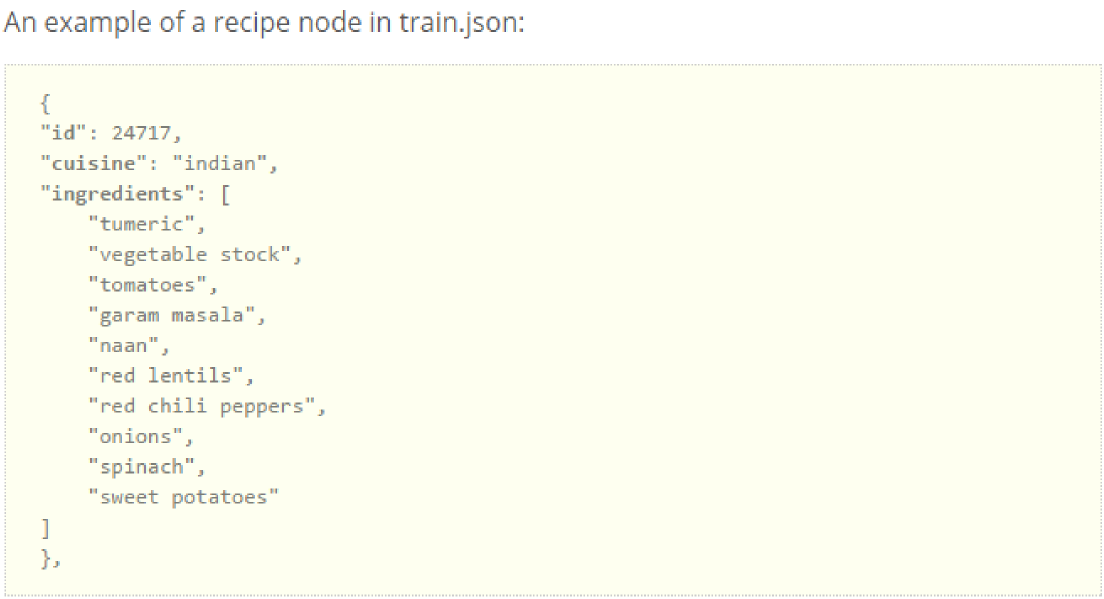
# Solving Machine Learning Problem using CRIPS

Feb 26. 2016
Soojung Hong

# Problem Description : *What’s Cooking*

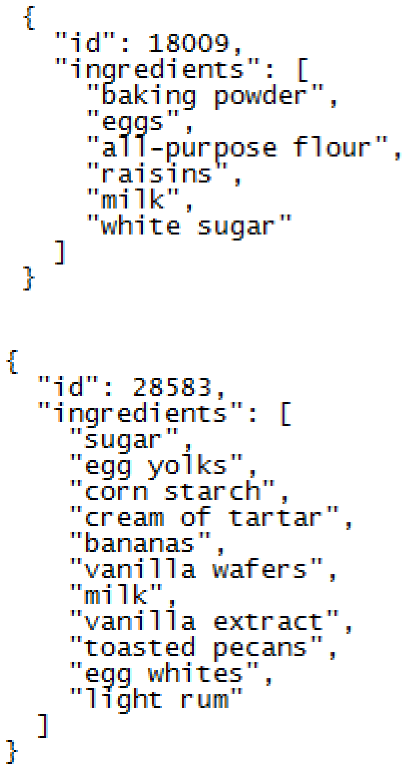
- https://www.kaggle.com/c/whats-cooking/data



An example of a recipe node in train.json:

```
{
"id": 24717,
"cuisine": "indian",
"ingredients": [
    "tumeric",
    "vegetable stock",
    "tomatoes",
    "garam masala",
    "naan",
    "red lentils",
    "red chili peppers",
    "onions",
    "spinach",
    "sweet potatoes"
]
},
```

Training Data



```
 {
   "id": 18009,
   "ingredients": [
     "baking powder",
     "eggs",
     "all-purpose flour",
     "raisins",
     "milk",
     "white sugar"
   ]
 }


{
  "id": 28583,
  "ingredients": [
    "sugar",
    "egg yolks",
    "corn starch",
    "cream of tartar",
    "bananas",
    "vanilla wafers",
    "milk",
    "vanilla extract",
    "toasted pecans",
    "egg whites",
    "light rum"
  ]
}
```

Test Data

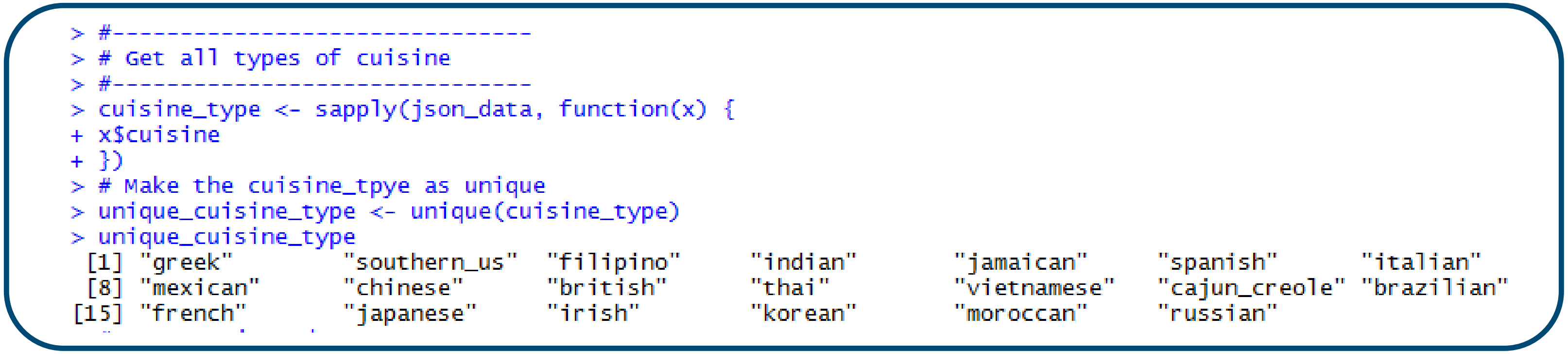
# CRISP :
# Cross Industry Standard Process for Data Mining

- 1. Business Understanding
- **2. Data Understanding**
- 3. Data Preparation
- **4. Modeling**
- **5. Evaluation**
- 6. Deployment

# Data Understanding

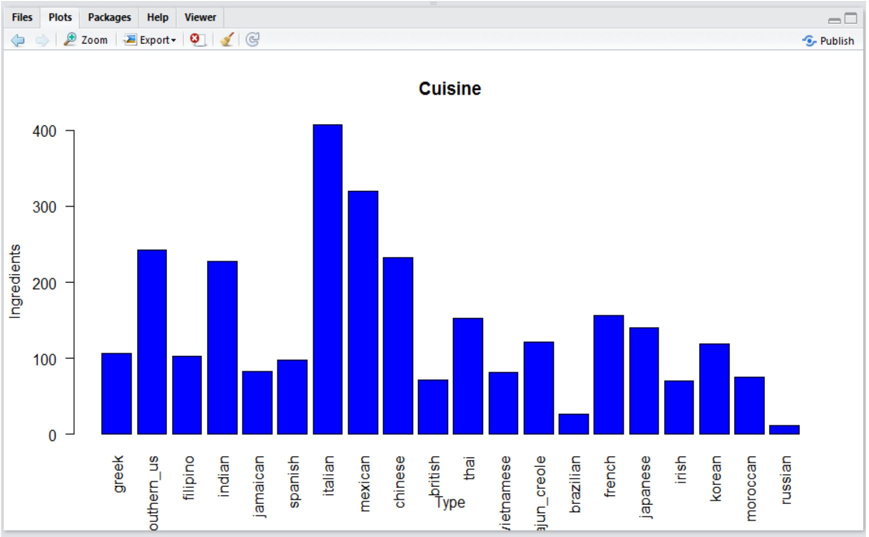
- **Given Data**
  JSON file with 39774 objects (id, cuisine, ingredients)

- **Training Data Analysis**
  20 different types of cuisine
  428271 ingredients (unique ingredients : 6714)
  3 most frequents cuisines : Italian, Mexican and Chinese
  Avg. 10 ingredients per one recipe

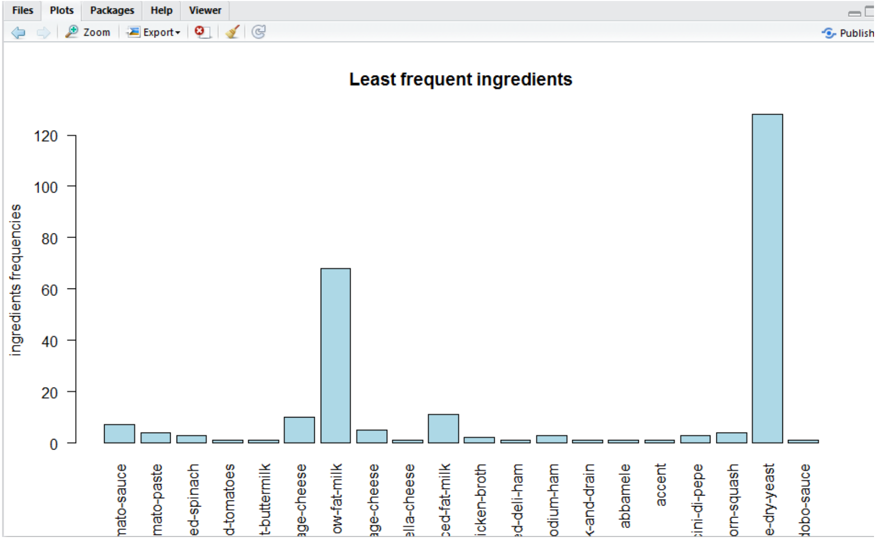


```
> #-------------------------------
> # Get all types of cuisine
> #-------------------------------
> cuisine_type <- sapply(json_data, function(x) {
+ x$cuisine
+ })
> # Make the cuisine_tpye as unique
> unique_cuisine_type <- unique(cuisine_type)
> unique_cuisine_type
 [1] "greek"        "southern_us"  "filipino"     "indian"       "jamaican"     "spanish"      "italian"
 [8] "mexican"      "chinese"      "british"      "thai"         "vietnamese"   "cajun_creole" "brazilian"
[15] "french"       "japanese"     "irish"        "korean"       "moroccan"     "russian"
```

# Training Data Analysis



Cuisine Type

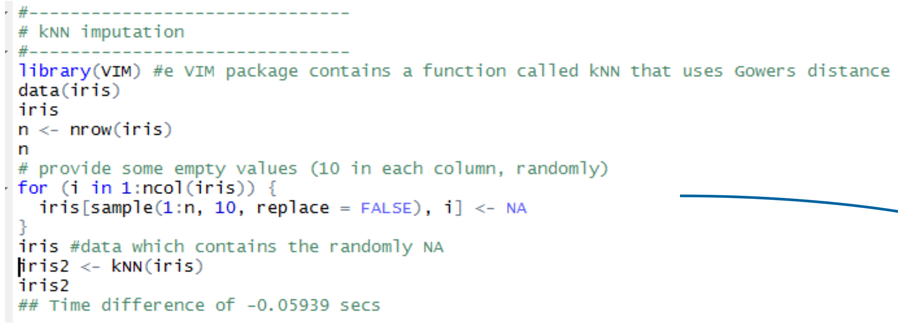


Least 20 Frequent Ingredients in ‘Italian’

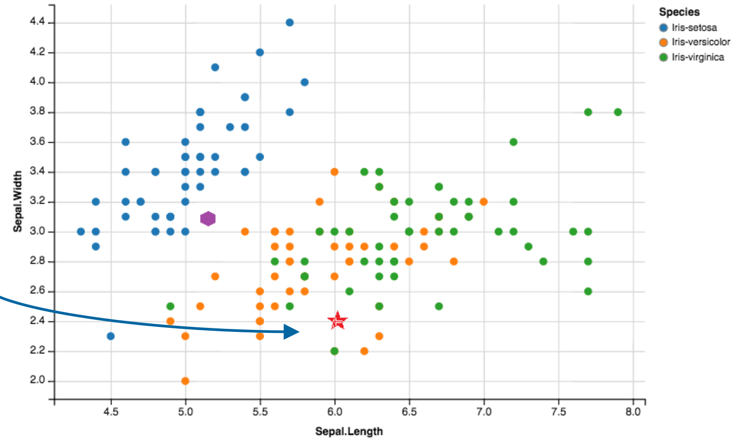
# Modeling using kNN, Similarity Measure

- K Nearest Neighbor Algorithm
  - Use training data as classifier
  - Given data point x, find k number of training data point close to x
  - Assign x the label of closest point

- Characteristics of K Nearest Neighbor
  - Find k closest training points
  - Take a majority vote between k points
  - Rule of Thumb, 3NN often works surprisingly well
  - (disadvantage : If the training set is big, then it is slow)



```
#-------------------------------
# kNN imputation
#-------------------------------
library(VIM) #e VIM package contains a function called kNN that uses Gowers distance
data(iris)
iris
n <- nrow(iris)
n
# provide some empty values (10 in each column, randomly)
for (i in 1:ncol(iris)) {
  iris[sample(1:n, 10, replace = FALSE), i] <- NA
}
iris #data which contains the randomly NA
iris2 <- kNN(iris)
iris2
## Time difference of -0.05939 secs
```

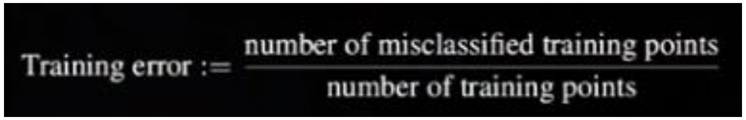


# First Approach using kNN

- Training Set Size (i.e. Classifier) : 19887 (= A half of training data)
- Test data size : 100/200/300

| | Training Group A | Training Group B | Training Group C |
|---|---|---|---|
| Training Data | 19887 | 19887 | 19887 |
| Test Data | 100 | 200 | 300 |
| Correct Classification | 62 | 118 | 171 |

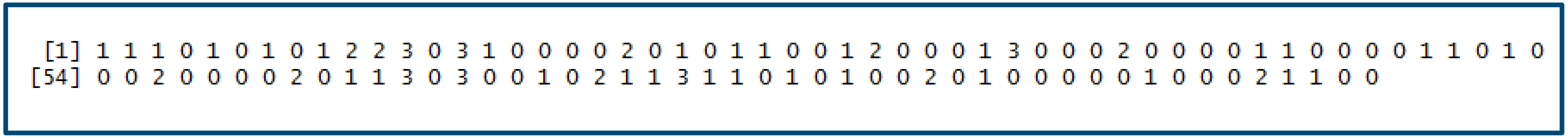
Training Error : 0.40



$$\text{Training error} := \frac{\text{number of misclassified training points}}{\text{number of training points}}$$

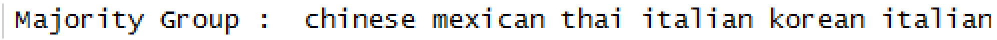
# Improvement (1) :
# Among elements with max. similiary value, vote

Similarity measure vector



```
 [1] 1 1 1 0 1 0 1 0 1 2 2 3 0 3 1 0 0 0 0 2 0 1 0 1 1 0 0 1 2 0 0 0 1 3 0 0 0 2 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0
[54] 0 0 2 0 0 0 0 2 0 1 1 3 0 3 0 0 1 0 2 1 1 3 1 1 0 1 0 1 0 0 2 0 1 0 0 0 0 0 1 0 0 0 2 1 1 0 0
```

Maximum matching number : 3
6 training samples match 3 ingredients



```
Majority Group :  chinese mexican thai italian korean italian
```

Vote

**Italian**

# Improvement (1) using KNN

- Training Set Size (i.e. Classifier) : 19887
- Test data size : 100 / 200 / 300

| | Training Group A | Training Group B | Training Group C |
|---|---|---|---|
| Training Data | 19887 | 19887 | 19887 |
| Test Data | 100 | 200 | 300 |
| Correct Classification | 64 | 122 | 184 |

Training Error : 0.38
(Previous Training Error : 0.40)

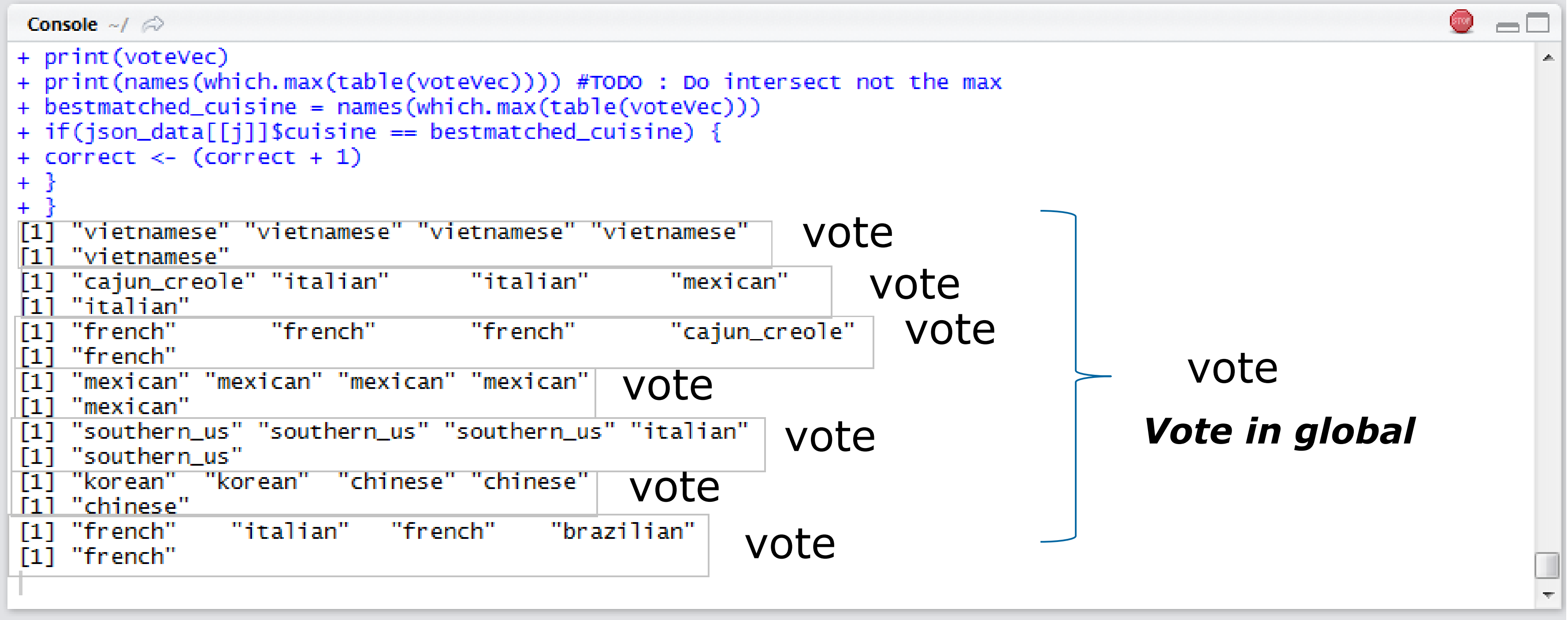
# Improvement (2) :
# Increase the size of Neighbors

- Training Set Size (i.e. Classifier) : 19887
- Test data size : 100 / 200 / 300

| | Training Group A | Training Group B | Training Group C |
|---|---|---|---|
| Training Data | 35000 | 35000 | 35000 |
| Test Data | 100 | 200 | 300 |
| Correct Classification | 72 | 141 | 211 |

Training Error : 0.29
(Previous Training Error : 0.38)

# Approach 2 : Partitioning training set
# Run kNN several times and make a vote



```
+ print(voteVec)
+ print(names(which.max(table(voteVec)))) #TODO : Do intersect not the max
+ bestmatched_cuisine = names(which.max(table(voteVec)))
+ if(json_data[[j]]$cuisine == bestmatched_cuisine) {
+ correct <- (correct + 1)
+ }
+ }
[1] "vietnamese" "vietnamese" "vietnamese" "vietnamese"
[1] "vietnamese"
[1] "cajun_creole" "italian"      "italian"      "mexican"
[1] "italian"
[1] "french"       "french"       "french"       "cajun_creole"
[1] "french"
[1] "mexican" "mexican" "mexican" "mexican"
[1] "mexican"
[1] "southern_us" "southern_us" "southern_us" "italian"
[1] "southern_us"
[1] "korean"  "korean"  "chinese" "chinese"
[1] "chinese"
[1] "french"    "italian"   "french"    "brazilian"
[1] "french"
```

***Vote in local Partition***

# Result of partitioned KNN approach

| | Training Group A | | | | Training Group B | | | | Group C |
|---|---|---|---|---|---|---|---|---|---|
| | Partition 1 | Partition 2 | Partition 3 | Partition 4 | Partition 1 | Partition 2 | Partition 3 | Partition 4 | - same - |
| Training Set | #1 ~ # 10000 | #10001 ~ #20000 | #20001 ~ #30000 | #30001 ~#39000 | #1 ~ # 10000 | #10001 ~ #20000 | #20001 ~ #30000 | #30001 ~#39000 | - same - |
| Test | 100 | | | | 200 | | | | 300 |
| Correct classification | 66 | | | | 131 | | | | 197 |

Training Error : 0.32

# Further Improvement

- For Multiple Classification : KNN works well
  However, when the training data is big, slow

- Further Improvement :
  - Optimize the R code to overcome the kNN performance
  - Apply another technique in voting e.g. geographical context
  - Applying Association Rule Learning