In [0]: Network analysis NLP

```
#load the data by installing gdown
import os.path as path
if not path.exists('food-com-recipes-and-user-interactions.zip'):
  !pip install gdown
  !gdown https://drive.google.com/uc?id=1CK99ASX3fsQ_KBY_RP7Nqms6dwsNs-jb
  !unzip food-com-recipes-and-user-interactions.zip
else :
  print('data is in places')
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.6/dist-packages (3.6.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (from gdown
) (4.28.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from gdown)
(1.12.0)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from g
down) (2.21.0)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-pac
kages (from requests->gdown) (1.24.3)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-pac
kages (from requests->gdown) (3.0.4)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (
from requests->gdown) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packag
es (from requests->gdown) (2019.9.11)
Downloading...
From: https://drive.google.com/uc?id=1CK99ASX3fsQ_KBY_RP7Nqms6dwsNs-jb
To: /content/food-com-recipes-and-user-interactions.zip
275MB [00:02, 127MB/s]
Archive:  food-com-recipes-and-user-interactions.zip
  inflating: ingr_map.pkl
  inflating: __MACOSX/._ingr_map.pkl
  inflating: interactions_test.csv
  inflating: __MACOSX/._interactions_test.csv
  inflating: interactions_train.csv
  inflating: __MACOSX/._interactions_train.csv
  inflating: interactions_validation.csv
  inflating: __MACOSX/._interactions_validation.csv
  inflating: new_data.csv
  inflating: __MACOSX/._new_data.csv
  inflating: PP_recipes.csv
  inflating: __MACOSX/._PP_recipes.csv
  inflating: PP_users.csv
  inflating: __MACOSX/._PP_users.csv
  inflating: RAW_interactions.csv
  inflating: __MACOSX/._RAW_interactions.csv
  inflating: RAW_recipes.csv
  inflating: __MACOSX/._RAW_recipes.csv
```

In [0]:

```
#Load packages
import pandas as pd
import numpy as np
import networkx as nx
from networkx.algorithms import bipartite
import matplotlib.pyplot as plt
from collections import Counter
from datetime import datetime
from datetime import timedelta
import community
import itertools
```

In [0]:

```
#import the data and inspect it
df_in = pd.read_csv('RAW_interactions.csv')
```

```
df_in.head()
```

Out[0]:

	user_id	recipe_id	date	rating	review
0	38094	40893	2003-02-17	4	Great with a salad. Cooked on top of stove for...
1	1293707	40893	2011-12-21	5	So simple, so delicious! Great for chilly fall...
2	8937	44394	2002-12-01	4	This worked very well and is EASY. I used not...
3	126440	85009	2010-02-27	5	I made the Mexican topping and took it to bunk...
4	57222	85009	2011-10-01	5	Made the cheddar bacon topping, adding a sprin...

In [0]:

```
#change rating to decimal
df_in['w'] = df_in.apply(lambda x: 0 if x['rating'] == 0 else x['rating'] / 5, axis=1)
```

In [0]:

```
#change date to datetime
df_in['date'] = pd.to_datetime(df_in['date'], errors = 'coerce')
#filter the df from mid of 2018 until now to be more "notebook-friendly" as the original
one is too big (more than 1 million rows)
start_date = '06-01-2018'
date_filter = (df_in['date'] > start_date)
df_int = df_in.loc[date_filter]
df_int.head()
```

Out[0]:

	user_id	recipe_id	date	rating	review	w
200	2000551249	195977	2018-07-11	5	We have substituted a lemon cake mix for the y...	1.0
201	2002265401	195977	2018-09-03	2	I will start by saying that I followed the rec...	0.4
202	2001205226	195977	2018-11-03	5	This is an exceptionally tasty cake! Very swee...	1.0
250	615758	373842	2018-06-24	5	Made this for dinner last night and had it for...	1.0
276	246482	217012	2018-07-04	4	Very nice I got 18 muffins out of it but I thi...	0.8

In [0]:

```
#Do the same with the other dataframe "recipe"
df_re = pd.read_csv('RAW_recipes.csv')
df_re.rename(columns={'id':'recipe_id'}, inplace=True)
df_re.set_index('recipe_id', inplace=True)
df_re.head()
```

Out[0]:

	name	minutes	contributor_id	submitted	tags	nutrition	n_steps	steps	description	ingredie
recipe_id										
137739	arriba baked winter squash mexican style	55	47892	2005-09-16	['60-minutes-or-less', 'time-to-make', 'course...	[51.5, 0.0, 13.0, 0.0, 2.0, 0.0, 4.0]	11	['make a choice and proceed with recipe', 'dep...	autumn is my favorite time of year to cook! th...	['win squas 'mexic seasonin 'mixed
31490	a bit different breakfast pizza	30	26278	2002-06-17	['30-minutes-or-less', 'time-to-make', 'course...	[173.4, 18.0, 0.0, 17.0, 22.0, 35.0, 1.0]	9	['preheat oven to 425 degrees f', 'press dough...	this recipe calls for the crust to be prebaked...	['prepar pizza cru 'sausa patt 'eg
						[269.8,			this	['grou

recipe_id	name	minutes	contributor_id	submitted	tags	nutrition	n_steps	steps	description	ingredie
112140	all in the kitchen chili	130	196586	2005-02-25	['time-to-make', 'course', 'preparation', 'mai...	22.0, 32.0, 48.0, 39.0, 27.0, 5.0]	6	['brown ground beef in large pot', 'add choppe...	this modified version of 'mom's' chili was a h...	['grou 'yell onior 'dic tomat
59389	alouette potatoes	45	68585	2003-04-14	['60-minutes-or-less', 'time-to-make', 'course...	[368.1, 17.0, 10.0, 2.0, 14.0, 8.0, 20.0]	11	['place potatoes in a large pot of lightly sal...	this is a super easy, great tasting, make ahea...	['spreadal cheese w garlic a herbs', 'ı
44061	amish tomato ketchup for canning	190	41706	2002-10-25	['weeknight', 'time-to-make', 'course', 'main-...	[352.9, 1.0, 337.0, 23.0, 3.0, 0.0, 28.0]	5	['mix all ingredients& boil for 2 1 / 2 hours ...	my dh's amish mother raised him on this recipe...	['toma juic 'apple cic vinega 'suga

In [0]:

```
df_re['submitted'] = pd.to_datetime(df_re['submitted'], errors = 'coerce')
start_date = '06-01-2018'
date_filter = (df_re['submitted'] > start_date)
df_rep = df_re.loc[date_filter]
df_rep.head()
```

Out[0]:

recipe_id	name	minutes	contributor_id	submitted	tags	nutrition	n_steps	steps	description	ingredients
537485	5 ingredient salted caramel crumble bars	45	2000378667	2018-11-12	['60-minutes-or-less', 'time-to-make', 'course...	[52.8, 3.0, 0.0, 4.0, 1.0, 1.0, 2.0]	21	['1', 'heat oven to 350f spray 8-inch square p...	delicious	['pillsbury sugar cookie dough', 'caramel topp...
536322	amish peanut butter	45	1052873	2018-07-15	['60-minutes-or-less', 'time-to-make', 'course...	[253.5, 16.0, 107.0, 4.0, 11.0, 11.0, 12.0]	7	['in large saucepan over medium heat , combine...	super sweet peanut butter spread	['brown sugar', 'water', 'corn syrup', 'maple ...
536384	amish triple butter biscuits	40	219942	2018-07-19	['60-minutes-or-less', 'time-to-make', 'course...	[410.5, 38.0, 6.0, 23.0, 13.0, 77.0, 13.0]	9	['pre-heat the oven to 425 degrees', 'while ov...	the way she describes these biscuits on her we...	['flour', 'salt', 'cream of tartar', 'sugar', ...
536318	argentine chili	50	400708	2018-07-15	['60-minutes-or-less', 'time-to-make', 'course...	[664.8, 67.0, 14.0, 42.0, 112.0, 82.0, 3.0]	8	['heat a heavy pot or dutch oven over high hea...	an argentina influenced chili by rachel ray to...	['olive oil', 'lean ground sirloin', 'chorizo ...
536401	argentinian steak sandwiches	25	47559	2018-07-21	['30-minutes-or-less', 'time-to-make', 'course...	[573.7, 62.0, 9.0, 13.0, 57.0, 47.0, 8.0]	11	['to make chimichurri sauce , combine all ingr...	a simple and quick hearty sandwich with loads ...	['beef sirloin steaks', 'olive oil', 'salt and...

Network analysis

Create network, a bipartite graph between users and recipes

The users are the vertices U and the recipes are the vertices V and there is an edge from u to v if u reviewed v. In this case the edges are weighted by the rating the users gave.

In [0]:

```
#have a look at types in df
print('log out types in dataframe interaction')
print(df_int.dtypes)

print('log out types in dataframe recipe')
print(df_rep.dtypes)
```

```
log out types in dataframe interaction
user_id                 int64
recipe_id               int64
date           datetime64[ns]
rating                  int64
review                 object
w                     float64
dtype: object
log out types in dataframe recipe
name                      object
minutes                    int64
contributor_id             int64
submitted         datetime64[ns]
tags                      object
nutrition                 object
n_steps                    int64
steps                     object
description               object
ingredients               object
n_ingredients              int64
dtype: object
```

Next, we create bipartite graph between users and recipe.

A network compromises of nodes and edges, the nodes representing the elements of the system while the edges are the relationship between them. For a bipartite graph(B = (U, V, E)) the nodes are partitioned into two sets. The nodes in one set cannot be connected to each other, they can only be connected to the nodes in the other set

If each edge in graph G has an associated weight, the graph G is called a weighted bipartite graph.

In [0]:

```
#Instantiating an empty directed graph. This will create a new Graph object, G, with noth
ing in it. Now we can add lists of nodes and edges.
G = nx.Graph()
G.add_nodes_from(df_int['recipe_id'], bipartite=1)
#set node attributes
nx.set_node_attributes(G, df_rep['nutrition'].to_dict(), 'nutrition')
nx.set_node_attributes(G, df_rep['n_steps'].to_dict(), 'n_steps')
nx.set_node_attributes(G, df_rep['minutes'].to_dict(), 'minutes')
nx.set_node_attributes(G, df_rep['tags'].to_dict(), 'tags')

G.add_nodes_from(df_int['user_id'], bipartite=0)
G.add_weighted_edges_from([(row['user_id'], row['recipe_id'], row['w']) for idx, row in d
f_int.iterrows()], weight='weight')
```

In [0]:

```
#have a look at the edges
print(list(G.edges(data=True))[:5])
```

```
[(195977, 2000551249, {'weight': 1.0}), (195977, 2002265401, {'weight': 0.4}), (195977, 2
```

```
[(195977, 2000331249, {'weight': 1.0}), (195977, 2002203401, {'weight': 0.4}), (195977, 2
001205226, {'weight': 1.0}), (373842, 615758, {'weight': 1.0}), (217012, 246482, {'weight
': 0.8})]
```

In [0]:

```python
# print some stats
numberOfNodes = G.number_of_nodes()
numberOfNodesInB0 = sum([1 for n in G.nodes(data=True) if n[1]['bipartite'] == 0])
numberOfNodesInB1 = sum([1 for n in G.nodes(data=True) if n[1]['bipartite'] == 1])

print('number of nodes: ', numberOfNodes)
print('number of nodes in B0: ', numberOfNodesInB0)
print('number of nodes in B1: ', numberOfNodesInB1)
print('number of edges: ', G.number_of_edges())
```

```
number of nodes:  13916
number of nodes in B0:  7879
number of nodes in B1:  6037
number of edges:  9424
```

In [0]:

```python
#create the user nodes partition and the recipe nodes one
user_nodes = {n for n, d in G.nodes(data=True) if d['bipartite'] == 0}
recipe_nodes = {n for n, d in G.nodes(data=True) if d['bipartite'] == 1}
```

A bipartite projection of a graph returns a graph containing only the nodes that belong to the user's partition and edges between them.

The weighted projected graph is the projection of the bipartite network onto the specified nodes and the weights are the number of shared neighbors.

In [0]:

```python
#Create the user nodes projection as a graph using a weighted projection
user_graph = bipartite.weighted_projected_graph(G, user_nodes)
```

Next we will compute the partition of the graph nodes where the modularity is maximized using the Louvain heuristices. This is the partition of highest modularity, i.e. the highest partition of the dendrogram generated by the Louvain algorithm.

In [0]:

```python
#implement community detection
user_communities = community.best_partition(user_graph, resolution = 1)
nx.set_node_attributes(user_graph, user_communities, 'usercommunity')
```

In [0]:

```python
#Calculate eigenvector centrality and set it as an attribute
user_eigenvector = nx.eigenvector_centrality(user_graph)
nx.set_node_attributes(user_graph, user_eigenvector, 'eigenvector_centrality')
```

In [0]:

```python
# Create a new attribute "activity" to see how active a user posts reviews about recipes
nx.set_node_attributes(user_graph, dict(df_int.user_id.value_counts()), 'activity')
removeNodes = set()
for n, d in user_graph.nodes(data=True):
  if 'activity' not in d:
    removeNodes.add(n)
for n in removeNodes:
  user_graph.remove_node(n)
print('Number of nodes:', user_graph.number_of_nodes())
```

```
Number of nodes: 7879
```

Assortativity **measures the similarity of connections in the graph with respect to the number of neighbors the**

node has

In [0]:

```
print('Assortativity coefficient:', nx.numeric_assortativity_coefficient(user_graph,'acti
vity'))
```

```
Assortativity coefficient: 0.3455518109365992
```

In [0]:

```
#Let's see the result
graph_proj_df = pd.DataFrame(dict(user_graph.nodes(data=True))).T
graph_proj_df.usercommunity.value_counts(normalize=True)
graph_proj_df.head()
```

Out[0]:

	bipartite	usercommunity	eigenvector_centrality	activity
2002190344	0.0	0.0	4.744473e-19	1.0
1441804	0.0	1.0	4.744473e-19	1.0
2002157583	0.0	2.0	8.367860e-02	8.0
2002255891	0.0	3.0	7.968893e-13	1.0
2002190356	0.0	4.0	3.109338e-14	1.0

Next is the Eigenvector Centrality, which decides that a node is important if it is connected to other important nodes. It computes the centrality for a node based on the centrality of its neighbors.

In [0]:

```
# Find the 5 most central for each identified community
user_per_com = graph_proj_df.groupby('usercommunity')['eigenvector_centrality'].nlargest(
5)
user_per_com
```

Out[0]:

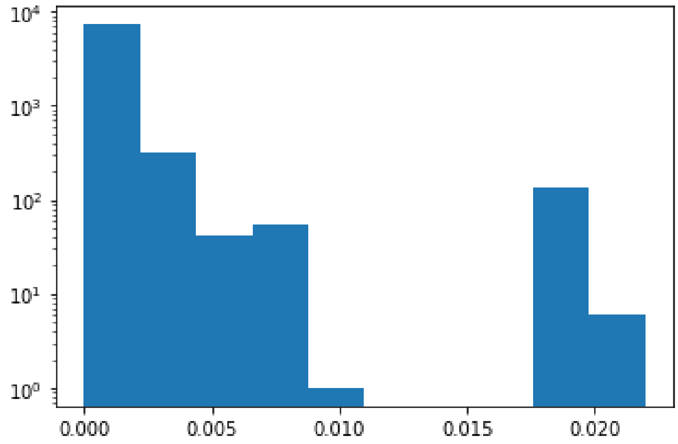
```
usercommunity
0.0            2002190344    4.744473e-19
1.0            1441804       4.744473e-19
2.0            2002216720    8.377065e-02
               2000637026    8.371248e-02
               2002188458    8.370697e-02
                                 ...
4678.0         2818015       4.744473e-19
4679.0         2002288613    4.744473e-19
4680.0         2002288620    4.744473e-19
4681.0         2002288621    4.744473e-19
4682.0         2001829873    4.744473e-19
Name: eigenvector_centrality, Length: 6458, dtype: float64
```

Degree centrality represents the number of neighbors a node has divided by the numbers of neighbors it could possibly have. Nx.degree_centrality(G) returns a dictionary in which the key is the node and the value is the degree centrality score for that node. The degree of the node is the number of neighbors that it has. Degree centrality for bipartite graphs is the number of nodes in the opposite partition. This is why we have to pass in the list of nodes from one partition into the Networkx bipartite degree centrality function.

In [0]:

```
# Calculate the degree centrality using nx.degree_centrality: dcs and plot the graph
dcs = nx.degree_centrality(user_graph)

plt.hist(list(dcs.values()))
plt.yscale('log')
plt.show()
```

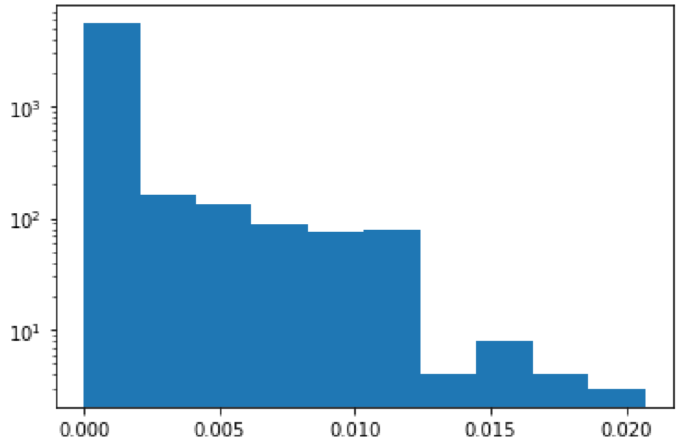


In [0]:

```
#Do the same
nx.bipartite.degree_centrality(G, recipe_nodes)

# Create the recipe nodes projection as a graph: G_recipe
G_recipe = nx.bipartite.projected_graph(G, nodes=recipe_nodes)

# Calculate the degree centrality using nx.degree_centrality: dcs
dcs = nx.degree_centrality(G_recipe)
plt.hist(list(dcs.values()))
plt.yscale('log')
plt.show()
```



Betweenness centrality represents the number of shortest paths through a node divided by all possible shortest paths. It captures bottleneck nodes in a graph rather than highly connected nodes.

In [0]:

```
#Top 2 most
print(sorted(nx.betweenness_centrality(user_graph).items(), key=lambda x: x[1], reverse=
True)[:2])
```

```
[(2123645, 0.004022114401750812), (2002295280, 0.004017610333215215)]
```

M2 Project - preprocessing

In this part we will have to do preprocessing again separately from the main part, because the two parts kept making errors. We will show the codes we did in the preprocessing part, then we saved the pd.DataFrame as a new csv and started a new colab where we uploaded this new csv file and worked on it later on

In [0]:

```
# Import the necessary packages
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

In [0]:

```
# Open file from Google Drive
from google.colab import drive
drive.mount('/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=94731898
9803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%
3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.tes
t%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2F
auth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readon
ly&response_type=code

Enter your authorization code:
··········
Mounted at /gdrive
```

In [0]:

```
df1 = pd.read_csv('/gdrive/My Drive/RAW_recipes.csv')
```

In [0]:

```
# Importing the data set
df = pd.read_csv('/gdrive/My Drive/RAW_interactions.csv')
```

In [0]:

```
#renaming the columns
df1.rename(columns={'id':'recipe_id'}, inplace=True)
```

In [0]:

```
# Delete rows with missing values
df.dropna(inplace=True)
```

Creating dataframe and mergin it on recipe_id to connect the review rate with the correct recipe

In [0]:

```
new_df = pd.merge(df, df1, on='recipe_id')
new_df.set_index('recipe_id')
new_df.dropna(inplace=True)
```

Here we delete everything else, except the rating of the recipe and the ingredients the recipe contained

In [0]:

```
df = new_df.drop(columns=['user_id', 'recipe_id', "date", "review", "name", "minutes", "
contributor_id", "tags", "submitted", "description", "nutrition", "n_steps", "steps", "n
_ingredients"])
```

NLP - preparation

Extraction & Cleaning, Tokenization, Filtering & Lemmatization / Stemming

The data is to some part preprocessed, but in the case we would need to do some initial cleaning up, these are the codes we would use for tokenazing, lemmatizing and cleaning, the only thing we need to do is remove commas, and remove some words such as "white, black or red" - usually attributes of food

In [0]:

```
# library to clean data
 import re

# Natural Language Tool Kit
 import nltk

 nltk.download('stopwords')

# to remove stopword
 from nltk.corpus import stopwords

# for Stemming propose
 from nltk.stem.porter import PorterStemmer
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [0]:

```
# Initialize empty array
# to append clean text
# corpus = []

# for i in range(len(data["tags"])):

    # column : "Review", row ith
    # ingredients = re.sub('[^a-zA-Z]', ' ', data['ingredients'][i])

    # convert all cases to lower cases
    # ingredients = ingredients.lower()

    # split to array(default delimiter is " ")
    # ingredients = ingredients.split()

    # creating PorterStemmer object to
    # take main stem of each word
    # ps = PorterStemmer()

    # loop for stemming each word
    # in string array at ith row
    # ingredients = [ps.stem(word) for word in ingredients
                # if not word in set(stopwords.words('english'))]

    # rejoin all string array elements
    # to create back into a string
    # ingredients = ' '.join(ingredients)

    # append each string to create
    # array of clean text
    # corpus.append(ingredients)
```

In [0]:

```
import string
string.punctuation
```

Out[0]:

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [0]:

```
#Function to remove Punctuation
def remove_punct(text):
    text_nopunct = "".join([char for char in text if char not in string.punctuation])
    return text_nopunct

df['ingredients'] = df['ingredients'].apply(lambda x: remove_punct(x))

df.head()
```

Out[0]:

	rating	ingredients
0	4	great northern beans yellow onion diced green ...
1	5	great northern beans yellow onion diced green ...
3	5	mayonnaise salsa cheddar cheese refried beans ...
4	5	mayonnaise salsa cheddar cheese refried beans ...
5	4	raspberries granulated sugar

Data Exploration

In [0]:

```
df.shape
```

Out[0]:

```
(1108688, 2)
```

In [0]:

```
df["rating"].value_counts()
```

Out[0]:

```
5    800161
4    182823
0     59702
3     39816
2     13728
1     12458
Name: rating, dtype: int64
```

Here we found out that we have a huge disproportion of 5 stars reviews in comparison to 1 star reviews. Thats why we decided to sample our data so that every review has the same equal representation of recipes

the code is below

In [0]:

```
def sampling_dataset(df_ingredients_tok):
    count = 10000
    class_df_sampled = pd.DataFrame(columns = ["rating", "ingredients"])
    temp = []
    for c in df.rating.unique():
        class_indexes = df[df.rating == c].index
        random_indexes = np.random.choice(class_indexes, count, replace = False)
        temp.append(df.loc[random_indexes])

    for each_df in temp:
        class_df_sampled = pd.concat([class_df_sampled,each_df],axis=0)

    return class_df_sampled
```

```
data = sampling_dataset(df_ingredients_tok)
data.reset_index(drop=True,inplace=True)
print (data.head())
print (data.shape)
```

```
  rating                                        ingredients
0      4  pork chop flour garlic powder salt pepper seas...
1      4  extra virgin olive oil garlic cloves onion pan...
2      4  butter olive oil vidalia onion garlic cloves r...
3      4  olive oil onion salt white pepper chicken stoc...
4      4  pork chops salt and pepper garlic powder pork ...
(60000, 2)
```

Here we save the dataset to new csv file and download it

In [0]:

```
data.to_csv("new_data.csv")
```

In [0]:

```
from google.colab import files
files.download('new_data.csv')
```

note it takes around 16 min to load the whole colab

Here we make our own code to open a file directly from the google docs

In [53]:

```
# install gdown
import os.path as path
if not path.exists('food-com-recipes-and-user-interactions.zip'):
 !pip install gdown
 !gdown https://drive.google.com/uc?id=1CK99ASX3fsQ_KBY_RP7Nqms6dwsNs-jb
 !unzip food-com-recipes-and-user-interactions.zip
else :
 print('data is in places')
```

```
data is in places
```

In [0]:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

In [0]:

```
df = pd.read_csv('new_data.csv')
```

In [0]:

```
df = df.rename(columns={'ingients': 'ing'})
```

In [57]:

```
#Checking the head of the dataframe
df.head()
```

Out[57]:

	Unnamed: 0	rating	ing
0	0	4	pork chop flour garlic powder salt pepper seas...
1	1	4	extra virgin olive oil garlic cloves onion pan...
2	2	4	butter olive oil vidalia onion garlic cloves ...
3	3	4	olive oil onion salt pepper chicken stock gar...
4	4	4	pork chops salt and pepper garlic powder pork ...

Borrowed explanation from geeksforgeeks.com

"Tokenization is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph."

Tokenizing the text, here we get the help from tweet tokenizer

In [0]:

```
from nltk.tokenize import TweetTokenizer
tknzr = TweetTokenizer()
df['Tokens'] = df['ing'].map(lambda textline: [tag for tag in tknzr.tokenize(textline)])
tokens = df['Tokens'].copy()
```

Tokenizing the text with normal word_tokenize

In [0]:

```
from nltk.tokenize import word_tokenize, sent_tokenize
import multiprocessing
p = multiprocessing.Pool()


dftok = (word_tokenize, df.ing)
```

BOW - Bag-of-words by using vectorizer

Due to fact that machines don't read text the same way humans do, the text must be converted into numeric structure. A common way to do this is through the use of Bag of Words. BoW splits words into pieces of text into tokens with no regard to word order. Afterwards the model is capable of counting the frequency a word is present in the text and assigns a weight proportional to this frequency.

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer

corpus = df.ing
vectorizer = CountVectorizer()
BOW = vectorizer.fit_transform(corpus)
```

Here we see that out of the 60,000 recipes and the ingredients used in them, we have almost 3000 different words

In [61]:

```
print(len(vectorizer.get_feature_names()))
```

```
2956
```

Next we will use LogisticRegression to see the weights of each word in the classes in our case recipes rated from 0-5 stars

In [0]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
```

In [0]:

```
vectorizer = CountVectorizer()
classifier = LogisticRegression()

model = Pipeline([
    ('vectorizer', vectorizer),
    ('classifier', classifier)
])
```

In [0]:

```
model.fit(df['ing'], df['rating'])
```

In [65]:

```
!pip -q install eli5
import eli5
eli5.show_weights(classifier, vec=vectorizer, top=30)
```

```
    |████████████████████████████████| 112kB 2.9MB/s
```

```
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:28: DeprecationWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signa
ect.getfullargspec()
  init_args = inspect.getargspec(class_.__init__)
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:28: DeprecationWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signa
ect.getfullargspec()
  init_args = inspect.getargspec(class_.__init__)
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:28: DeprecationWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signa
ect.getfullargspec()
  init_args = inspect.getargspec(class_.__init__)
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:28: DeprecationWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signa
ect.getfullargspec()
  init_args = inspect.getargspec(class_.__init__)
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:28: DeprecationWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signa
ect.getfullargspec()
  init_args = inspect.getargspec(class_.__init__)
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:28: DeprecationWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signa
ect.getfullargspec()
  init_args = inspect.getargspec(class_.__init__)
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:28: DeprecationWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signa
ect.getfullargspec()
  init_args = inspect.getargspec(class_.__init__)
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:28: DeprecationWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signa
ect.getfullargspec()
  init_args = inspect.getargspec(class_.__init__)
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:28: DeprecationWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signa
ect.getfullargspec()
  init_args = inspect.getargspec(class_.__init__)
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:28: DeprecationWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signa
ect.getfullargspec()
  init_args = inspect.getargspec(class_.__init__)
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:28: DeprecationWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signa
ect.getfullargspec()
  init_args = inspect.getargspec(class_.__init__)
/usr/local/lib/python3.6/dist-packages/eli5/base_utils.py:36: DeprecationWarning: The usage of `cmp` is deprecated and will be removed on or after 2021-
use `eq` and `order` instead.
  return attr.s(class_, these=these, init=False, slots=True, **attrs_kwargs)  # type: ignore
Using TensorFlow backend.
```

Out[65]:

y=0 top features		y=1 top features		y=2 top features		y=3 top features		y=4 top features		y=5 top features	
Weight?	Feature	Weight?	Feature	Weight?	Feature	Weight?	Feature	Weight?	Feature	Weight?	Feature
+1.793	paraffin	+1.680	wasabihorseradish	+1.535	genoa	+1.919	seitan	+1.646	gala	+1.652	papayas
+1.737	vienna	+1.575	kumquats	+1.443	bock	+1.679	rutabagas	+1.591	niblets	+1.464	cap
+1.735	caramelized	+1.490	four	+1.218	square	+1.361	brickle	+1.450	mesclun	+1.378	rag
+1.668	guavas	+1.384	kamut	+1.214	tutti	+1.342	strand	+1.406	pastina	+1.323	orientalflavor
+1.500	ditali	+1.331	truffle	+1.214	betterthancreamcheese	+1.339	hens	+1.372	feijoas	+1.321	bakers
+1.479	harp	+1.305	quicker	+1.209	garnet	+1.324	heath	+1.318	herring	+1.314	beefy
+1.419	simply	+1.243	post	+1.191	mirepoix	+1.301	honeynut	+1.311	pattypan	+1.296	szechwan
+1.404	eagle	+1.230	surejell	+1.156	crawfish	+1.209	bitters	+1.297	er	+1.295	drumstick
+1.398	kiss	+1.221	a1	+1.151	pheasant	+1.204	aubergine	+1.230	taro	+1.282	lifesavers
+1.381	pigeon	+1.216	dream	+1.146	dumpling	+1.201	skippy	+1.223	cups	+1.281	energ
+1.373	giardiniera	+1.207	quahogs	+1.140	fajita	+1.134	absolut	+1.111	coating	+1.278	breadstick
+1.358	margarita	+1.193	cremora	+1.139	ammonia	+1.120	fructose	+1.109	frankfurter	+1.217	field
+1.319	carnation	+1.192	spike	+1.135	amaranth	+1.116	pop	+1.106	passion	+1.203	waffle
+1.298	feet	+1.180	pancakes	+1.115	turnip	+1.100	glutinousrice	+1.101	farms	+1.200	supreme
+1.262	necks	+1.168	gizzards	+1.102	cavatelli	+1.096	calvados	+1.101	morningstar	+1.191	beaters
+1.252	meyers	+1.146	phyllo	+1.095	decorations	+1.090	flakey	+1.096	mutton	+1.189	lebanese
+1.241	satay	+1.138	cocacola	+1.086	buttery	+1.085	meyer	+1.089	halloumi	+1.161	teff
+1.231	boboli	+1.130	hair	*... 1116 more positive ...*		*... 1150 more positive ...*		+1.085	grapenuts	+1.155	wishbone
+1.228	cupcake	+1.126	digestive	*... 1811 more negative ...*		*... 1777 more negative ...*		+1.055	kidneys	+1.151	raclette
+1.211	monkfish	+1.079	diwip	-1.073	emerils	-1.073	pretzels	+1.050	sardines	+1.146	pomegranates
+1.195	liver	+1.076	oatabix	-1.088	singlecrust	-1.074	locatelli	*... 1164 more positive ...*		+1.136	longhorn
+1.178	spiced	*... 1110 more positive ...*		-1.107	segments	-1.090	avocados	*... 1763 more negative ...*		*... 1184 more positive ...*	
+1.170	jars	*... 1817 more negative ...*		-1.126	pot	-1.112	grilled	-1.058	seafood	*... 1743 more negative ...*	
... 1291 more positive ...		-1.086	bagel	-1.149	neufchatel	-1.129	sprout	-1.072	brisket	-1.132	fivespice
... 1636 more negative ...		-1.090	spiced	-1.157	cellophane	-1.144	clarified	-1.073	simply	-1.140	prune
-1.228	turnips	-1.116	tiny	-1.163	ravioli	-1.159	glace	-1.118	swede	-1.154	coke
-1.242	livers	-1.170	10inch	-1.181	wedges	-1.162	paraffin	-1.125	tops	-1.160	snapper
-1.303	guinness	-1.229	grands	-1.203	asafoetida	-1.259	simply	-1.138	oysters	-1.160	szechuan
-1.354	gingersnap	-1.270	tfee	-1.300	savoy	-1.279	arugula	-1.303	hellmanns	-1.169	pepsi
-1.357	sole	-1.338	piece	-1.318	mexicanstyle	-1.290	muenster	-1.311	bonein	-1.185	sultana
-1.357	grenadine	-1.363	seasonings	-1.473	limeade	-1.333	husks	-1.420	plantains	-1.329	oriental
-1.649	<BIAS>	-1.471	<BIAS>	-1.688	<BIAS>	-1.715	<BIAS>	-1.507	<BIAS>	-1.666	<BIAS>

TF-IDF - Term Frequency: Inverse Document Frequency

TF-IDF(Term frequency – inverse document frequency) is another method through which we can judge the topic of a text through the words it contains. Compared to BoW, TF-IDF measures the relevance of the words, not their frequency. TF is the result of using BoW while IDF systematically discounts the words that appear too often. As a result we will be left with only the most frequent and distinctive words as markers

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
recipes_bow_tf = vectorizer.fit_transform(corpus)
```

LSA - Latent Semantic Analysis (BOW + TF-IDF) Dimensional reduction

Latent Semantic Analysis(LSA) is a technique which retrieves information that is then analyzed in order to identify existing patterns in the text and the relationship between them.

In [0]:

```
from sklearn.decomposition import TruncatedSVD
from sklearn.random_projection import sparse_random_matrix

recipes_bow_tf = sparse_random_matrix(250, 250, density=0.01, random_state=42)
svd = TruncatedSVD(n_components=90, n_iter=7, random_state=42)
X_lsa = svd.fit_transform(recipes_bow_tf)
```

we managed to reduce dimensionality

In [68]:

```
len(X_lsa)
```

Out[68]:

```
250
```

LDA - model

LDA is a topic model and it is used in order to find the abstract topics that might occur in a document. The way it works is that it builds a topic per document model and words per topic model, which are modeled as Dirichlet distributions.

We do some neccessary steps in order to make LDA model learn our data

In [0]:

```
!pip install pyLDAvis
```

In [0]:

```
import pyLDAvis.gensim
%matplotlib inline
pyLDAvis.enable_notebook()
```

In [0]:

```
import spacy
nlp = spacy.load("en")
```

In [0]:

```
from gensim.corpora.dictionary import Dictionary
dictionary = Dictionary(tokens)
```

In [0]:

```
corpus = [dictionary.doc2bow(doc) for doc in tokens]
```

Here the LDA model is learning our corpus

In [0]:

```
from gensim.models import LdaMulticore
lda_model = LdaMulticore(corpus, id2word=dictionary,  num_topics=10, workers = 2, passes=10)
```

Here we see something like clusters - we see a connections in these words

In [75]:

```
lda_model.print_topics(-1)
```

Out[75]:

```
[(0,
  '0.074*"cheese" + 0.071*"corn" + 0.045*"cream" + 0.030*"beans" + 0.030*"cheddar" + 0.029*"onion" + 0.026*"sour" + 0.024*"rice" + 0.023*"tortillas" + 0
,
 (1,
  '0.095*"flour" + 0.093*"sugar" + 0.074*"baking" + 0.069*"salt" + 0.054*"butter" + 0.050*"powder" + 0.042*"vanilla" + 0.038*"eggs" + 0.036*"milk" + 0.0
e"'),
 (2,
  '0.064*"pepper" + 0.064*"fresh" + 0.060*"olive" + 0.059*"oil" + 0.051*"salt" + 0.046*"garlic" + 0.033*"dried" + 0.027*"parsley" + 0.024*"cloves" + 0.0
,
 (3,
  '0.080*"ground" + 0.073*"cinnamon" + 0.059*"sugar" + 0.044*"nutmeg" + 0.026*"juice" + 0.024*"lemon" + 0.024*"apple" + 0.022*"pumpkin" + 0.021*"water"
er"'),
 (4,
  '0.061*"sauce" + 0.041*"oil" + 0.040*"juice" + 0.037*"soy" + 0.035*"sugar" + 0.034*"garlic" + 0.032*"pepper" + 0.032*"vinegar" + 0.031*"chicken" + 0.0
 (5,
  '0.060*"pepper" + 0.053*"salt" + 0.049*"cheese" + 0.044*"cream" + 0.041*"onion" + 0.035*"butter" + 0.034*"milk" + 0.029*"soup" + 0.026*"cheddar" + 0.0
),
 (6,
  '0.071*"pepper" + 0.051*"ground" + 0.049*"onion" + 0.048*"salt" + 0.047*"powder" + 0.047*"garlic" + 0.044*"beef" + 0.041*"sauce" + 0.025*"tomato" + 0.
,
 (7,
  '0.064*"cream" + 0.053*"sugar" + 0.042*"vanilla" + 0.033*"butter" + 0.032*"chocolate" + 0.032*"milk" + 0.025*"cheese" + 0.024*"eggs" + 0.022*"extract"
"'),
 (8,
  '0.040*"oil" + 0.038*"powder" + 0.036*"cooking" + 0.031*"salt" + 0.029*"plain" + 0.029*"yogurt" + 0.028*"rice" + 0.027*"ground" + 0.024*"spray" + 0.02
 (9,
  '0.104*"cheese" + 0.067*"chicken" + 0.038*"boneless" + 0.034*"parmesan" + 0.033*"skinless" + 0.032*"garlic" + 0.031*"breasts" + 0.026*"pepper" + 0.024
021*"italian"')]
```

In [0]:

```
lda_display = pyLDAvis.gensim.prepare(lda_model, corpus, dictionary, sort_topics=False)
```

And here we can nicely see each topic with cluster, some of them are obviously baking recipes, some for lemonades, some for grilling and some are mix

In [77]:

```
pyLDAvis.display(lda_display)
```

Out[77]:

Selected Topic: 0 Previous Topic Next Topic Clear Topic

Slide to adjust relevance metric:

Selected Topic: 0 Previous Topic Next Topic Clear Topic

Slide to adjust relevance metric: (2)
λ = 1

Unsupervised Machine Learning

Again we do some steps neccessary in order to make the Clusters. Like fitting the model to corpus, making LSI model and using gensim similarities

In [0]:

```
from gensim.models.tfidfmodel import TfidfModel
```

In [0]:

```
tfidf = TfidfModel(corpus)
```

In [0]:

```
tfidf_corpus = tfidf[corpus]
```

In [0]:

```
# Just like before, we import the model
from gensim.models.lsimodel import LsiModel

lsi = LsiModel(tfidf_corpus, num_topics=100)
```

In [0]:

```
lsi_corpus = lsi[tfidf_corpus]
```

In [0]:

```
from gensim.similarities import MatrixSimilarity

# Create the document-topic-matrix
document_topic_matrix = MatrixSimilarity(lsi_corpus)
document_topic_matrix_ix = document_topic_matrix.index
```

In [0]:

```
sims = document_topic_matrix[lsi_corpus[0]]
sims = sorted(enumerate(sims), key=lambda item: -item[1])
print(sims)
```

We use this to make less neighbors for clustering

In [0]:

```
import umap
embeddings = umap.UMAP(n_neighbors=15, metric='cosine').fit_transform(document_topic_matrix_ix)
```

In [86]:

```
from sklearn.cluster import KMeans
clusterer = KMeans(n_clusters = 10)
clusterer.fit(document_topic_matrix_ix)
```

Out[86]:

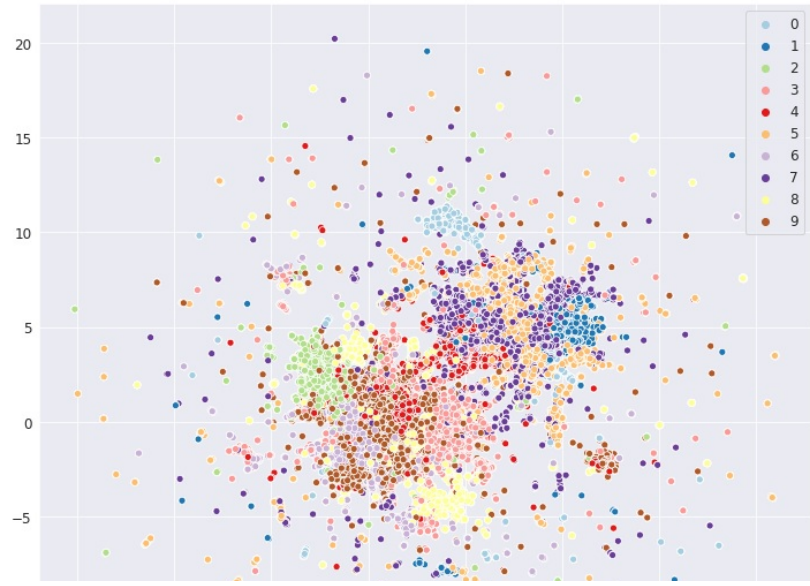
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=10, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

Here we plot the clusters into one scatterplot

In [87]:

```
# Plotting things
import seaborn as sns
sns.set_style("darkgrid")

plt.rcParams.update({'font.size': 12})
plt.figure(figsize=(12,12))
g = sns.scatterplot(*embeddings.T,
                    #reduced[:,0],reduced[:,1],
                   hue=clusterer.labels_,
                    palette="Paired",
                   legend='full')
```



In [0]:

```
# Let's explore the clusters, that should actually correlate with topics found by LDA
df['cluster'] = clusterer.labels_
```

This should be the recipes and their ingrediences that were found in different topics, sorted by LDA analysis

In [89]:

```
df[df['cluster'] == 2]['ing']
```

Out[89]:

```
2        butter olive oil vidalia onion garlic cloves  ...
23       lean pork garlic cloves cornstarch sugar soy s...
39        beans flank steak soy sauce cornstarch chines...
70       garlic cloves soy sauce water honey vegetable ...
74       chicken wings margarine soy sauce  sugar water...
                               ...
59957    cornstarch worcestershire sauce top sirloin st...
59974    pork chops onion celery garlic ginger salt pep...
59975    whole chickens teriyaki sauce chinese five spi...
59980    eggs chinese vegetables oil  onions beef broth...
59993      pork spareribs salt pepper barbecue sauce onion
Name: ing, Length: 3990, dtype: object
```

Train a word-embedding model of your choice (Word2Vec, GloVe or Fasttext)

This module implements the word2vec family of algorithms, using highly optimized C routines, data streaming and Pythonic interfaces.

In [0]:

```
from gensim.models import Word2Vec
import logging
from gensim.models.phrases import Phrases, Phraser
phrases = Phrases(tokens, min_count=5, threshold=10)
bigram = Phraser(phrases)
tokens_phrases = tokens.map(lambda t: bigram[t])
model_paprika = Word2Vec(tokens_phrases, size=100, window=5, min_count=5, workers=4, iter=3)
```

Here we make our model paprika and we try to find out which are the most similar words from this text with our chosen word paprika - the results speak for themselves :-P

In [91]:

```
model_paprika.wv.most_similar('paprika')
```

```
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion of the second argument of issubdtype from `int` to `np.signedin
ecated. In future, it will be treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

Out[91]:

```
[('cayenne', 0.7914016246795654),
 ('pepper', 0.6527678370475769),
 ('cumin', 0.6453433036804199),
 ('chili', 0.6301818490028381),
 ('ancho_chile', 0.6186747550964355),
 ('seed', 0.617232620716095),
 ('coarse', 0.6162411570549011),
 ('coriander', 0.6096932888031006),
 ('oregano', 0.6072298288345337),
 ('powder', 0.606373131275177)]
```

Machine Learning - Supervised

Model preparation

In [0]:

```
# Defining the dependent variable. What are we looking for?
y = df['rating']


# Importing the train-test split function.
from sklearn.model_selection import train_test_split

# Making the split. We keep 75% for training and 25% for test.
X_train, X_test, y_train, y_test = train_test_split(BOW, y, test_size = 0.25, random_state=42)
```

In [97]:

```
print( X_train.shape, y_train.shape)
print( X_test.shape, y_test.shape)
```

```
(45000, 2956) (45000,)
(15000, 2956) (15000,)
```

Logistic Regression

In [0]:

```
from sklearn.metrics import classification_report

from sklearn.model_selection import cross_val_score
```

In [99]:

In [99]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

scores = cross_val_score(model, X_train, y_train, cv = 5)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(model.score(X_test, y_test))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
ence this warning.
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:469: FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Speci
lass option to silence this warning.
  "this warning.", FutureWarning)
```

```
0.21
```

Logistic regression gives us only 21% true prediction for this model, which is very, very low. We will contemplate about this result in stakeholders report and in exam

In [100]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.20      0.20      0.20      2452
           1       0.24      0.31      0.27      2545
           2       0.20      0.16      0.18      2564
           3       0.19      0.18      0.19      2472
           4       0.21      0.19      0.20      2539
           5       0.21      0.22      0.21      2428

    accuracy                           0.21     15000
   macro avg       0.21      0.21      0.21     15000
weighted avg       0.21      0.21      0.21     15000
```

Preparing to print a bit fancier confusion matrix

In [0]:

```
!pip install -U mlxtend
# Import the confusion matrix plotter module
from mlxtend.plotting import plot_confusion_matrix

# We will also import sklearns confusion matrix module (makes it easy to produce a confusion matrix)
# It's actually just a cross-tab of predicted vs. real values
from sklearn.metrics import confusion_matrix
```

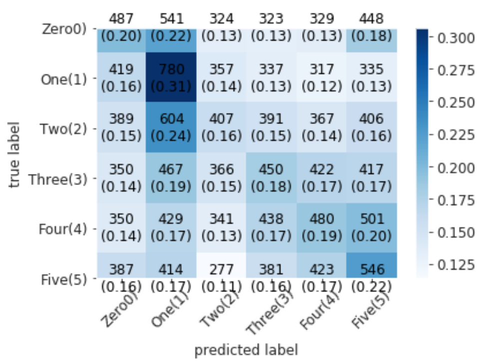
The number is the number of stars the recipe got

In [103]:

```
classNames = ["Zero0)","One(1)","Two(2)", "Three(3)", "Four(4)", "Five(5)"]
plot_confusion_matrix(conf_mat=cm,
                                colorbar=True,
                                show_absolute=True,
                                show_normed=True,
                                class_names = classNames)
```

Out[103]:

```
(<Figure size 432x288 with 2 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7facf94a6160>)
```



We can see that the prediction for each star are very low, explanation will be in stakeholders report

MultinomialNB

preparation

Here we will try a different Model, maybe it will give us a better conclusion

In [0]:

```
y = df["rating"]
X_train, X_test, y_train, y_test = train_test_split(df["ing"], y, test_size=0.33, random_state=53)
count_vectorizer = CountVectorizer()
count_train = count_vectorizer.fit_transform(X_train)
count_test = count_vectorizer.transform(X_test)
```

In [105]:

```
# Import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)
```

```python
# Transform the training data: tfidf_train
tfidf_train = tfidf_vectorizer.fit_transform(X_train)

# Transform the test data: tfidf_test
tfidf_test = tfidf_vectorizer.transform(X_test)

# Print the first 5 vectors of the tfidf training data
print(tfidf_train.A[:5])
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

In [106]:

```python
# Following some of the steps from NLP lectures and colabs provided

# Create the CountVectorizer DataFrame: count_df
count_df = pd.DataFrame(count_train.A, columns=count_vectorizer.get_feature_names())

# Create the TfidfVectorizer DataFrame: tfidf_df
tfidf_df = pd.DataFrame(tfidf_train.A, columns=tfidf_vectorizer.get_feature_names())

# Print the head of count_df
print(count_df.head())

# Print the head of tfidf_df
print(tfidf_df.head())

# Calculate the difference in columns: difference
difference = set(count_df.columns) - set(tfidf_df.columns)
print(difference)

# Check whether the DataFrames are equal
print(count_df.equals(tfidf_df))
```

```
   10  10inch  10minute  10x  ...  ziploc  ziti  zucchini  zwieback
0   0       0         0    0  ...       0     0         0         0
1   0       0         0    0  ...       0     0         0         0
2   0       0         0    0  ...       0     0         0         0
3   0       0         0    0  ...       0     0         0         0
4   0       0         0    0  ...       0     0         0         0

[5 rows x 2748 columns]
    10  10inch  10minute  10x  ...  ziploc  ziti  zucchini  zwieback
0  0.0     0.0       0.0  0.0  ...     0.0   0.0       0.0       0.0
1  0.0     0.0       0.0  0.0  ...     0.0   0.0       0.0       0.0
2  0.0     0.0       0.0  0.0  ...     0.0   0.0       0.0       0.0
3  0.0     0.0       0.0  0.0  ...     0.0   0.0       0.0       0.0
4  0.0     0.0       0.0  0.0  ...     0.0   0.0       0.0       0.0

[5 rows x 2712 columns]
{'back', 'without', 'no', 'all', 'after', 'with', 'one', 'ie', 'on', 'thin', 'and', 'con', 'made', 'the', 'any', 'fire', 'mill', 'in', 'side', 'whole'
', 'top', 'cant', 'bottom', 'other', 'very', 'for', 'not', 'its', 'thick', 'up', 'your', 'from', 'de', 'four'}
False
```

In [107]:

```python
# Here we will train our MultinomialNB model
# Import the necessary modules
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

# Instantiate a Multinomial Naive Bayes classifier: nb_classifier
nb_classifier = MultinomialNB()

# Fit the classifier to the training data
nb_classifier.fit(count_train, y_train)

# Create the predicted tags: pred
pred = nb_classifier.predict(count_test)

# Calculate the accuracy score: score
score = metrics.accuracy_score(y_test, pred)
print(score)
```

```
0.21378787878787878
```

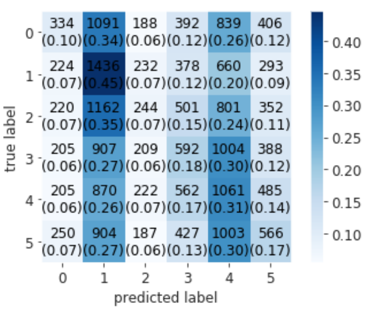
Again we see, that the score is almost the same as Logistic Regression, its very low.

In [108]:

```python
confmatrix = confusion_matrix(y_test,pred)
plot_confusion_matrix(conf_mat=confmatrix,
colorbar=True,
show_absolute=True,
show_normed=True)
```

Out[108]:

```
(<Figure size 432x288 with 2 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7facedda8c18>)
```



And again we can see each of the label in confusion matrix, and the results are very low as you can see