Data Processing Algorithms: Clustering

Fatema Bhayani
6-22-2020

Table of Contents

1 Introduction 2
  1.1 An Introduction to Label Propagation 2
2 The Label Propagation Algorithm 3
  2.1 The Input 3
  2.2 The Algorithm 3
  2.3 The Output 4
3 Tutorial and Examples 5
  3.1 Download Data from Twitter 5
  3.2 Adjacency Array 5
  3.3 Run the label propagation algorithm 6
  3.4 Example of a Printout 6

1 Introduction

The algorithms developed in this project can be categorized into the following categories:

1. Twitter Download Algorithms: These algorithms download social network data from Twitter using the Twitter API.

2. Tweet Processing Algorithms: These algorithms process the tweets by removing stop words, hashtags, tweets in foreign languages etc. Moreover, these algorithms also analyze the word frequency of the user's tweets and determine what communities they belong to.

3. Clustering Algorithms: These algorithms cluster users based on their similarity and their connections in the social network.

This report will focus on Clustering Algorithms. We have implemented the label propagation clustering algorithm. More clustering algorithms will be added later.

1.1 An Introduction to Label Propagation

Label propagation is a semi-supervised machine learning algorithm that assigns labels to previously unlabeled data points.

Each node is initialized with a unique label and at every iteration of the algorithm, each node adopts a label that a maximum number of its neighbors have, with ties broken uniformly randomly. As the labels propagate through the network in this manner, densely connected groups of nodes form a consensus on their labels.

At the end of the algorithm, nodes having the same labels are grouped together as communities. The number of communities and their sizes are not known a priori and are only determined at the end of the algorithm.

2 The Label Propagation Algorithm

Each node is initialized with a unique label and at every iteration of the algorithm, each node adopts a label that a maximum number of its neighbors have, with ties broken uniformly randomly. As the labels propagate through the network in this manner, densely connected groups of nodes form a consensus on their labels. At the end of the algorithm, nodes having the same labels are grouped together as communities. The number of communities and their sizes are not known a priori and are only determined at the end of the algorithm.

2.1 The Input

They initialize every node x with unique label $L_x(t = 0)$. We need n nodes and the connectivity graph that describes which nodes are connected with each other. The graph needs to be undirected and connected for the algorithm to work.

2.2 The Algorithm

Suppose that a node x has neighbors $x_1, x_2, \dots, x_k$ and that each neighbor carries a label denoting the community to which they belong to. Then x determines its community based on the labels of its neighbors. They assume that each node in the network chooses to join the community to which the maximum number of its neighbors belong to, with ties broken uniformly randomly.

They use asynchronous updating, where node x at iteration t updates its label based on the label of its neighbors at iteration t (as opposed to updating based on the label of its neighbors at iteration t-1). As a result, $L_x(t) = f\left(L_{x_{i1}}(t), \dots, L_{x_{im}}(t), L_{x_{i(m+1)}}(t-1), \dots, L_{x_{ik}}(t-1)\right)$where $x_{i1}, \dots, x_{im}$ are neighbors of x that have already been updated in the current iteration (iteration t), while $x_{i(m+1)}, \dots, x_{ik}$ are neighbours of x that have not yet been updated in the current iteration. The order in which all the n nodes in the network are updated at each iteration is chosen randomly.

They perform the iterative process until every node in the network has a label to which the maximum number of its neighbors belong to. If $L_1, \dots, L_p$ are the labels currently active in the network and $d_i^{L_j}$ is the number of neighbours of node i with label L_j, then the algorithm stops if every node i has label L_m, then $d_i^{L_m} > d_i^{L_j}, \forall j$

In summary, the label propagation algorithm takes the following steps:

1. Initialize the labels at all the nodes in the network. Every node x has a unique label $L_x(t = 0)$
2. Set $t = 1$

3. Arrange the nodes in the network in a random order and set it to X

4. For each $x \in X$ chosen in that specific order, let $L_x(t) = f\left(L_{x_{i1}}(t), \ldots, L_{x_{im}}(t), L_{x_{i(m+1)}}(t-1), \ldots, L_{x_{ik}}(t-1)\right)$. f returns the label occurring with the highest frequency among neighbors and ties are broken uniformly randomly.

5. If every node has a label that the maximum number of their neighbors have, then stop the algorithm. Else, set $t = t + 1$ and go to Step (3).

2.3 The Output

When the algorithm stops, we have P communities with labels $L_1, \ldots, L_p$(one label for each community) and each node i has a label $L_1, \ldots, L_p$ (the community in which node i belongs to also referred to as cluster membership). This data will be in form of a dictionary where the keys are the labels of the clusters/communities and the values are the nodes that belong to that cluster.

3 Tutorial and Examples

3.1 Download Data from Twitter

To run the label propagation algorithm, we first need to download data from Twitter. To run the label propagation algorithm, we need to generate a connectivity graph by 1) get all friends of the users, these are the nodes of the graph 2) for these nodes, if A is a friend of B and visa versa, extend an edge between both of them.

Since we want to download a list of a user's friends, we could first modify the output datastore section for our datastore config file.

```
output-datastores:

  output-ds1:

    project-name: "TwitterDownload"

    datastore-name: "TwitterDownload"

    collection-name: "Friends"

    type: "MongoDB"

    location: "mongodb://localhost:27018"
```

Then, we instantiate a Downloader object and call the respective method to download.

```
twitterDownloader = TwitterFriendsDownloader("ds-init-config.yaml")

twitterDownloader.get_friends_by_screen_name(user_name, num_friends_to_get)
```

3.2 Adjacency Array

Now we generate the adjacency array, which is a collection of strings representing the connections in the graph. The first item in the string gives you the root node and the remaining items in the string represent the set of direct neighbours (vertices) of the graph. There are n strings (one for each node, if there are n nodes) which form the adjacency array.

Example: If there are 5 users in the graph, the database looks like:

{name: 1, adj_list: ["1 2 3", "2 3 5", "3 4", "4 5"]}

The set of strings (which represent the adjacency array) are the value, while the user is the key to the database. This data is stored in MongoDB.

Let ‘friends’ be a list of friends of a user. Let ‘graph’ be an empty graph.

```
for agent in friends:
    graph.add_node(agent)
list_of_nodes = list(graph.nodes)
for i in range(len(list_of_nodes)):
    for range(i, len(list_of_nodes)):
        if list_of_nodes[i] in friends[list_of_nodes[j]] and list_of_nodes[j] in friends[list_of_nodes[i]:
            graph.add_edge(list_of_nodes[i], list_of_nodes[j])
```

To generate the adjacency graph, run generate_from_user()

```
lg = LocalGraph()

lg.generate_from_user("david_madras")
```

3.3 Run the label propagation algorithm

Now that we have the adjacency graph in the form of an array, we can run the label propagation algorithm.

To run the label propagation algorithm on the adjacency graph, run get_clusters()

```
clusters = lg.get_clusters(method='lprop')
```

The output of the label propagation algorithm is represented as a dictionary (a dictionary holds a collection of key-value pairs where the values can be accessed through the keys), where the keys are the community number (doesn’t carry any intrinsic meaning), while the values are the list of users that belong to the community. This is also stored in MongoDB in the following structure. An example is given below in the printout. This format of the printout will be improved.

3.4 Example of a Printout

```
{
 "0": [
  "david_madras",
  "dirk_hovy",
```

"david_sontag",

"Smerity",

"yoavgo",

"RandomlyWalking",

"SussilloDavid",

"aivillage_dc",

"aerinykim",

"tyrell_turing",

"melosare",

"marthadarby",

"MrDanielMax",

"annakoop",

"angelamczhou",

"NicolasPapernot",

"goodfellow_ian",

"rctatman",

"1vnzh",

"sarahcolq",

"moyix",

"jefrankle",

"A_Aspuru_Guzik",

"EthanFetaya",

"RonSoupprune",

"nickfrosst",

"kongaloosh",

"rajiinio",

"hardmaru",

"ShalitUri",

"sina_lana",

"james_r_lucas",

"mtesfald",

"liannapisani",

"AidanNGomez",

"jackclarkSF",

"zacharylipton",

"fhuszar",

"PariAIML",

"jh_jacobsen",

"roydanroy",

"j_asminewang",

"leilanigilpin",

"catherineols",

"AkiyoshiKitaoka",

"korymath",

"mrtz",

"leeclemnet",

"_asubbaswamy",

"mark_riedl",

"carrasqu",

"AmirRosenfeld",

"Miles_Brundage",

"alex_y_gao",

"__sarah_anne__",

"jessebett",

"umangsbhatt",

"julius_adebayo",

"RogerGrosse",

"tdietterich",

"MarzyehGhassemi",

"berty38",

"ZOMGZinc",

"judyhshen",

"KLdivergence",

"blancamiller_",

"osageev",

"SingularMattrix",

"ermgrant",

"alexdamour",

"adversariel",

"haldaume3",

"VectorInst",

"wgrathwohl",

"realnamepolicy",

"chloepouprom",

"kswersk",

```
  "SoloGen",
  "patrickcorneil",
  "MattBobkin",
  "coolboi95",
  "tw_killian",
  "beenwrekt",
  "databoydg",
  "james_c_atwood",
  "variationalBAE",
  "orionbuske",
  "math_rachel",
  "SirrahChan",
  "nicohowe",
  "suchisaria",
  "cobyviner",
  "mmitchell_ai",
  "raebett",
  "Lv100Swagneton",
  "mattschleg",
  "randal_olson",
  "johnmyleswhite",
  "iamstripedhorse",
  "thesasho"
],
"1": [
```

"david_madras",

"octonion",

"julia_azari",

"edwardhkennedy",

"ashleyfeinberg",

"SarahTaber_bww",

"NBA_Math",

"ithrow88",

"MichaelSLinden",

"GeoffreySupran",

"jen_keesmaat",

"BlakeMurphyODC",

"AlecMacGillis",

"jessesingal",

"bencasselman",

"TerribleMaps",

"m_layton",

"NateSilver538",

"skepticalsports",

"Nate_Cohn",

"JohnHolbein1",

"leah_boustan",

"linamkhan",

"ModeledBehavior",

"AndrewDBailey",

"3r1nG",

"SethAMandel",

"poverty_action",

"LisPower1",

"OldHossRadbourn",

"TimHarford",

"MattGertz",

"juliagalef",

"matthewstoller",

"ForecasterEnten",

"HeerJeet",

"maya_sen",

"OsitaNwanevu",

"ddale8",

"madbev14",

"bruce_arthur",

"Yair_Rosenberg",

"curaffairs",

"egavactip",

"gelliottmorris",

"KT_So_It_Goes",

"Noahpinion",

"langston_poems",

"dmorey",

"Undercoverhist",

"LorenRaeDeJ",

"dick_nixon",

"ZachLowe_NBA",

"BrendanNyhan",

"ClareMalone",

"TorontoStar",

"causalinf",

"timoreilly",

"ollie",

"Susan_Athey",

"bencfalk",

"ContraPoints",

"whatwouldDOOdo",

"byrdinator",

"DaraKaye",

"RespectableLaw",

"SeanMcElwee",

"GraphicMatt",

"elyratner",

"lawfareblog",

"peltzmadeline",

"SolomonMg",

"theintercept",

"ggreenwald",

"willsommer",

```
  "Richard_Florida",
  "ne0liberal",
  "ekoreen",
  "briebriejoy",
  "DoctorVive",
  "SeanTrende",
  "aprilaser",
  "theneedledrop",
  "mkoplow",
  "ErikLoomis",
  "WillOremus",
  "mattyglesias",
  "qjurecic",
  "alexstamos"
],
"2": [
  "david_madras",
  "SatterthwaiteML",
  "ghadfield",
  "elizabeth_joh",
  "ndiakopoulos",
  "supergovernance",
  "benzevgreen",
  "ceh0",
  "EvanSelinger",
"natematias",
"harvard_data",
"megberetta",
"mikekimbackward",
"jennwvaughan",
"lawlkat",
"propublica",
"Aaroth",
"wsisaac",
"geomblog",
"BKCHarvard",
"AINowInstitute",
"ShannonVallor",
"SarahJamieLewis",
"mikarv",
"nathanjurgenson",
"superwuster",
"FrankPasquale",
"GretchenAMcC",
"JuliaAngwin",
"grimmelm",
"MarkupReal",
"jamiemorgenste1",
"packer_ben",
"BrendaKLeong",
  "team_markup",
  "TrooperSanders",
  "alexhanna",
  "mariadearteaga",
  "timhwang",
  "hannawallach",
  "jasonmillar",
  "CausalKathy",
  "katecrawford",
  "mathbabedotorg",
  "random_walker",
  "maria__cuellar",
  "j2bryson",
  "zeynep",
  "zittrain",
  "_galyo",
  "mbogen",
  "elanazeide",
  "ncasenmare"
],
"3": [
  "david_madras",
  "LStanley24",
  "IanGormely",
  "chancetherapper",
  "dhackett1565",
  "pitchfork",
  "joey_doubleyou",
  "AnnaJaneSmith4",
  "JasonIsbell",
  "william_lou",
  "exclaimdotca",
  "vincestaples",
  "carolinespence_",
  "wtevs"
],
"4": [
  "david_madras",
  "notsapphoe",
  "slydesilva",
  "risk4bisc",
  "JaggamanJames",
  "TaiUdovicic",
  "jasserole",
  "HelloHops",
  "mchlglry"
],
"5": [
  "david_madras",
  "EgSophie",
  "tabletmag",
  "haaretzcom",
  "jstreetdotorg",
  "rabbijilljacobs"
],
"6": [
  "david_madras",
  "dramemariama20",
  "Mo_Fad3l",
  "payaicha15",
  "ltondji",
  "panfordkobby"
],
"7": [
  "david_madras",
  "_HeatherE",
  "CANADALAND",
  "ddebow",
  "JesseBrown"
],
"8": [
  "david_madras",
  "CBCNews",
  "dsouzamichaelc"
],
```

"9": [
  "david_madras",
  "A_single_bear"
],
"10": [
  "david_madras",
  "HMSHCP"
],
"11": [
  "david_madras",
  "rxgau"
],
"12": [
  "david_madras",
  "forexposure_txt"
],
"13": [
  "david_madras",
  "ewarren"
],
"14": [
  "david_madras",
  "Kurt_Vonnegut"
],
"15": [

```
  "david_madras",
  "computerfact"
],
"16": [
  "david_madras",
  "vanillatary"
],
"17": [
  "david_madras",
  "abbagoldbot"
],
"18": [
  "david_madras",
  "Limericking"
],
"19": [
  "david_madras",
  "ProbCausation"
],
"20": [
  "david_madras",
  "fermatslibrary"
],
"21": [
  "david_madras",
    "TedOnPrivacy"
  ],
  "22": [
    "david_madras",
    "MicroSFF"
  ],
  "23": [
    "david_madras",
    "GeoffRockwell"
  ],
  "24": [
    "david_madras",
    "dril"
  ],
  "25": [
    "david_madras",
    "ProBirdRights"
  ],
  "26": [
    "david_madras",
    "aliceysu"
  ]
}
```