Math 381: Assignment 7: MDS

Ella Kim

December 3, 2021

1 Introduction

A study recorded the percentage of all households with various foods in European countries. Below is the table/matrix of the countries with the corresponding percentages for the foods (written in a code in the first column):

Code	West Germany	Italy	France	Netherlands	Belgium	Luxemburg	Great Britain	Portugal	Austria	Switzerland	Sweden	Denmark	Norway	Finland	Spain	Ireland
GC	90	82	88	96	94	97	27	72	55	73	97	96	92	98	70	13
IC	49	10	42	62	38	61	86	26	31	72	13	17	17	12	40	52
TB	88	60	63	98	48	86	99	77	61	85	93	92	83	84	40	99
SS	19	2	4	32	11	28	22	2	15	25	31	35	13	20	0	11
BP	57	55	76	62	74	79	91	22	29	31	0	66	62	64	62	80
SP	51	41	53	67	37	73	55	34	33	69	43	32	51	27	43	75
ST	19	3	11	43	25	12	76	1	1	10	43	17	4	10	2	18
IP	21	2	23	7	9	7	17	5	5	17	39	11	17	8	14	2
FF	27	4	11	14	13	26	20	20	15	19	54	51	30	18	23	5
VF	21	2	5	14	12	23	24	3	11	15	45	42	15	12	7	3
AF	81	67	87	83	76	85	76	22	49	79	56	81	61	50	59	57
OF	75	71	84	89	76	94	68	51	42	70	78	72	72	57	77	52
FT	44	9	40	61	42	83	89	8	14	46	53	50	34	22	30	46
JS	71	46	45	81	57	20	91	16	41	61	75	64	51	37	38	89
CG	22	80	88	15	29	91	11	89	51	64	9	11	11	15	86	5
BR	91	66	94	31	84	94	95	65	51	82	68	92	63	96	44	97
ME	85	24	47	97	80	94	94	78	72	48	32	91	94	94	51	25
OO	74	94	36	13	83	84	57	92	28	61	48	30	28	17	91	31
YT	30	5	57	53	20	31	11	6	13	48	2	11	2	0	16	3
CD	26	18	3	15	5	24	28	9	11	30	93	34	62	64	13	9

The food codes correspond to these names:

- GC: Ground coffee
- IC: Instant coffee
- TB: Tea bags
- SS: Sugarless sweets
- BP: Packaged biscuits
- SP: Packaged soup
- ST: Tinned soup

- IP: Instant potatoes
- FF: Frozen fish
- VF: Frozen vegetables
- AF: Fresh apples
- OF: Fresh oranges
- FT: Tinned fruit
- JS: Shop jam
- CG: Garlic clove
- BR: Butter
- ME: Margarine
- OO: Olive oil
- YT: Yogurt
- CD: Crispbread

As seen in the table above, there is a set of various food, which each have a vector of the different European countries and the corresponding percentages of households in that country that have the said food type.

I define the distance between each pair of values as the euclidean norm of the difference of these vectors. Once I compute these distances between each pair, I can apply MDS (Multidimensional Scaling) to create a low-dimensional model (using R and the cmdscale function/command) for the set of food types.

2 Defining the Distances & Creating Models

Before I compute the distances, I want to try normalizing/standardizing the columns of my table. The types of values in the whole matrix are all percentages however, so this step is not really necessary (especially since some of them assume normal distribution of the data), but I will do it anyway just to see the results.

As my analysis and application of MDS is in R, I now show the R code I use to create the matrix-form of the values for my computations:

```
library(BBmisc)
library(wordcloud)



# read in csv of food data
mydata <- read.csv("file45.csv",header=FALSE)

a# make subset of the data above without the name of food
mydata_no_name <- subset(mydata,select=-V1)

# make data frame into matrix
mydata_matrix <- data.matrix(mydata_no_name)
```

* * *

Besides looking at the original values, I will also normalize the values by standardizing (subtracting each value by the mean of the column and divide by the standard deviation), and normalizing with a [0,1] range (subtracting the minimum and dividing by the difference between the maximum and minimum in the columns). As the percentages are in a uniform 0-100 scale, I would like to see how this normalizing would change if some sets do not include a wide range of percentages.

R Code for the Various Normalizing/Standardizing Methods:

```
# normalize values (standardize, and give set range within columns)
nonnormalize <- mydata_matrix
normalize_one <- normalize(mydata_matrix,method="standardize",margin=2)
normalize_two <- normalize(mydata_matrix,method="range", margin=2)
```

* * *

As I do not think that specific ranges of percentages should have higher/lower importance, I do not add a function to adjust the values before computing distances. I define my distance method using "minkowski" (the distance between row i and row j in column m):

$$d_{ij} = (\sum_{m} |r_{i,m} - r_{j,m}|^p)^{1/p}$$

Where p = 1.8 (not p = 2, the classic Euclidean distance, explained later in the "goodness" section). This one-dimensional distances is then written in matrix form for visual representation and for calculating the eigenvalues.

R Code for Distances and Matrix Form for Distances:

```
# get distances between each
distances = dist(nonnormalize, method = "minkowski", p=2)
distances_stand = dist(normalize_one, method = "minkowski", p=2)
distances_range = dist(normalize_two, method = "minkowski", p=2)

# make these distances into matrix form
distances_mat <- as.matrix(distances)
distances_mat_stand <- as.matrix(distances_stand)
distances_mat_range <- as.matrix(distances_range)
```

* * *

At this moment, to understand the distance matrix, I want to find out which types of food have the largest differences in percentages. If the distance matrices are correct, then the food types with the greatest and least difference would be the largest and smallest values in the whole matrix. I used R code to find the column and row values, as well as the distance value (in the non-standardized/normalized matrix).

R Code for Finding Maximum and Minimum Distances Between Food Types:

```
# get max distance
max <- 0
indexmax <- 0
rownummax <- 0
for (i in 1:20) {
  index <- which.max(distances_mat[,i])
  val <- max(distances_mat[,i])
  if (val > max) {
    max <- val
    indexmax <- index
    rownummax <- i
  }
}

# get min distance, have to ignore 0 in diag so add super large value
temp <- diag(999999, 20, 20)
distance_for_min <- distances_mat + temp

min <- 999999
indexmin <- 0
rownummin <- 0
for (i in 1:20) {
  index <- which.min(distance_for_min[,i])
  val <- min(distance_for_min[,i])
  if (val < min && val != 0) {
    min <- val
```

```
    indexmin <- index
    rownummin <- i
  }
}
```

* * *

The code results in the maximum distances between Instant Potatoes and Ground Coffee and the minimum distances between Sugarless Sweets and Frozen Vegetables. This was also the same for the other two standardized/normalized distances. Below are tables of the said rows to confirm that the distances were correct.

Max Distance (distance was approx. 276.456 in the original distance matrix)

Code	WG	IT	FR	NS	BM	LG	GB	PL	AA	SD	SW	DK	NY	FD	SP	ID
IP	21	2	23	7	9	7	17	5	5	17	39	11	17	8	14	2
GC	90	82	88	96	94	97	27	72	55	73	97	96	92	98	70	13

Min Distance (distance was approx. 30.033 in the original distance matrix)

Code	WG	IT	FR	NS	BM	LG	GB	PL	AA	SD	SW	DK	NY	FD	SP	ID
SS	19	2	4	32	11	28	22	2	15	25	31	35	13	20	0	11
VF	21	2	5	14	12	23	24	3	11	15	45	42	15	12	7	3

After confirming these distance matrices, I will now use the cmdscale function in R on these distances to get the MDS in the 1st, 2nd, and 3rd-dimensional, euclidean models. Below is the code for these models, as well as the code for the plots of the 1st and 2nd-dimensional models.

R Code for Models and Plots:

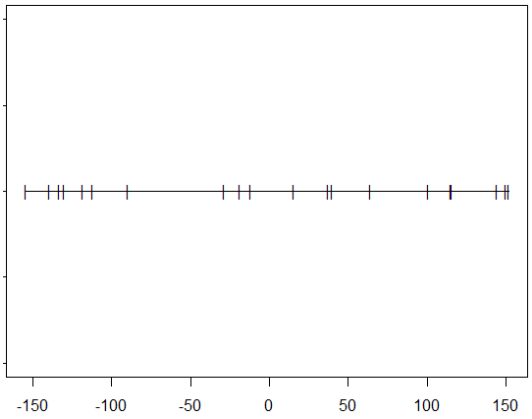
```
# get models (MDS) for 1-3rd dimension
model1 <- cmdscale(distances_mat, k=1) # k = dimension
model2 <- cmdscale(distances_mat, k=2)
model2_1 <- cmdscale(distances_mat, k=3)
model2_2 <- cmdscale(distances_mat, k=4)

model3 <- cmdscale(distances_mat_stand, k=1) # k = dimension
model4 <- cmdscale(distances_mat_stand, k=2)
model4_1 <- cmdscale(distances_mat_stand, k=3)
model4_2 <- cmdscale(distances_mat_stand, k=4)

model5 <- cmdscale(distances_mat_range, k=1) # k = dimension
model6 <- cmdscale(distances_mat_range, k=2)
model6_1 <- cmdscale(distances_mat_range, k=3)
model6_2 <- cmdscale(distances_mat_range, k=4)
```

```
# plot 1st dimensions
x <- data.frame(model1,1)
plot(x, type = 'o', pch = '|', ylab = '')

x1 <- data.frame(model3,1)
plot(x1, type = 'o', pch = '|', ylab = '')

x2 <- data.frame(model5,1)
plot(x2, type = 'o', pch = '|', ylab = '')


# plot 2nd dimensions
plot(model2, asp = 1,
     ylim = c(min(model2[,2]), max(model2[,1])),
     xlim = c(min(model2[,2]), max(model2[,1])),
     ylab="", xlab="")

plot(model4, asp = 1,
     ylim = c(min(model4[,2]), max(model4[,1])),
     xlim = c(min(model4[,2]), max(model4[,1])),
     ylab="", xlab="")

plot(model6, asp = 1,
     ylim = c(min(model6[,1]), max(model6[,1])),
     xlim =  c(min(model6[,1]), max(model6[,1])),
     ylab="", xlab="")
```

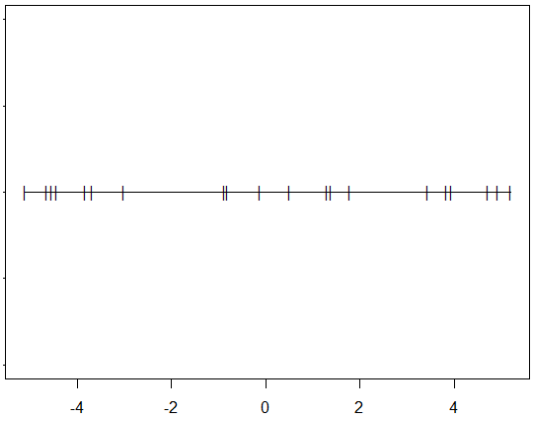
* * *

Below are the resulting plots for the code above:

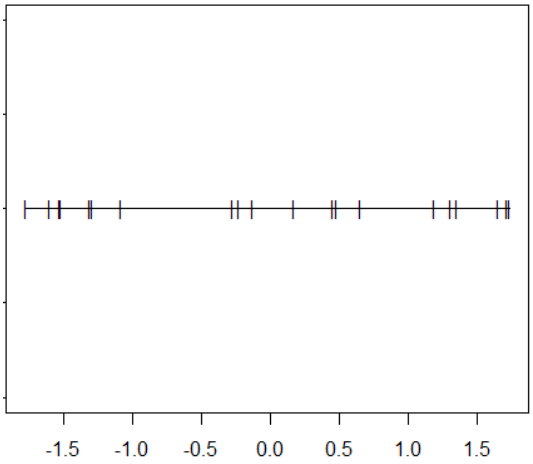
Original Distance 1-dimensional Model:



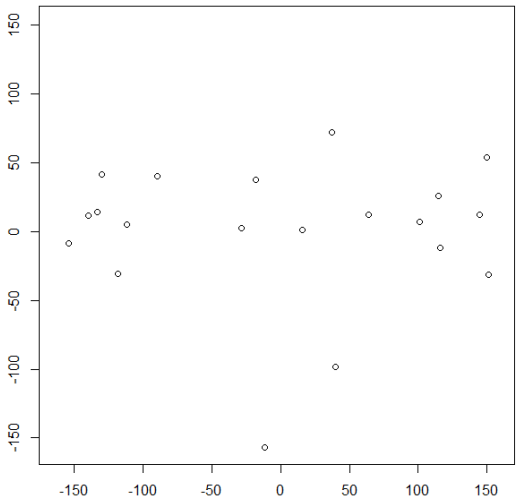
Standardized Distance 1-dimensional Model:



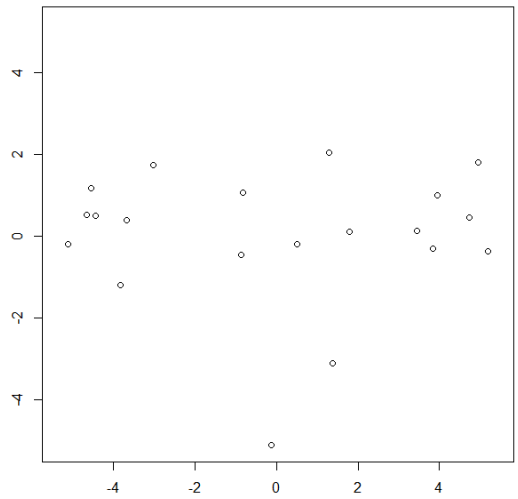
Ranged Distance 1-dimensional Model:



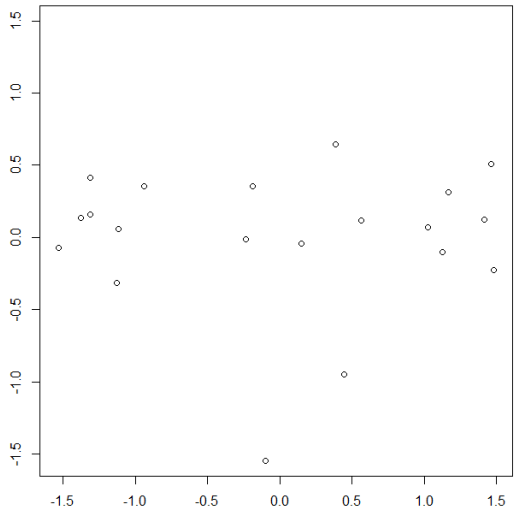
Original Distance 2-dimensional Model:



Standardized Distance 2-dimensional Model:



Ranged Distance 2-dimensional Model:

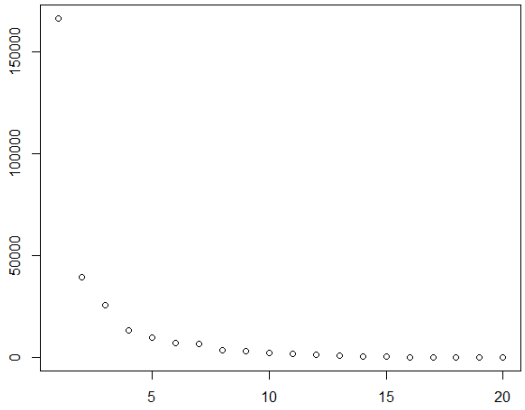


3 "Goodness" of the Various Methods

Before analyzing the patterns in the model plots, I first will investigate in several ways whether these models are significant (i.e, if these models are good). Below is the R code for computing and plotting the eigenvalues of the distance matrices (they do not change with the various models, so the one below is of the original distances):

```
# calculate eigenvalues
eigvalues <- cmdscale(distances_mat, k=2, eig = TRUE)$eig

plot(c(1:20), eigvalues)

data.frame(cmdscale(distances_mat, k=2, eig = TRUE)$eig)
```

Plot of Eigenvalues:



As seen in the plot, the first 16 eigenvalues are non-zero, which corresponds that our data fits with the 16D space. Although the 20th eigenvalue is negative in the output, this is most likely a rounding error. One way to prove this is the GOF values, and two GOF are equal. When the eigenvalue is negative, there are two different ways to computing eigenvalues (first using the absolute value of the eigenvalue, second using the max(eigenvalue, 0) (i.e., use zero when it is negative), which will each result in slightly different values for the GOF as the GOF uses the two different eigenvalues to compute the two different GOFs. The GOFs being equal proves that all the eigenvalues are positive, and that these distances are euclidean.

Back to the eigenvalues however, there is a huge jump from the first eigenvalue to the second one, which I initially believed would mean that a 1D model would be sufficient. However, when studying each eigenvalue, the 2nd eigenvalue's magnitude is significantly large, and would have a great affect on the "goodness" of the model, meaning that a low-dimensional model will not be perfect,

I would argue that a 2D model would not be too bad (will argue with more "goodness" methods below) of a model, a 3D model would be better, but higher dimensions would not give significant improvement.

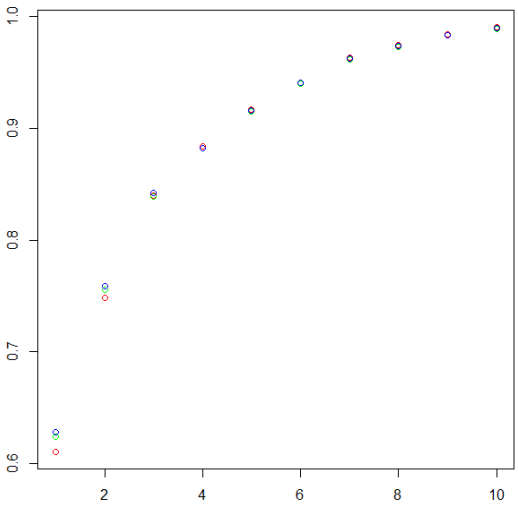
To show this, I use R to compute and plot the GOF for various dimensions for each model (original distance: red, standardized: green, ranged: blue):

R code for GOF plot:

```
# how good is the model (goodness of fit), 0-1: 1 being perfect
plotGOF <- function(nonnrom_mat, stand_mat, range_mat) {
  gof <- list()
  gof_stand <- list()
  gof_range <- list()
  for (i in 1:19) {
    gof <- c(gof, max(cmdscale(nonnrom_mat, k=i, eig = TRUE)$GOF))
    gof_stand <- c(gof_stand, max(cmdscale(stand_mat, k=i, eig = TRUE)$GOF))
    gof_range <- c(gof_range,  max(cmdscale(range_mat, k=i, eig = TRUE)$GOF))
  }
  finalList <- list(gof, gof_stand, gof_range)
  return(finalList)
}

# get GOF of each model type
listGOF <- plotGOF(distances_mat, distances_mat_stand, distances_mat_range)
# plot each one
plot(c(1:19), listGOF[[1]], col="red")
points(c(1:19), listGOF[[2]], col="green")
points(c(1:19), listGOF[[3]], col="blue")
# each of them go to 1 when dimension 16 is reached
# i.e. number of columns in original data
```

* * *

Plot of DOF vs MDS Dimensions:



The GOF plot shows that, as the dimensions increase, so does the GOF of the model. The GOF hits 1 (perfect match) when the dimension is 16 for each model, which is also corresponds to the number of non-zero eigenvalues and the original dimension of the data. The original percentages has the lower GOF in the 1st dimension, followed by the standardized model and then the ranged, but this difference is so small it is hardly significant in higher dimensions. Using the code below, I pulled the 2D GOF for the 3 different models:

R Code for 2D GOFs:

```
gof <- cmdscale(distances_mat, k=2, eig = TRUE)$GOF
gof1 <- cmdscale(distances_mat_stand, k=2, eig = TRUE)$GOF
gof2 <- cmdscale(distances_mat_range, k=2, eig = TRUE)$GOF
```

* * *

- Original: 0.736
- Standardized: 0.744
- Ranged: 0.747

To ensure that this is a good GOF range, I ran 1,000 random distances and their GOF and got a range from 0.149-0.185, which is more than 1/4th of the GOFs in the 2D models above. I therefore argue that a 2D model is relatively effective and is easier to visualize than a higher-dimensional model, making analyzing easier.

R Code for Random Distances and Their GOF:

```
min = 1
max = 0
# to see if really better than random
for (i in 1:1000) {
  randtest <- cmdscale(dist(replicate(20,runif(154))), k = 2, eig = TRUE)$GOF
  if (randtest < min) {
    min = randtest
  }
  if (randtest > max) {
    max = randtest
  }
}
```

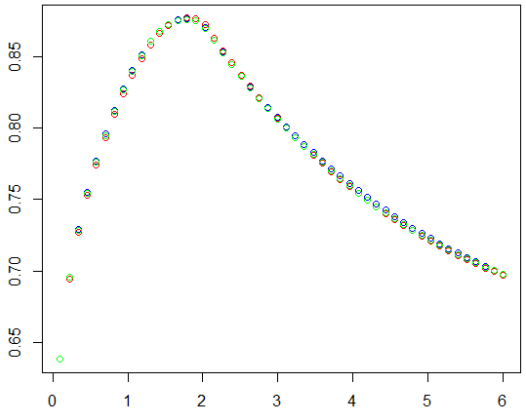
* * *

Another test of goodness I investigated was how the choice of distance function effects the GOF (i.e. the p value for the Minkowski exponent). Below is the R code and plot to compute this for with a constant dimension of 2 (the colors in the plot associate to the same models as the GOF plot above):

R Code for GOF vs Minkowski Exponent Plot:

```
# get the GOF vs Minkowski exponent
plotGOFMin <- function(nonnrom, stand, range) {
  gof <- list()
  gof_stand <- list()
  gof_range <- list()
  for (i in seq(0.1, 6, length.out = 50)) {
    gof <- c(gof, min(cmdscale(dist(nonnrom, method = "minkowski", p=i),
                k=2, eig = TRUE)$GOF))
    gof_stand <- c(gof_stand, min(cmdscale(dist(stand, method = "minkowski", p=i),
                k=2, eig = TRUE)$GOF))
    gof_range <- c(gof_range, min(cmdscale(dist(range, method = "minkowski", p=i),
                k=2, eig = TRUE)$GOF))
  }
  finalList <- list(gof, gof_stand, gof_range)
  return(finalList)
}

# get GOF of each model type
listGOFMin <- plotGOFMin(nonnormalize, normalize_one, normalize_two)
# plot each one
plot(seq(0.1, 6, length.out = 50), listGOFMin[[3]], col="blue")
points(seq(0.1, 6, length.out = 50), listGOFMin[[1]], col="red")
points(seq(0.1, 6, length.out = 50), listGOFMin[[2]], col="green")
```

Plot of GOF vs Mikowski Exponent:



As seen in the plot, although there is some rounding errors, the best p value is 2, or in other words, the classic Euclidean distance. This gives further reason to show a 2D model could give some insight about European countries and the food consumption.

To further investigate the goodness, I compare the mean absolute difference, mean squared difference, and maximum absolute difference between the input distance matrix and the corresponding model's distance matrix. Below is the R code to compute these differences for 2D in each model, respectively:

```
# mean absolute difference for all models
meanabs <- 1/20^2*sum(absdist)

meanabs1 <- 1/20^2*sum(absdist1)

meanabs2 <- 1/20^2*sum(absdist2)


# mean squared difference for all models
meansq <- 1/20^2*sum((distances_mat - as.matrix(dist((cmdscale(
            distances_mat,k=2, eig = TRUE))$points)))^2)

meansq1 <- 1/20^2*sum((distances_mat_stand - as.matrix(dist((
            cmdscale(distances_mat_stand, k=2, eig = TRUE))$points)))^2)

meansq2 <- 1/20^2*sum((distances_mat_range - as.matrix(dist((
            cmdscale(distances_mat_range, k=2, eig = TRUE))$points)))^2)
```

```
# maximum absolute difference for all models
absdist <- abs(distances_mat - as.matrix(dist((cmdscale(
            distances_mat, k=2, eig = TRUE))$points)))
maxabsdist <- max(abs(absdist))

absdist1 <- abs(distances_mat_stand - as.matrix(dist(model4)))
maxabsdist1 <- max(abs(absdist1))

absdist2 <- abs(distances_mat_range - as.matrix(dist(model6)))
maxabsdist2 <- max(abs(absdist2))
```

* * *

Output:
Mean absolute difference:

- Original: 30.378
- Standardized: 0.977
- Ranged: 0.332

Mean squared difference:

- Original: 1429.370
- Standardized: 1.469
- Ranged: 0.169

Maximum absolute difference:

- Original: 105.242
- Standardized: 3.574
- Ranged: 1.148

For reference, the range for the original matrix is 0-276.456, standardized is 0-9.265, and ranged is 0-3.149. The minimum absolute distance is 0 for all of them, and as seen, the largest absolute difference are all under 1/2 of the total range of values for each model, albeit the difference is still substantial. This is representative of the somewhat closeness of the model distances to their input matrices, but that it is not very close/accurate (also shown in the goodness test in the paragraph below). I can also see the maximum percentage error in these models (by dividing the max difference by the original matrix) are 80.15%, 85.01%, 79.49% respectively, which is very large, but again the smallest is also 0%, so the differences for each pair varies greatly.

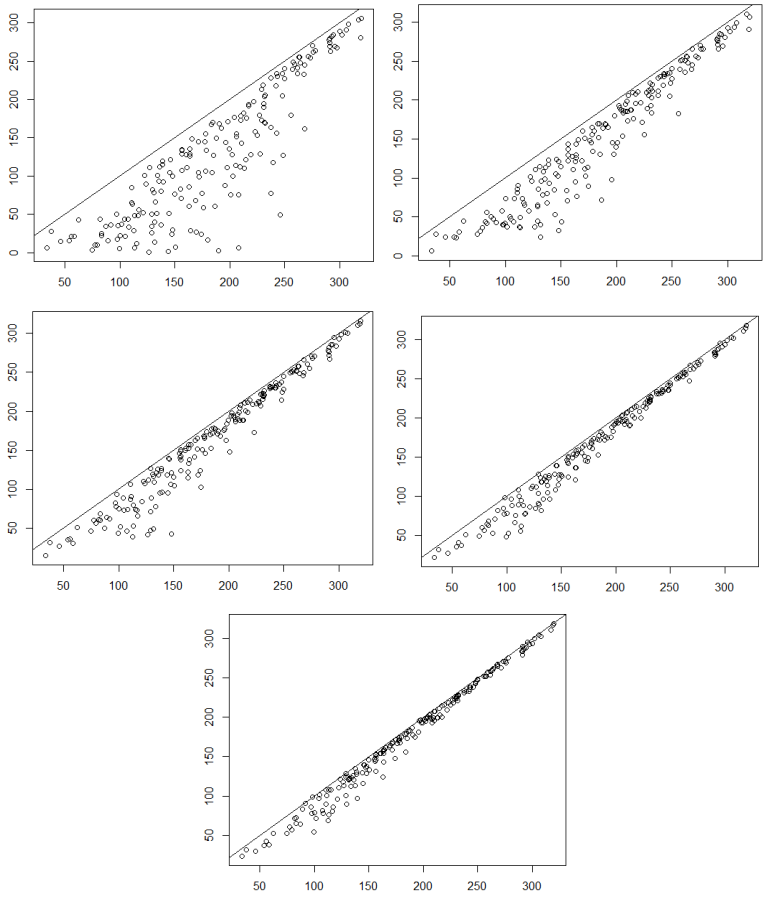
I also compare the input distances to the model distances by plotting each pair. If the distances were perfect matches, each distance pair would be on the y =

x line, as shown as a reference in each plot. The R code and output plots are shown below for the 1st-4th dimensions (order goes from left to right, top to bottom):

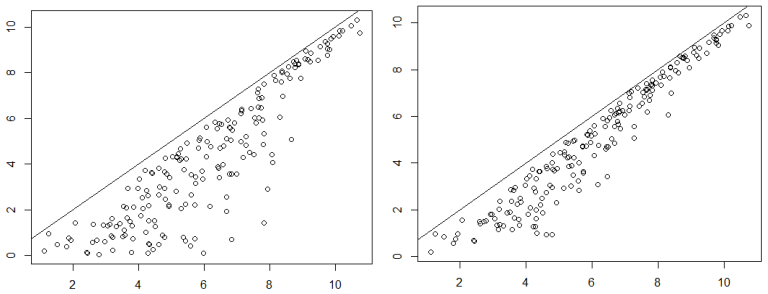
```
# distance in model compared to original distances
# ideally, all points lie on y = x
plot(distances, dist(model1))
abline(0,1)
plot(distances, dist(model2))
abline(0,1)
plot(distances, dist(model2_1))
abline(0,1)
plot(distances, dist(model2_2))
abline(0,1)

plot(distances_stand, dist(model3))
abline(0,1)
plot(distances_stand, dist(model4))
abline(0,1)
plot(distances_stand, dist(model4_1))
abline(0,1)
plot(distances_stand, dist(model4_2))
abline(0,1)

plot(distances_range, dist(model5))
abline(0,1)
plot(distances_range, dist(model6))
abline(0,1)
plot(distances_range, dist(model6_1))
abline(0,1)
plot(distances_range, dist(model6_2))
abline(0,1)
# plot 1 dimensional and 2 dimensional, see if there is an improvement
# should show amount reflected in eigenvalue fall-off
```

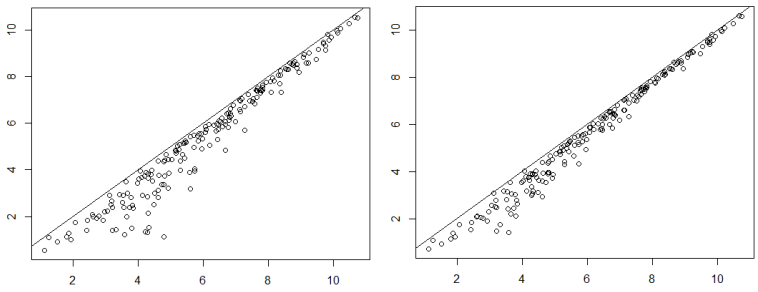
* * *

Plot of Original Distance vs Model:

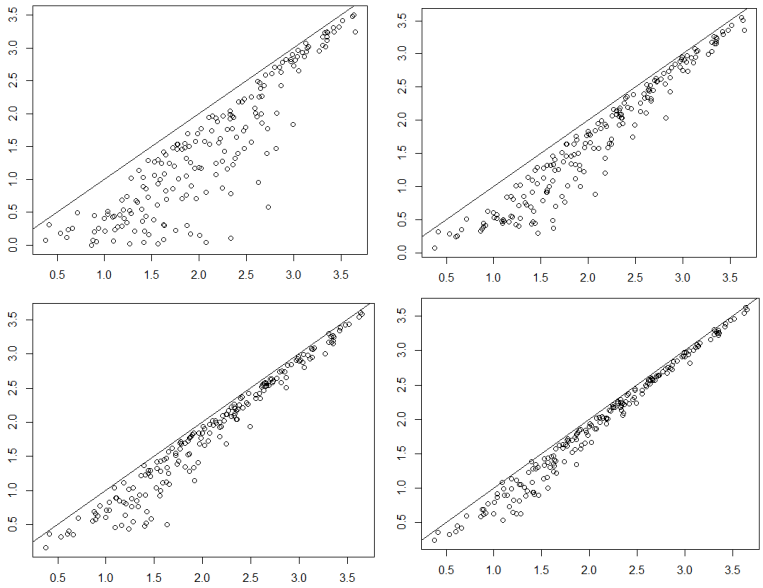


Plot of Standardized Distance vs Model:




2
4
6
8
10
2
4
6
8
10
2
4
6
8
10
2
4
6
8
10

Plot of Ranged Distance vs Model:



As seen in the plots, there is a significant improvement from 1D to 2D models, but the amount of improvement decreases as the dimension increases. The higher dimension will always be better than the one below it, but as I want a low-dimension model that I can visualize, I argue that the 2D model is somewhat near the y=x line, especially for larger distances.

I reiterate this similarity but not exactness in a histogram of the same comparisons, where the blue is the input distances and orange is the methods:

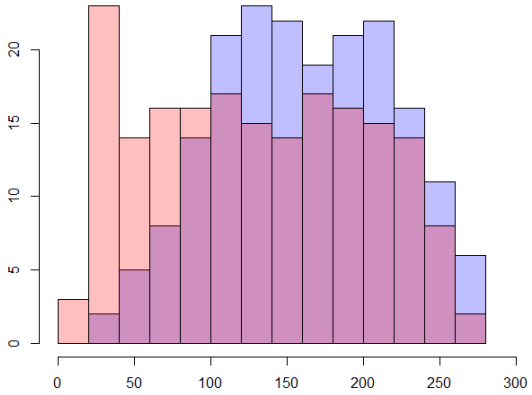
R Code for Histogram 2D Distance Comparisons:

```
# histogram of original distances vs histogram in the MDS model
# ideally it would look the same
p1 <- hist(distances, breaks = 15)
p2 <- hist(dist(model2), breaks = 15)
plot(p1, col=rgb(0,0,1,1/4), xlim=c(0,300)) # plot first
plot(p2, col=rgb(1,0,0,1/4), xlim=c(0,300), add=T)

p3 <- hist(distances_stand, breaks = 15)
p4 <- hist(dist(model4), breaks = 15)
plot(p3, col=rgb(0,0,1,1/4), xlim=c(0,10))  # first histogram
plot(p4, col=rgb(1,0,0,1/4), xlim=c(0,10), add=T)

p5 <- hist(distances_range, breaks = 15)
p6 <- hist(dist(model6), breaks = 15)
```

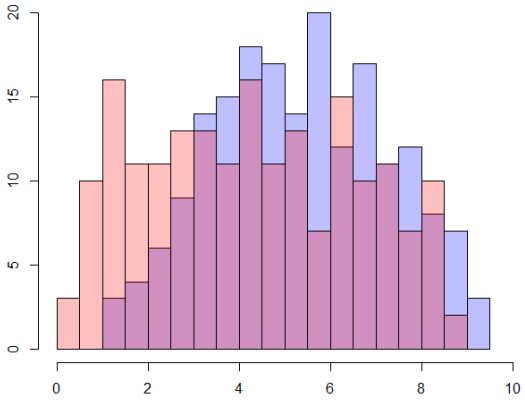
```
plot(p5, col=rgb(0,0,1,1/4), xlim=c(0,3.5))  # first histogram
plot(p6, col=rgb(1,0,0,1/4), xlim=c(0,3.5), add=T)
```

* * *

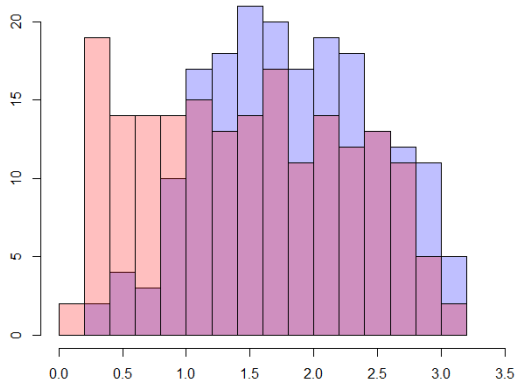
Plot of Original Distance vs Model:



Plot of Standardized Distance vs Model:



Plot of Ranged Distance vs Model:



While there is a good amount of similarity, the model tends to have more smaller distances, and has its highest bin in the smaller distances rather than near the center like the input distances.

In summary, the 2D model for each of the methods is not in any way the most accurate dimension, but for visualization purposes, the 2D model is "good" enough to make some analysis and be in assurance it is at the very least better than random distances.

4 Analysis/Findings

For easier identification and visualization, I have outputted the 2D plots for each model with the points as the name of the food types:

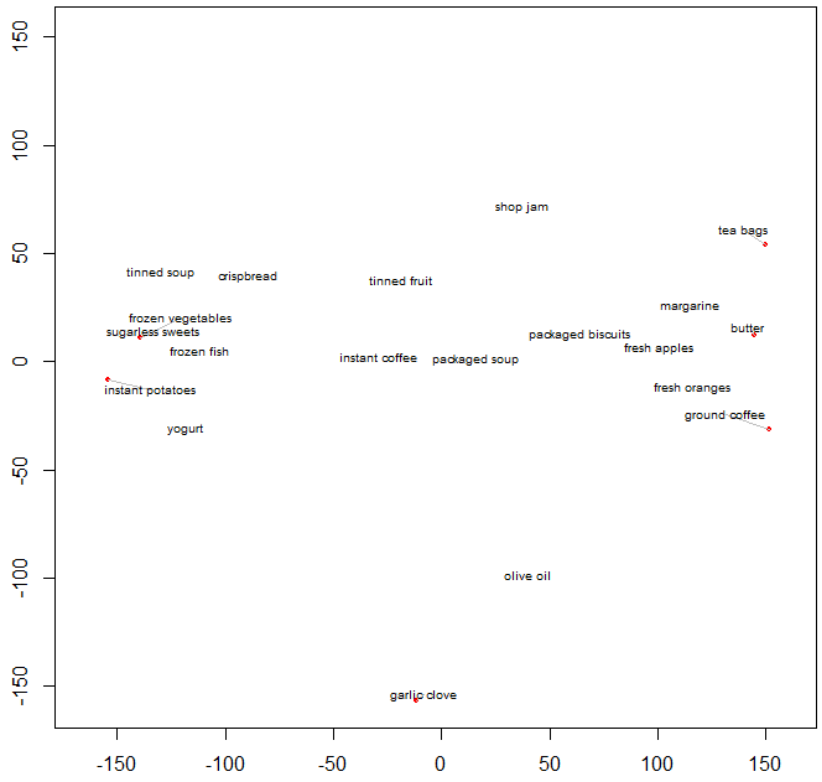
R Code for 2D Models of Distances:

```
# 2D models with names of food
textplot(model2[,1], model2[,2],
         mydata$V1,
         asp = 1,
         xlim = c(min(model2[,2]), max(model2[,1])),
         ylim = c(min(model2[,2]), max(model2[,1])),
         cex=0.6)

textplot(model4[,1], model4[,2],
         mydata$V1,
         asp = 1,
         xlim = c(min(model4[,2]), max(model4[,1])),
         ylim = c(min(model4[,2]), max(model4[,1])),
```

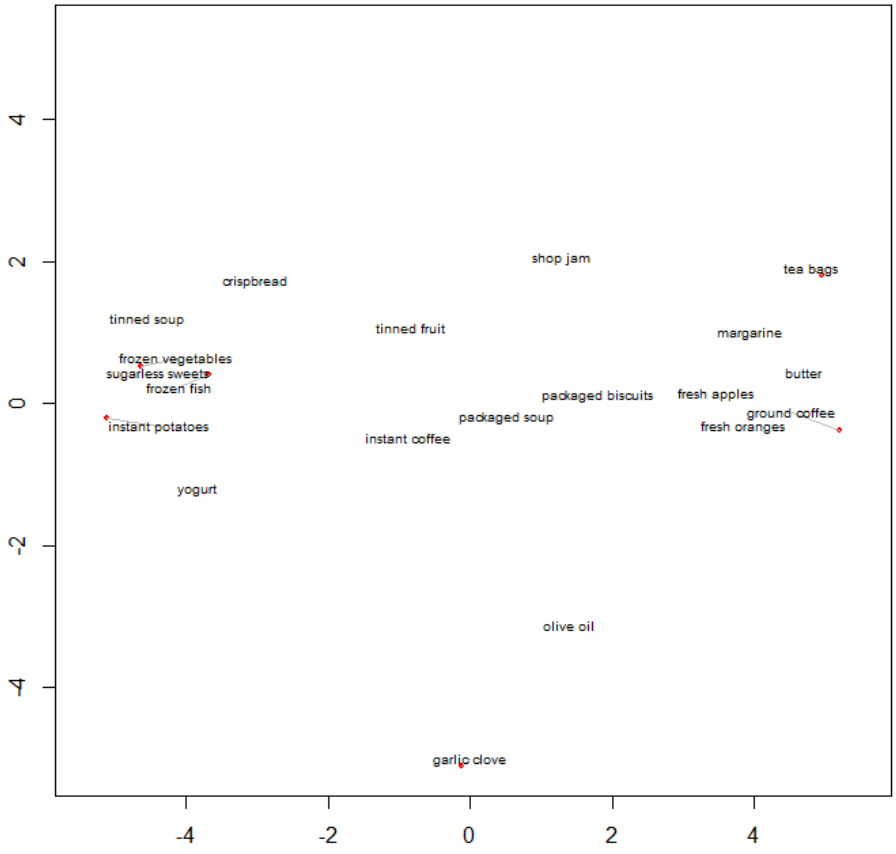
```
        cex=0.6)

textplot(model6[,1], model6[,2],
        mydata$V1,
        asp = 1,
        xlim = c(min(model6[,1]), max(model6[,2])),
        ylim = c(min(model6[,2]), max(model6[,2])),
        cex=0.6)
```

* * *

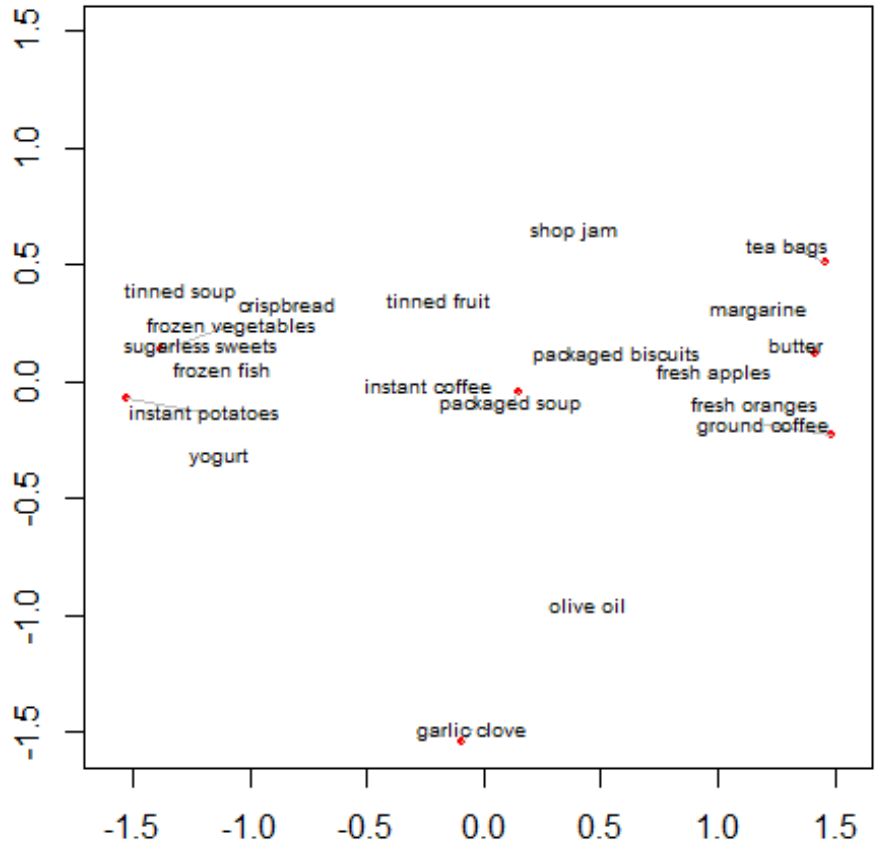
Plot of 2D Model of Original Distances::



Plot of 2D Model of Standardized Distances::



Plot of 2D Model of Ranged Distances::



As I had initially predicted, since all the values in the data were percentages and could be compared against each other easily, only the scale is different between the original and range models as the coordinates hold the same relative positions in the models (with only very slight differences in the standardized model). In the analysis, I will use the model that uses the original percentages (no normalization). Most of the food are grouped around each other with the exception of garlic cloves. Looking at the table with the original percentages, the percentage of households with for garlic cloves in each country tends to be the opposite of the majority percentages (i.e, when the other countries have high percentages for foods, the garlic cloves' percentage is low, and vice versa).

Starting with the horizontal (x) axis, I believe this relationship is showing the popularity (percentage) of certain foods as a whole across all countries. To support this claim, I computed in R the correlation coefficients each country's percentage has against the x and y values in the model, (i.e. the horizontal and

vertical plots):

R Code for Correlation Coefficients:

```
# find correlation coefficients for x and y
xcoeff <- list()
ycoeff <- list()
for (i in 1:16) {
  curr <- cor(model2[,1], nonnormalize[,i])
  curr1 <- cor(model2[,2], nonnormalize[,i])
  xcoeff <- c(xcoeff, curr)
  ycoeff <- c(ycoeff, curr1)
}

coeffall <- data.frame(mydata_1$V2, unlist(xcoeff), unlist(ycoeff))
```

* * *

Table for Correlation Between Country and X and Y for 2D Models:

Countries	x	y
West Germany	0.951517	0.125729
Italy	0.796177	-0.46643
France	0.799265	-0.32877
Netherlands	0.702908	0.366568
Belgium	0.903858	-0.08872
Luxemburg	0.861372	-0.32778
Great Britian	0.619996	0.477933
Portugal	0.723044	-0.51172
Austria	0.886833	-0.11104
Switzerland	0.816059	-0.18482
Sweden	0.395621	0.405739
Denmark	0.816738	0.385764
Norway	0.834198	0.317982
Finland	0.805853	0.289766
Spain	0.735528	-0.60114
Ireland	0.671113	0.368656

As seen in the x column, all of the correlations are all high (with the exception of Sweden), so this also shows that is not a unique characteristic between specific countries to show a relationship and instead shows a general trend for the popularity of foods across all countries. To address Swedend's low correlation, I refer to Sweden's true percentages in the original data, I observed a 0% for packaged biscuits, which does not seem accurate, as it seems impossible for a country to not have any packaged biscuits (if there was a low amount in households, other countries at the very least had 2 or 1%). This potential error is where I believe

the lack of correlation occurred for Sweden, as other countries tend to have a very high percentage of households having packaged biscuits. Another way to support this claim is to look at the average of the percentages for each food type. Below is the R computation to find the mean value across all countries:

R Code for Average Food Percentages:

```
# get avg of foods
averages <- list()
for (i in 1:20) {
  curravg <- mean(nonnormalize[i,])
  averages <- c(averages, curravg)
}

avgfood <- data.frame(mydata$V1, unlist(averages))
```

* * *

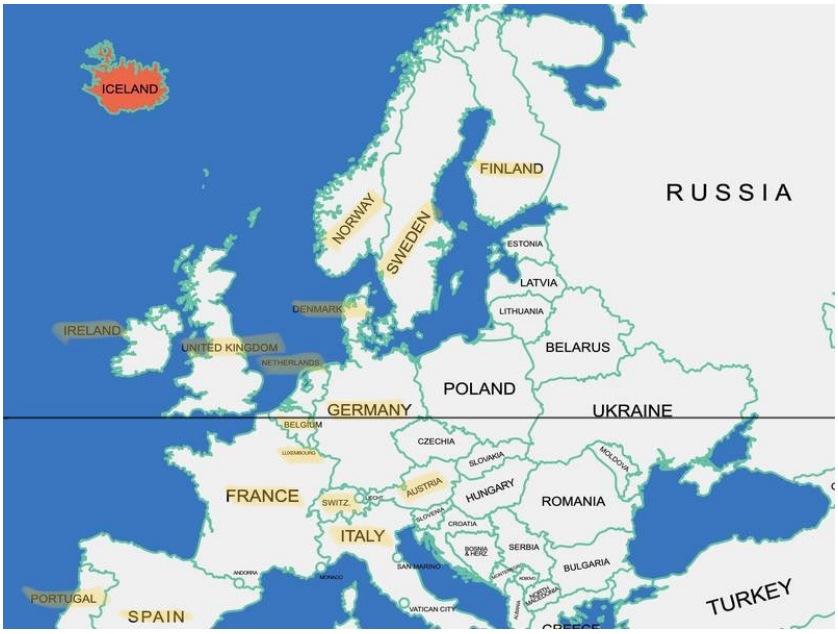
Table for Food Averages:

food	avg
instant potatoes	12.75
frozen vegetables	15.875
sugarless sweets	16.875
tinned soup	18.4375
yogurt	19.25
frozen fish	21.875
crispbread	27.75
instant coffee	39.25
tinned fruit	41.9375
garlic clove	42.3125
packaged soup	49
olive oil	54.1875
shop jam	55.1875
packaged biscuits	56.875
fresh apples	66.8125
margarine	69.125
fresh oranges	70.5
butter	75.8125
ground coffee	77.5
tea bags	78.5

As seen in the table and the plot of the 2D model, the average values (from smallest to largest) closely follows the order of the x-axis order of ingredients in the plot. This claim shows that butter, ground coffee, fresh fruit, and margarine are common in European countries, while instant potatoes, frozen foods, sugarless sweets, tinned soup, etc. are not as common to find in a European household.

Now looking at the vertical (y) relationship, I believe that there is an association between geographical location, where Northern European countries have a positive association with percentages of food in households, while Southern European countries have a negative association. In the map below, I have highlighted in yellow the countries that were included in this study, and I define a southern country to have its territory completely under the black line.

Map for y Relationship:



From the table comparing the x and y axis in the 2D model to the original percentages, I have highlighted the countries that are north and south of the black line (red is south, green is north countries):

Table for y Relationship:

Countries	x	y
West Germany	0.951517	0.125729
Italy	0.796177	-0.46643
France	0.799265	-0.32877
Netherlands	0.702908	0.366568
Belgium	0.903858	-0.08872
Luxemburg	0.861372	-0.32778
Great Britian	0.619996	0.477933
Portugal	0.723044	-0.51172
Austria	0.886833	-0.11104
Switzerland	0.816059	-0.18482
Sweden	0.395621	0.405739
Denmark	0.816738	0.385764
Norway	0.834198	0.317982
Finland	0.805853	0.289766
Spain	0.735528	-0.60114
Ireland	0.671113	0.368656

While not all of the relationships are strong (especially the countries near the black line), countries on the north side of the line all have a positive correlation while countries on the south side of the line all have a negative correlation with percentage of food in a European country's household.