Data Mining

Prepared for CS527

By: Josh Levine, Pranav Shivkumar, Shounak Rangwala, Pratik Mistry, Swapnil Kamate, Vikhyat Dhamija

Introduction to assignment

- To select the targets and top 10 probes from the “Genome” wide data set and to transfer the data to RDBMS in a new table
- To create models for KNN classification and K-means clustering on above data
- To create Recommendation System based on Instacart Dataset

Task 1: Upload Data from Wide Dataset

Query Result : Select * from Genome1;

	Source_Name	Value1	Value2	Value3	Value4	Value5	Value6	Value7	Value8	Value9	Value10
1	GSM1366711 1	19.00065276	18.93236743	18.89469309	18.87350665	18.857951	18.85380342	18.75718509	18.71939662	18.68931234	18.63645652
2	GSM1366712 1	19.00065276	18.94119796	18.91507311	18.89469309	18.86706526	18.84611278	18.80104894	18.77539643	18.68931234	18.66361169
3	GSM1366713 1	19.00065276	18.97202237	18.93533363	18.92840902	18.90133189	18.87054287	18.84259104	18.81775304	18.77539643	18.71939662
4	GSM1366714 1	19.00065276	18.97202237	18.95956353	18.94683931	18.92840902	18.90626607	18.89146153	18.84611278	18.81775304	18.75718509
5	GSM1366715 1	18.92214082	18.87054287	18.86411734	18.80104894	18.7908401	18.77539643	18.71939662	18.68931234	18.66361169	18.60712066
6	GSM1366716 1	18.857951	18.83871954	18.83456008	18.82348674	18.81196729	18.7908401	18.75718509	18.71939662	18.68931234	18.66361169
7	GSM1366717 1	18.92214082	18.90375617	18.82885387	18.81196729	18.7908401	18.75718509	18.74271124	18.68931234	18.66361169	18.63645652
8	GSM1366718 1	19.00065276	18.81775304	18.7908401	18.77539643	18.71939662	18.68931234	18.66361169	18.63645652	18.60712066	18.56796384
9	GSM1366719 1	19.00065276	18.97202237	18.92840902	18.87350665	18.86411734	18.84259104	18.83456008	18.75718509	18.66361169	18.63645652
10	GSM1366720 1	19.00065276	18.97202237	18.95956353	18.94683931	18.89814043	18.83871954	18.75718509	18.74271124	18.71939662	18.68931234
11	GSM1366721 1	19.00065276	18.97202237	18.95956353	18.92528899	18.86411734	18.85380342	18.84259104	18.77539643	18.71939662	18.66361169
12	GSM1366722 1	19.00065276	18.97202237	18.92214082	18.87054287	18.85380342	18.84259104	18.77539643	18.71939662	18.68931234	18.63645652
13	GSM1366723 1	19.00065276	18.97202237	18.89146153	18.88095024	18.86411734	18.85380342	18.81775304	18.81196729	18.68931234	18.63645652
14	GSM1366724 1	19.00065276	18.94119796	18.90375617	18.89469309	18.88430478	18.86411734	18.86076773	18.68931234	18.66361169	18.63645652
15	GSM1366725 1	18.90941486	18.86706526	18.84611278	18.83871954	18.82348674	18.81775304	18.77539643	18.68931234	18.66361169	18.63645652
16	GSM1366726 1	19.00065276	18.94683931	18.91507311	18.90133189	18.86706526	18.86076773	18.75718509	18.68931234	18.66361169	18.63645652
17	GSM1366727 1	18.93533363	18.91286356	18.86076773	18.82348674	18.80104894	18.75718509	18.74271124	18.71939662	18.66361169	18.63645652
18	GSM1366728 1	19.00065276	18.88430478	18.86411734	18.84974133	18.75718509	18.74271124	18.71939662	18.68931234	18.66361169	18.60712066
19	GSM1366729 1	18.93533363	18.88095024	18.87054287	18.81775304	18.7908401	18.77539643	18.75718509	18.71939662	18.68931234	18.63645652
20	GSM1366730 1	19.00065276	18.97202237	18.95956353	18.94683931	18.89469309	18.88430478	18.8768442	18.857951	18.7908401	18.71939662
21	GSM1366731 1	19.00065276	18.92840902	18.84259104	18.83456008	18.82885387	18.80104894	18.75718509	18.68931234	18.66361169	18.63645652
22	GSM1366732 1	19.00065276	18.97202237	18.95956353	18.91286356	18.89146153	18.84974133	18.84611278	18.77539643	18.75718509	18.74271124
23	GSM1366733 1	19.00065276	18.97202237	18.89146153	18.84974133	18.84259104	18.82348674	18.75718509	18.74271124	18.71939662	18.63645652
24	GSM1366734 1	19.00065276	18.84611278	18.83456008	18.82348674	18.80104894	18.75718509	18.74271124	18.68931234	18.66361169	18.56796384
25	GSM1366735 1	19.00065276	18.97202237	18.94683931	18.94119796	18.90626607	18.84974133	18.7908401	18.71939662	18.68931234	18.66361169
26	GSM1366736 1	18.85380342	18.82885387	18.81775304	18.81196729	18.80104894	18.77539643	18.71939662	18.68931234	18.66361169	18.63645652
27	GSM1366737 1	18.94119796	18.86411734	18.81196729	18.77539643	18.75718509	18.74271124	18.71939662	18.68931234	18.66361169	18.60712066
28	GSM1366738 1	19.00065276	18.97202237	18.94683931	18.91850639	18.90375617	18.82885387	18.81775304	18.75718509	18.71939662	18.66361169
29	GSM1366739 1	18.97202237	18.95956353	18.85380342	18.82885387	18.81196729	18.75718509	18.71939662	18.68931234	18.66361169	18.63645652
30	GSM1366740 1	18.94119796	18.92840902	18.89469309	18.88095024	18.87054287	18.84974133	18.71939662	18.66361169	18.60712066	18.56796384
31	GSM1366741 1	19.00065276	18.90626607	18.83456008	18.81775304	18.81196729	18.7908401	18.74271124	18.71939662	18.66361169	18.63645652
32	GSM1366742 1	19.00065276	18.86076773	18.84259104	18.82348674	18.81775304	18.7908401	18.74271124	18.71939662	18.66361169	18.63645652
33	GSM1366743 1	19.00065276	18.88809108	18.85380342	18.81196729	18.80104894	18.75718509	18.74271124	18.71939662	18.66361169	18.63645652

Task 2: KNN Classification Model

- KNN: K-nearest neighbor

$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

- A simple machine learning algorithm which uses existing data to classify new data points.

- Algorithm uses existing data points and calculates the ‘Distance’ between the test points and existing data points

- The target is classified based on the minimum distance i.e. by finding “Closest Neighbour” to the test points

Task 2: Stored KNN Procedure

```
CREATE procedure [dbo].[usp_knn]
(@v1 float,@v2 float,@v3 float,@v4 float,@v5 float,@v6 float,@v7 float,@v8 float,@v9 float,@v10 float)
as
declare @target varchar(100) ,@temp float, @iter int , @r_target varchar(100), @dist float,@value1 float, @value2 float, @value3 float, @value4 float, @value5 float
declare @value6 float, @value7 float, @value8 float, @value9 float, @value10 float
set @dist=0
set @temp=0
set @iter=0
declare Cur cursor for select * from Genome1
open Cur
fetch next from Cur into @target,@value1,@value2,@value3,@value4,@value5,@value6,@value7,@value8,@value9,@value10

WHILE @@FETCH_STATUS = 0
    BEGIN
    set @temp=sqrt(square(@v1-@value1)+square(@v2-@value2)+square(@v3-@value3)+square(@v4-@value4)+square(@v5-@value5)+square(@v6-@value6)+square(@v7-@value7)+square(@v8-@value8)+square(@v9-@value9)+squa
        if @iter=0
          BEGIN
           set @dist=@temp
           set @r_target=@target
          END
        else
          BEGIN
             if @temp < @dist
             BEGIN
                set @dist=@temp
                set @r_target=@target
             END
         END
        set @iter=@iter+1
        FETCH NEXT FROM Cur into @target,@value1,@value2,@value3,@value4,@value5,@value6,@value7,@value8,@value9,@value10
    END

close Cur

select  @r_target as Classified_Target
```

Query Results

Execution of the stored procedure usp_knn:

exec dbo.usp_knn
19.00065276,18.93236743,18.89469309,18.8
7350665,18.857951,18.85380342,18.7753964
3,18.68931234,18.66361169,18.63645652

	Classified_Target
1	GSM1366726 1

Task 3: K-Means Clustering Model

- K-means clustering is one of the popular unsupervised machine learning algorithms
- The objective of K-means is to group similar data points together to form a fixed number (*k*) clusters in a dataset
- K-means algorithm identifies *k* number of centroids, and then allocates every data point to the nearest cluster corresponding to the centroids
- Then we average the data belonging to particular cluster to find the new centroid
- It halts creating and optimizing clusters when either:
    - The centroids have stabilized — there is no change in their value because the clustering has been successful.
    - The defined number of iterations has been achieved.

Task 3: K-Means Clustering

```
Create PROCEDURE [dbo].[usp_kmeans](@k1 float , @k2 float) AS BEGIN
declare @temp int ,@iter int, @sql nvarchar(1000),@curs_str nvarchar(1000),@min float,@label int , @target float,@cent1 float ,@cent2 float ,@dist1 float ,@dist2 float
declare @old_centroid1 float,@old_centroid2 float,@new_centroid1 float,@new_centroid2 float,@source varchar(50)
set @old_centroid1 = @k1
set @old_centroid2 = @k2
set @new_centroid1 = 0
set @new_centroid2 = 0
Update cluster_ set C1= @k1
Update cluster_ set C2= @k2
update cluster_ set D1=abs(Target1-C1)--This is to calculate K distances from K centroids
update cluster_ set D2=abs(Target1-C2)
while @old_centroid1 != @new_centroid1
begin
set @old_centroid1=@new_centroid1
set @old_centroid2=@new_centroid2
declare curs cursor for select * from cluster_
open curs
fetch next from curs into @target,@cent1,@cent2,@dist1,@dist2,@label
WHILE @@FETCH_STATUS = 0
   BEGIN
      if @dist1 < @dist2
        BEGIN
           update cluster_ set label=1 where Target1=@target
        END
      else
        BEGIN
           update cluster_ set label=2 where Target1=@target
        END
      fetch next from curs into @target,@cent1,@cent2,@dist1,@dist2,@label
    END
close curs
deallocate curs
set @new_centroid1=(select avg(Target1) from cluster_  where label=1)
set @new_centroid2=(select avg(Target1) from cluster_  where label=2)
update cluster_ set C1=@new_centroid1
update cluster_ set C2=@new_centroid2
end END
```

	Target1	C1	C2	D1	D2	label
1	188.22	187.9504	188.68875	0.219999999999999	0.680000000000007	1
2	188.39	187.9504	188.68875	0.389999999999986	0.510000000000019	1
3	188.76	187.9504	188.68875	0.759999999999991	0.140000000000015	2
4	189.03	187.9504	188.68875	1.03	0.129999999999995	2
5	187.7	187.9504	188.68875	0.300000000000011	1.20000000000002	1
6	187.79	187.9504	188.68875	0.210000000000008	1.11000000000001	1
7	187.75	187.9504	188.68875	0.25	1.15000000000001	1
8	187.27	187.9504	188.68875	0.72999999999999	1.63	1
9	188.37	187.9504	188.68875	0.370000000000005	0.530000000000001	1
10	188.52	187.9504	188.68875	0.52000000000001	0.379999999999995	2
11	188.58	187.9504	188.68875	0.580000000000013	0.319999999999993	2
12	188.28	187.9504	188.68875	0.280000000000001	0.620000000000005	1
13	188.42	187.9504	188.68875	0.419999999999987	0.480000000000018	1
14	188.34	187.9504	188.68875	0.340000000000003	0.560000000000002	1
15	187.87	187.9504	188.68875	0.129999999999995	1.03	1
16	188.24	187.9504	188.68875	0.240000000000009	0.659999999999997	1
17	187.85	187.9504	188.68875	0.150000000000006	1.05000000000001	1
18	187.78	187.9504	188.68875	0.219999999999999	1.12	1
19	187.87	187.9504	188.68875	0.129999999999995	1.03	1
20	188.9	187.9504	188.68875	0.900000000000006	0	2
21	187.98	187.9504	188.68875	0.0200000000000102	0.920000000000016	1
22	188.71	187.9504	188.68875	0.710000000000008	0.189999999999998	2
23	188.24	187.9504	188.68875	0.240000000000009	0.659999999999997	1
24	187.73	187.9504	188.68875	0.27000000000001	1.17000000000002	1
25	188.48	187.9504	188.68875	0.47999999999999	0.420000000000016	2
26	187.6	187.9504	188.68875	0.400000000000006	1.30000000000001	1
27	187.57	187.9504	188.68875	0.430000000000007	1.33000000000001	1
28	188.53	187.9504	188.68875	0.530000000000001	0.370000000000005	2
29	187.89	187.9504	188.68875	0.110000000000014	1.01000000000002	1
30	187.92	187.9504	188.68875	0.0800000000000125	0.980000000000018	1
31	187.92	187.9504	188.68875	0.0800000000000125	0.980000000000018	1
32	187.9	187.9504	188.68875	0.0999999999999943	1	1
33	187.87	187.9504	188.68875	0.129999999999995	1.03	1

RESULT:

exec dbo.usp_kmeans 188 ,188.9

Task 4: Recommendation System

- The data:
  - Using the instacart data
  - Order_products
  - Products

- Because the data is so large we used a subset of the data
  - Identified a large number of orders and grabbed all the products from those orders.

Task 4: Market Basket Table

- Create a lookup table with the structure: Product A, Product B, Frequency

- To achieve this we create two views
  - Order_products_combos
    - Creates combinations for all products within an order
  - Products_combos
    - Creates combinations for all products in the table

Order_products_combos

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `admin`@`%`
    SQL SECURITY DEFINER
VIEW `order_products_combos` AS
    SELECT
        `p1`.`product_id` AS `product_1`,
        `p2`.`product_id` AS `product_2`,
        `p1`.`order_id` AS `order_id`
    FROM
        (`order_products_subset` `p1`
        JOIN `order_products_subset` `p2`)
    WHERE
        ((`p1`.`product_id` < `p2`.`product_id`)
            AND (`p1`.`order_id` = `p2`.`order_id`))
```

reordered	add_to_cart_order	product_id	order_id
0	3	10246	1
1	2	11109	1
0	6	13176	1
1	8	22035	1
1	5	43633	1
0	7	47209	1
1	1	49302	1
0	4	49683	1

- Above is order_products table where order_id = 1.
- To the right is a preview of order_products_combos for order_id = 1

product_1	product_2	order_id
10246	11109	1
10246	13176	1
11109	13176	1
10246	22035	1
11109	22035	1
13176	22035	1
10246	43633	1
11109	43633	1
13176	43633	1
22035	43633	1
10246	47209	1
11109	47209	1
13176	47209	1
22035	47209	1
43633	47209	1
10246	49302	1

products_combos

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `admin`@`%`
    SQL SECURITY DEFINER
VIEW `products_combos` AS
    SELECT
        `p1`.`product_id` AS `product_id_1`,
        `p1`.`products_name` AS `product_1_name`,
        `p2`.`product_id` AS `product_id_2`,
        `p2`.`products_name` AS `product_2_name`
    FROM
        (`products_subset` `p1`
        JOIN `products_subset` `p2`)
    WHERE
        (`p1`.`product_id` < `p2`.`product_id`)
    ORDER BY `p1`.`product_id`
```

Reminder: The reason we are seeing product_id_2 starting at 10 is because we used a subset of data to complete the task.

product_id_1	product_1_name	product_id_2	product_2_name
1	Chocolate Sandwich Cookies	10	Sparkling Orange Juice & Prickly Pear Beverage
1	Chocolate Sandwich Cookies	34	Peanut Butter Cereal
1	Chocolate Sandwich Cookies	37	Noodle Soup Mix With Chicken Broth
1	Chocolate Sandwich Cookies	45	European Cucumber
1	Chocolate Sandwich Cookies	79	Wild Albacore Tuna No Salt Added
1	Chocolate Sandwich Cookies	99	Local Living Butter Lettuce
1	Chocolate Sandwich Cookies	117	Petit Suisse Fruit
1	Chocolate Sandwich Cookies	123	Sherry Reserve Vinegar
1	Chocolate Sandwich Cookies	130	Vanilla Milk Chocolate Almond Ice Cream Bars M...
1	Chocolate Sandwich Cookies	162	Organic Mini Homestyle Waffles
1	Chocolate Sandwich Cookies	195	Grade A Pasteurized 2% Milkfat Lowfat Cottage...
1	Chocolate Sandwich Cookies	196	Soda
1	Chocolate Sandwich Cookies	199	Herb Thyme Clamshell
1	Chocolate Sandwich Cookies	206	Roasted Vegetable Souffl+\xc2\xac
1	Chocolate Sandwich Cookies	210	Homemade Hot Arrabbiata Fra Diavolo Sauce
1	Chocolate Sandwich Cookies	260	Cantaloupe

Market Basket Table

- Created as a view using a join query.
- Joined both the order_products_combos and the product_combos tables.

```
SELECT
    pc.product_1_name as productA, pc.product_2_name as productB, count(*) as freq
FROM
    products_combos pc
JOIN
    order_products_combos opc
ON
    ((opc.product_1 = pc.product_id_1 AND opc.product_2 = pc.product_id_2)
    OR (opc.product_1 = pc.product_id_2 AND opc.product_2 = pc.product_id_1))
GROUP BY
    pc.product_id_1, pc.product_id_2;
```

productA	productB	freq
Organic Strawberries	Banana	11
Bag of Organic Bananas	Organic Hass Avocado	9
Bag of Organic Bananas	Organic Strawberries	8
Organic Baby Spinach	Banana	8
Banana	Limes	8
Organic Strawberries	Limes	7
Organic Baby Spinach	Limes	7
Organic Baby Spinach	Large Lemon	7
Organic Strawberries	Organic Baby Spinach	6
Organic Strawberries	Organic Hass Avocado	6
Organic Baby Arugula	Organic Baby Spinach	6
Organic Baby Spinach	Organic Grape Tomatoes	6
Banana	Honeycrisp Apple	6
Bag of Organic Bananas	Organic Baby Spinach	5
Bag of Organic Bananas	Organic Garlic	5
Bag of Organic Bananas	Organic Raspberries	5

Procedure: usp_recommender

```
Create PROCEDURE usp_recommender(IN productName VARCHAR(256))
    SELECT
        if(STRCMP(productA, productName), productA, productB)
                                        AS recommended_product
    FROM
        market_basket mb
    WHERE
        mb.productA = productName OR mb.productB = productName
    ORDER BY
        freq DESC
    LIMIT 3;
```

If statement: checks if product A = the product name passed. If so select the other product name.

Limit 3: selects the top 3 because we are ordering by frequency descending.

Usp_recommender: Results Confirmed

```
CALL usp_recommender("Banana");
```

recommended_product
Organic Strawberries
Organic Baby Spinach
Limes

Top 3 recommended products purchased with Bananas.

```
SELECT
    *
FROM
    market_basket
WHERE
    productA = 'Banana'
OR
    productB = 'Banana'
ORDER BY
    freq
DESC;
```

We can confirm these results are correct.

productA	productB	freq
Organic Strawberries	Banana	11
Organic Baby Spinach	Banana	8
Banana	Limes	8
Banana	Honeycrisp Apple	6
Strawberries	Banana	5
Banana	Fresh Cauliflower	5
Banana	Red Vine Tomato	5
Yellow Onions	Banana	4
Organic Baby Arugula	Banana	4
Red Peppers	Banana	4
Banana	Organic Blueberr...	4
Banana	Organic Grape T...	4
Banana	Honey Nut Chee...	4
Organic 2% Reduce...	Banana	3
Seedless Red Grapes	Banana	3
100% Whole Wheat ...	Banana	3

Thank You