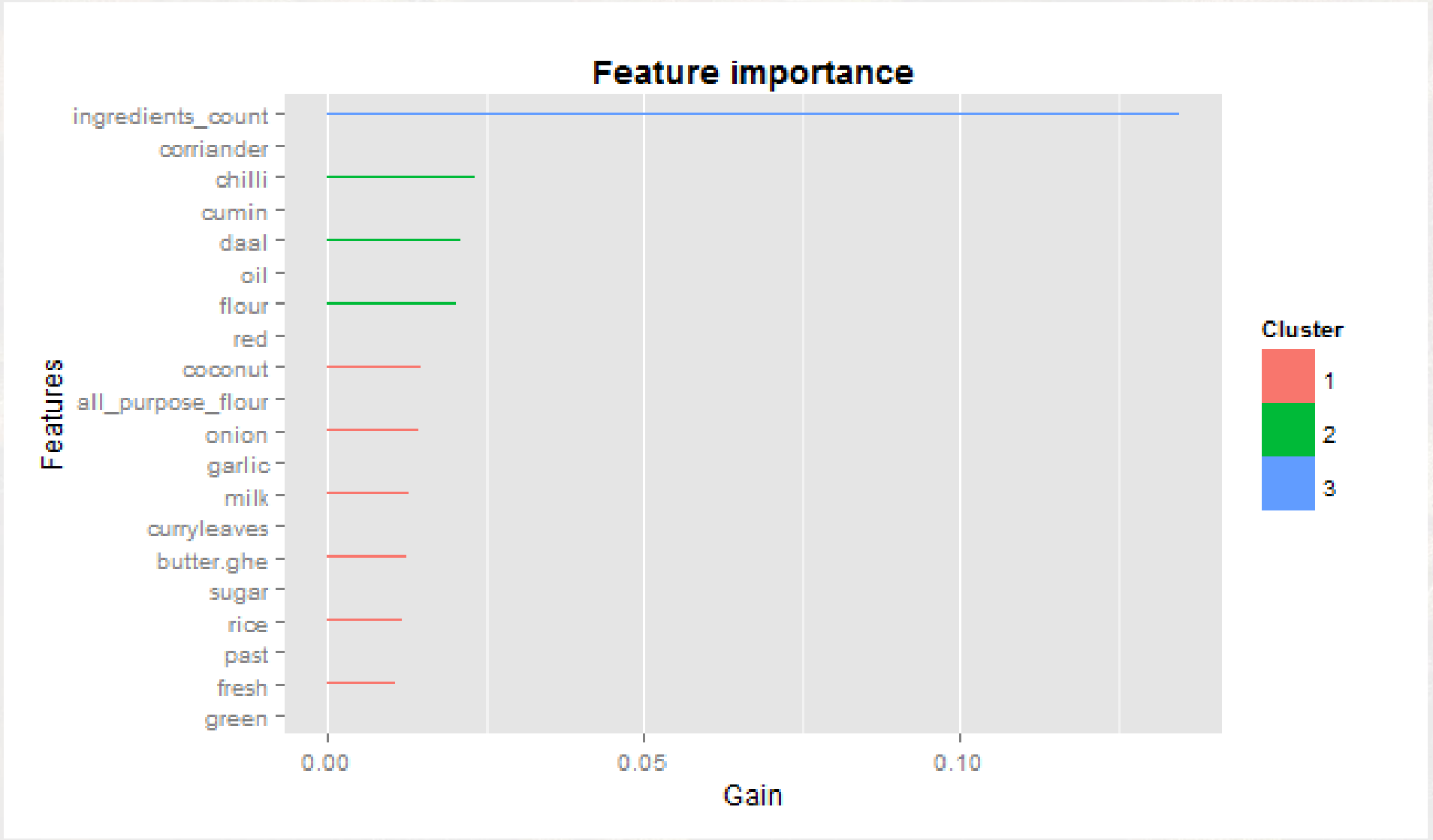
IMPORTANCE MATRIX AND IMPORTANCE PLOT

- XGBoost allows to make an ***importance matrix*** which contains sorted features based on relative importance
- ***Gain*** contribution of each feature of each tree is taken into account, then average gain per feature is calculated for a vision of the entire model
- Highest percentage means ***important feature*** to predict the label used for the training.

IMPORTANCE MATRIX

	A	B	C	D
1	Feature	Gain	Cover	Frequence
2	ingredients_count	0.13455938	0.09509893	0.16357348
3	corriander	0.0261617	0.00811651	0.02843675
4	chilli	0.02341534	0.01455112	0.026826
5	cumin	0.02279992	0.00716818	0.02150818
6	daal	0.02102959	0.013988	0.01499414
7	oil	0.02097818	0.0103166	0.01801428
8	flour	0.02027859	0.01295254	0.02174505
9	red	0.01539915	0.00716912	0.01300439
10	coconut	0.01498239	0.00808402	0.00996056
11	all_purpose_flour	0.0146167	0.01502093	0.02136605
12	onion	0.01443858	0.00482059	0.01144102
13	garlic	0.01391403	0.00351766	0.00993687

IMPORTANCE PLOT



MULTICLASS SOFTPROB ~ SHOWING PROBABILITIES

	B	C	D	E	F	G
1	South Indian	East Indian	North Indian	North Eastern	Final Prediction	Cuisine
2	4.40E-06	0.00284953	0.99634618	0.000132038	0.996346176	North Indian
3	1.92E-05	0.97539616	0.00104444	0.018707385	0.975396156	East Indian
4	6.38E-05	0.1460567	0.85144532	0.001472618	0.851445317	North Indian
5	0.00028066	0.60097462	0.00067893	0.20004487	0.600974619	East Indian
6	0.50179303	0.00647875	0.00044439	0.491053164	0.501793027	South Indian
7	3.05E-05	0.50175124	0.24942328	0.248498023	0.501751244	East Indian

CHALLENGES FACED

- Proper ***structured data*** was not available
- Individual ***knowledge biased data*** when labeling became crucial for classification
- ***Inconsistency*** was a big threat, repetitions of word in ingredient list made our task really difficult
- A proper ***Indian Spice Dictionary*** wasn't available to map total number of features
- Our subjective clustering might not be correct, presence of ***local effect*** on dishes ignored
- Unsupervised learning could have been a way to detect original clusters present in data, but ***an appropriate distance measure*** between two dishes was unavailable
- It would be good if DTM could be made ***weighted*** but proper measures (quantity, or some other measure which emphasize importance) to calculate respective weight of ingredients in a dish were absent to classify for cuisine

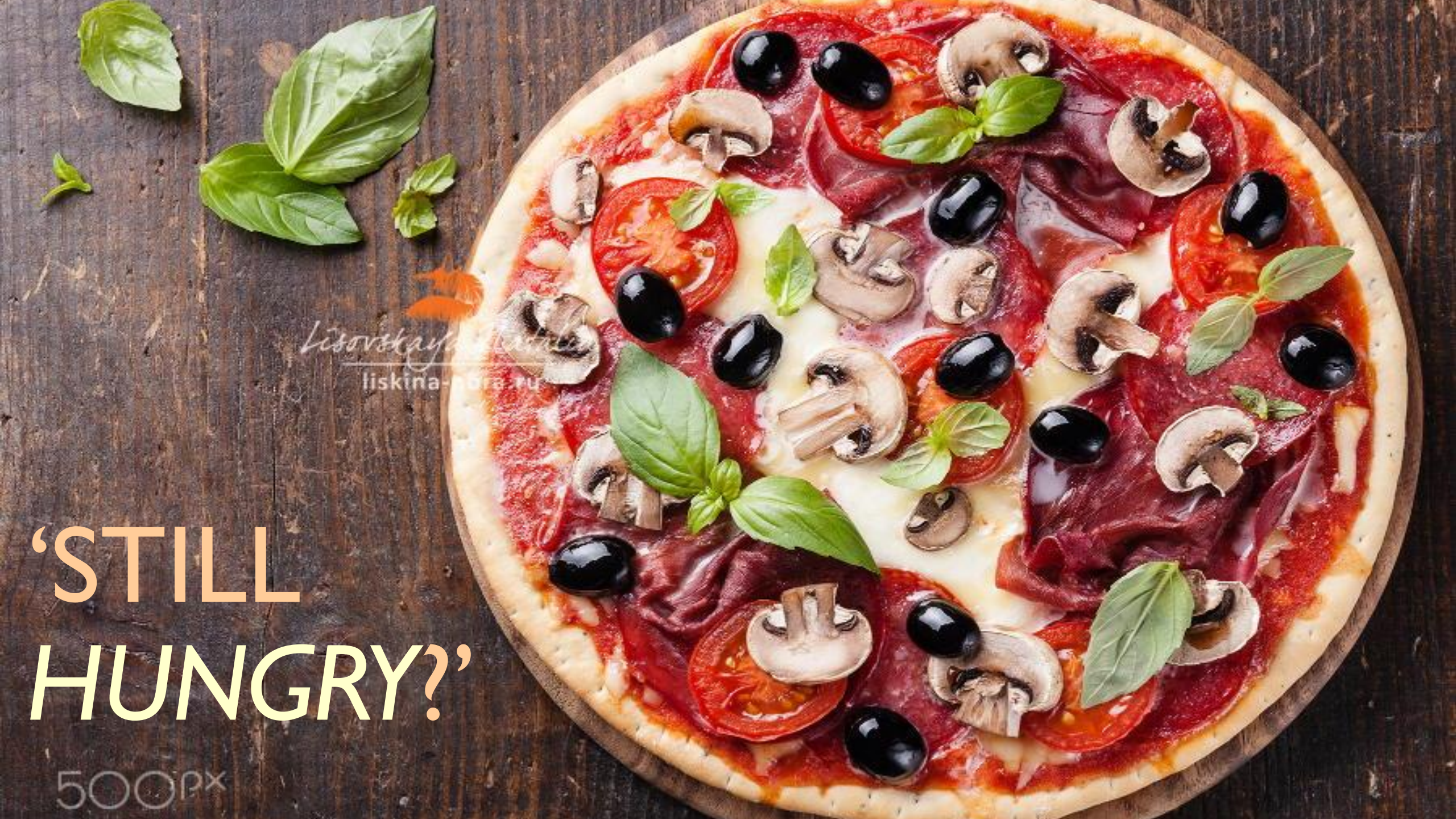
FUTURE SCOPE

- A ***proper distance measure*** can be studied to measure similarity or distance between two dishes
- An unsupervised learning or ***clustering*** can be done to see number of clusters present in data
- ***Two-level clustering*** could be effective
- ***DTM*** can be ***weighted*** based on importance of the ingredients in a dish
- Classification can be more deep leading specific ***dish prediction***
- Can be used for ***recommendation system*** in big chain retailers
- ***Indian Database*** could be made more properly to have an organized data

ACKNOWLEDGEMENT AND REFERENCES

- https://cran.r-project.org/web/packages/xgboost/xgboost.pdf
- http://xgboost.readthedocs.org/en/latest/model.html
- https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf
- https://www.kaggle.com/c/whats-cooking/forums
- http://www.isical.ac.in/~debapriyo/teaching/datamining2014/slides/NaiveBayes.pdf

We acknowledge our instructor ***Sourav Sen Gupta*** for his continuous support and valuable inputs.



‘STILL HUNGRY?’